PENERAPAN TANDA TANGAN DIGITAL PADA ARSIP *STREAM*

Brahmasta Adipradana – NIM : 13503082

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13082@students.if.itb.ac.idl

Abstrak

Tanda tangan digital saat ini merupakan sebuah komponen penting dalam sebuah dokumen digital, baik dokumen tersebut berupa arsip teks kaya *(rich text)*, gambar, maupun arsip multimedia seperti audio dan video. Kebutuhan atas tanda tangan ini semakin dikuatkan karena semakin berkembangnya teknologi internet, yang memungkinkan penyebaran dokumen secara mudah dan bebas.

Adapun kelebihan yang dapat diberikan dari tanda tangan digital adalah kemampuan untuk melakukan otentikasi atau verfikasi dari pemilik dokumen serta memfasilitasi anti penyangkalan (*non repudiation*). Verifikasi memberikan kepastian kepada seorang penerima dokumen bahwa dokumen yang dibacanya memang benar-benar dari pengirim yang ia ketahui. Sedangkan anti penyangkalan memungkinkan penyimpanan bukti oleh penerima jika pengirim melakukan sesuatu yang mengganggu penerima dan penerima ingin menuntut pengirim.

Stream adalah sekumpulan bit yang sangat panjang (mungkin tanpa batas) yang dikirimkan pengirim ke penerima.Saat ini penggunaan *stream* sudah sangat jamak pula. Bentuk arsip *stream* yang berbeda dengan arsip konvensional membutuhkan penanganan yang berbeda dalam hal skema tanda tangan digital.

Untuk itu, dalam makalah ini, akan dibahas metode pemberian tanda tangan digital untuk arsip *stream* yang meliputi dua jenis arsip *stream* yang ada, yaitu metode *off-line* untuk arsip *stream* yang terbatas, dan metode *online* untuk arsip stream yang tidak terbatas.

Selain itu pada makalah ini akan dibahas mengenai aspek keamanan, implementasi, dan contoh aplikasi dari metode-metode yang telah dibahas di atas.

Kata kunci*:* tanda tangan digital, arsip *stream*, solusi *off-line*, solusi *online*

1. Pendahuluan

Saat ini, tanda tangan digital sudah cukup jamak digunakan terhadap suatu arsip. Hal ini dikarenakan tanda tangan digital dapat menjamin autentikasi suatu arsip, yakni apakah arsip tersebut memang hasil karya pemiliknya atau mungkin sudah dimodifikasi pihak lain dalam proses penyampaiannya.

Proses autentikasi dengan menggunakan tanda tangan digital juga memiliki sebuah kelebihan, yaitu jika menggunakan pilihan dua macam kunci yaitu kunci publik dan kunci privat. Konsep ini memungkinkan sebuah arsip dapat diautentikasi oleh semua orang dengan kunci pubik, dan dapat dikunci hanya oleh pemiliknya, dengan kunci privat.

Namun, seiring dengan perkembangan jaman, muncullah sebuah jenis arsip yang merupakan arsip *stream*. Arsip ini membutuhkan penanganan yang berbeda untuk arsip biasa dalam pemberian tanda tangan digital. Sebab arsip ini digunakan secara parsial dan bisa jadi tidak diketahui kapan berakhirnya. Penerapan tanda tangan digital selama ini kebanyakan dilakukan pada arsip biasa yang diketahui ukurannya, sehingga penerapannya akan berbeda untuk arsip stream.

Mengapa arsip *stream* butuh menggunakan tanda tangan digital? Karena saat ini, arsip stream digunakan untuk membublikasikan hasil karya seperti video dan audio. Untuk mencegah penyalahgunaan, maka digunakan tanda tangan digital.

1.1. Tanda Tangan Digital

Bicara mengenai tanda tangan digital, teknologi ini muncul disebabkan kebutuhan dalam dunia kriptografi untuk beberapa hal sebagai berikut:

1. Kerahasiaan pesan (*confidentiality*)

2. Otentikasi (*authentication*)
3. Keaslian pesan (*data integrity*)
4. Anti penyangkalan (*non repudiation*)

Kerahasiaan pesan berarti sebuah pesan yang dikirimkan oleh suatu pihak yang bersifat rahasia harus bisa terjaga. Otentikasi bertujuan untuk melakukan validasi apakah pesan yang diterima benar-benar dikirim oleh pihak benar. Keaslian pesan berarti pesan tidak terganggu atau dimodifikasi dalam proses penyampaiannya ke penerima.Anti penyangkalan bertujuan untuk bisa membuktikan bahwa sebuah pesan itu dikirim oleh seseorang, dan orang tersebut tidak bisa menyangkal bahwa itu pesan miliknya.

Dalam kriptografi, kebutuhan akan kerahasiaan pesan dan keaslian pesan diselesaikan dengan enkripsi dan dekripsi. Untuk menjamin sebuah pesan tidak bisa dibaca oleh pihak yang tidak berwenang, dilakukan enkripsi terhadap pesan tersebut. Pihak yang berwenang dapat membuka pesan tersebut dengan dekripsi. Pesan yang telah dirusak selama perjalanan akan membuat hasil dekripsi menjadi salah, sehingga dapat diketahui keaslian pesan yang diterima atau tidak.

Di sisi lain, kebutuhan akan otentikasi dan anti penyangkalan diselesaikan dengan tanda tangan digital. Dengan konsep tanda tangan digital, kita dapat mengetahui siapa pengirim pesan (otentikasi), dan dapat membuktikan jika pengirim menyangkal telah mengirimkan pesan.

1.1.1. Konsep Tanda Tangan

Bicara mengenai tanda tangan digital tentu tidak bisa melepaskan dari istilah tanda tangan itu sendiri. Tanda tangan digital lahir mengikuti sifat-sifat yang dimiliki tanda tangan, yaitu :

1. Merupakan bukti yang otentik
2. Tidak dapat dilupakan
3. Tidak dapat dipindahkan untuk digunakan ulang
4. Dokumen yang telah ditandatangani tidak dapat diubah
5. Tidak dapat disangkal

Tanda tangan digital mengikuti sifat-sifat tanda tangan untuk diterapkan dalam arsip digital. Tanda tangan yang dimaksud di sini tentunya bukan tanda tangan biasa yang di-*scan* ke dalam format digital, tapi tanda tangan yang telah menerapkan konsep-konsep kriptografi.

Salah satu yang membedakan antara tanda tangan pada dokumen biasa dengan tanda tangan digital adalah tanda tangan pada dokumen sama untuk setiap dokumen, sedangkan tanda tangan digital berbeda pada setiap dokumen.

1.1.2. Metode Pemberian Tanda Tangan

Dalam memberikan tanda-tangan terhadap sebuah dokumen digital, dapat digunakan dua buah alternatif, yaitu :

1. Menggunakan enkripsi pesan
2. Menggunakan fungsi *hash* dan kriptografi kunci publik

Untuk alternatif menggunakan enkripsi pesan sendiri, terdapat dua alternatif, yaitu menggunakan kunci simetri atau kunci publik.

Jika menggunakan alternatif enkripsi pesan dan kunci simetri, metode ini sudah memberikan solusi untuk otentikasi pengirim dan keaslian pesan, sebab kunci hanya diketahui oleh pengirim dan penerima. Namun kekurangan dari metode ini adalah tidak adanya mekanisme anti penyangkalan. Untuk mengatasinya, pada metode ini biasa digunakan pihak ketiga yang dipercaya oleh pengirim dan penerima (*abitrase*).

Alternatif lain adalah dengan menggunakan enkripsi pesan dan kunci publik. Pada metode ini, jika yang dienkripsi merupakan keseluruhan pesan, maka pesan dienkripsi dengan kunci publik penerima, lalu nantinya penerima akan dapat membuka pesan tersebut dengan kunci privatnya. Namun jika ingin melakukan enkripsi tanda tangan, maka pesan dienkripsi dengan kunci privat pengirim, lalu dekripsi, dilakukan dengan kunci publik pengirim. Dengan cara ini, kerahasiaan pesan dan otentikasi dapat dilakukan sekaligus. Metode ini ditemukan oleh Diffie dan Hellman.

Adapun metode tersebut dapat dirumuskan sebagai berikut, untuk penandatanganan pesan, dirumuskan sebagai berikut:

$$S = E_{SK}(M)$$

Sedangkan untuk otentikasi pesan, dirumuskan sebagai berikut:

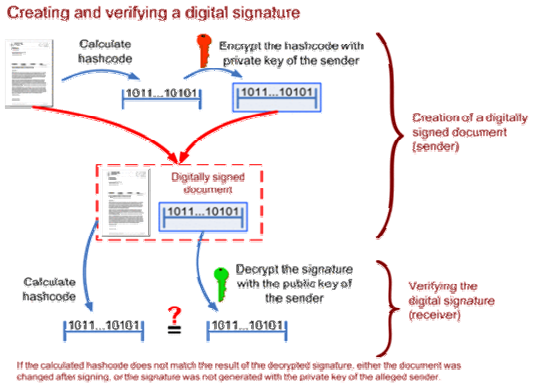
$$M = D_{PK}(S)$$

Keterangan :

SK = *secret key* = kunci privat pengirim
PK = *public key* = kunci publik pengirim
E = fungsi enkripsi
D = fungsi dekripsi
M = pesan semula
S = *signature* = hasil enkripsi pesan

Dengan algoritma kunci publik ini, maka tidak dibutuhkan lagi perantara untuk menjamin kerahasiaan kunci.

Selain alternative untuk menggunakan enkripsi pesan, terdapat alternatif kedua yaitu menggunakan fungsi *hash* dan kriptografi kunci publik. Alternatif ini muncul karena alternative pertama sering mengeluarkan fungsi ganda, yaitu kerahasiaan pesan dan otentikasi pesan.Pada beberapa kasus, seringkali yang dibutuhkan hanya otentikasi pesan. Kerahasiaan pesan tidak dibutuhkan. Untuk inilah digunakan alternatif kedua. Skema untuk alternatif kedua dapat dilihat pada gambar 1.



Gambar 1 Skema Gambar menggunakan hash dan kriptografi kunci publik

Seperti ditunjukkan pada gambar 1, untuk memberi tanda tangan pada suatu arsip, diberikan fungsi *hash* terhadap pesan yang ada di arsip tersebut sehingga menghasilkan *message digest*. *Message digest* ini nantinya akan dienkripsi dengan algoritma enkripsi tertentu dan menggunakan kunci privat pengirim. Hasil enkripsi ini nantinya akan dianggap sebagai *signature*. *Signature* ini nantinya akan dimasukkan ke dalam pesan sebelum akhirnya dikirimkan.

Untuk melakukan otentikasi atau verifikasi dari pesan, pada awalnya dilakukan dekripsi terhadap hasil *signature* dari pesan, dengan menggunakan kunci publik dari pengirim pesan. Jika hasil dekripsi sama dengan hasil fungsi *hash* terhadap pesan, maka berarti arsip tersebut valid. Jika tidak maka sebaliknya.

Algoritma enkripsi/dekripsi sendiri yang digunakan bermacam-macam. Namun kebanyakan digunakan RSA dan ElGamal.

Sementara itu, algoritma *hash* yang digunakan juga bermacam-macam, contohnya adalah SHA-1.

1.2. Kriptografi Kunci-Publik

Dalam pembahasan sebelumnya sering diungkit megenai kunci privat dan kunci publik. Kedua kunci tersebut merupakan komponen penting dalam kriptografi kunci-publik. Untuk lebih jelasnya, pada bagian ini akan dijelaskan mengenai ini.

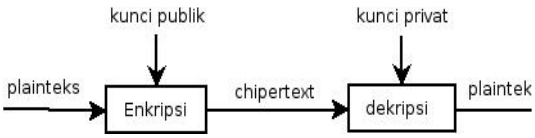
Kriptografi kunci publik muncul disebabkan atas kelemahan kriptografi kunci simetri yang membutuhkan sepasang kunci yang sama untuk enkripsi dan dekripsi, sehingga kedua belah pihak harus berkomunikasi terlebih dahulu dan saling mempercayai.

Konsep dan sistem kriptografi kunci publik ditemukan oleh Diffie dan Hellman, yang dipresentasikan pada tahun 1976.

Penggunaan tanda tangan digital yang baik adalah dengan menggunakan kriptografi kunci publik. Dalam metode ini digunakan dua jenis kunci, yaitu kunci privat dan kunci publik.

Kunci privat adalah kunci yang dimiliki oleh pemilik pesan. Kunci privat tidak untuk diberitahukan kepada orang lain. Dalam kriptografi kunci publik, kunci privat digunakan pemilik pesan untuk megenkripsi pesan.

Kunci publik adalah kunci dari pemilik pesan yang digunakan untuk publik. Kegunaan kunci ini adalah saat melakukan otentikasi pengirim. Jika dokumen yang diotentikasi dengan kunci publik menghasilkan nilai valid, maka dokumen tersebut benar-benar dimiliki oleh pengirim.



Gambar 2 Skema Kriptografi Kunci-Publik

Kelebihan dari kriptografi kunci publik ini adalah

1. Hanya kunci privat yang perlu dijaga kerahasiaannya oleh tiap entitas yang berkomunikasi

2. Pasangan kunci publik dan kunci privat tidak perlu diubah, bahkan dalam waktu yang panjang
3. Dapat digunakan untuk pengiriman kunci simetri
4. Beberapa algotitma kunci-publik dapat digunakan untuk memberi tanda tangan digital pada pesan.

Namun kriptografi kunci-publik juga memiliki kelemahan-kelemahan sebagai berikut

1. Enkripsi dan dekripsi lebih lambat daripada kriptografi kunci simetri, karena enkripsi dan dekripsi menggunakan bilangan yang besar dan melibatkan operasi perpangkatan yang besar.
2. Ukuran chiperteks lebih besar daripada plainteks, bahkan bisa mencapai empat kali ukuran plainteks.

1.3. Fungsi *Hash* Satu Arah

Seperti telah dijelaskan pada penjelasan mengenai tanda tangan digital, skema tanda tangan digital menggunakan fungsi *hash* satu arah.

Fungsi *hash* adalah fungsi yang menerima masukan string yang panjangnya sembarang dan mengkonversinya menjadi *string* keluaran yang panjangnya tetap. Umumnya berukuran jauh lebih kecil daripada ukuran *string* semula.

Fungsi *hash* menerima masukan *string* apa saja. Jika dimisalkan *string* sebagai sebuah pesan yang dilambangkan dengan M yang berukuran bebas, maka fungsi *hash* dapat dituliskan sebagai

$$h = H(M)$$

dengan *h* adalah keluaran fungsi *hash yang* disebut juga nilai *hash* (*hash value*) atau pesan-ringkas *(message digest)*, dan *H* adalah fungsi hash.

Seperti telah disebutkan sebelumnya, fungsi *hash* selalu menghasilkan nilai *hash* yang panjangnya sama. Misalkan diketahui sebuah fungsi *hash* dapat menghasilkan *string* dengan panjang 128 bit. Maka meski *string* masukan berupa teks dengan panjang sepuluh karakter atau sebuah string berisi sepuluh ribu karakter, semuanya akan menghasilkan keluaran sepanjang 128 bit.

Fungsi *hash* satu arah adalah fungsi *hash* yang bekerja dalam satu arah, yaitu pesan dapat diubah menjadi *message digest*, tapi *message digest* tersebut tidak dapat diubah menjadi pesan.

Hal yang menarik dari fungsi *hash* satu arah ini adalah setiap pesan akan menghasilkan nilai *hash* yang berbeda. Hal inilah yang dimanfaatkan dalam konsep tanda tangan digital.

Sifat-sifat fungsi *hash* satu arah adalah sebagai berikut[SCH96]:

1. Fungsi *H* diterapkan pada blok data berukuran berapa saja
2. *H* menghasilkan nilai *h* dengan panjang tetap
3. *H(x)* mudah dihitung untuk setiap nilai *x* yang diberikan
4. Untuk setiap nilai *h* yang diberikan, tidak mungkin menemuukan *x* sedemikian sehingga *H(x) = h*. Hal inilah yang menyebabkan fungsi ini dinamakan fungsi satu arah
5. Untuk setiap nilai *x* yang diberikan, tidak mungkin mencari $y \neq x$ sedemikian sehingga *H(y) = H(x)*
6. Tidak mungkin secara komputasi mencari pasangan *x* dan *y* sedemikian sehingga *H(x) = H(y)*

Sebuah fungsi *hash* dianggap tidak aman jika

1. Secara komputasi dimungkinkan menemukan pesan yang bersesuaian dengan pesan ringkasnya
2. Terjadi kolisi *(collision)*, yaitu terdapat beberapa pesan berbeda yang memiliki pesan ringkas yang sama.

Dalam pembahasan selanjutnya akan diungkin mengenai fungsi *collision-resistant hash*. Fungsi *hash* yang dimaksud di sini adalah fungsi yang memenuhi poin nomor 2 di atas.

Saat ini ada beberapa fungsi *hash* yang banyak digunakan, yang paling populer adalah MD5 dan SHA-1. Keduanya merupakan fungsi *collision resistant hash* Selain itu, terdapat juga MD2, MD4, RIPMEND, WHIRPOOL, dan lain-lain.

1.4. MAC

Dalam pembahasan berikut juga akan menyinggung mengenai MAC. MAC juga merupakan sebuah fungsi satu arah, tapi fungsi ini menggunakan kunci rahasia (kunci privat/*secret key*) dalam pembangkitan nilai *hash*. MAC disebut juga sebagai *keyed hash function* atau *key-dependent one-way hash function*. MAC memiliki sifat dan properti yang sama seperti fungsi *hash* satu arah yang telah dibahas sebelumnya, hanya saja ditambahkan komponen kunci.

Secara matematis, MAC dapat dinyatakan sebagai

$$MAC = C_K(M)$$

dengan *MAC* adalah nilai *hash*, *C* adalah fungsi *hash* atau algoritma *MAC* dan *K* adalah kunci rahasia.

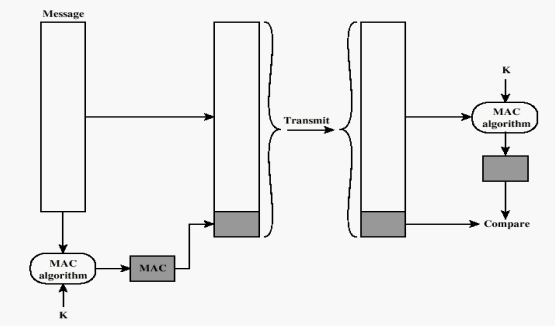


Figure 3.1 Message Authentication Using a Message Authentication Code (MAC)

Gambar 3: Skema autentikasi dengan menggunakan MAC

Pengguanaan MAC adalah untuk otentikasi pesan tanpa perlu merahasiakan atau mengenkripsi pesan. Contohnya adalah penjagaan otentikasi arsip yang digunakan oleh dua atau lebih pengguna. Selain itu MAC juga digunakan untuk menjamin integritas (keaslian) isi arsip terhadap perubahan, misalnya disebabkan karena virus.

1.5. Arsip *Stream*

Stream adalah sekumpulan bit yang sangat panjang (mungkin tanpa batas) yang dikirimkan pengirim ke penerima. *Stream* selalu dikirim dalam *rate* yang diisepakati antara pengirim dan penerima, atau ada sebuah protokol *demand-response* di mana setiap penerima secara berulang-ulang mengirim permintaan kepada pengirim terhadap data yang terbatas jumlahnya. Kelebihan utama dari *stream* bila dibandingkan dengan pesan biasa adalah *stream* dapat dikonsumsi tanpa harus menunggu arsip peuh dikirimkan, dan penerima tidak perlu menyimpan semua data yang dikirimkan untuk mengkonsumsi.

Contoh penggunaan stream yang umum adalah *stream* video dan audio yang berada di situs-situs internet. Arsip-arsip tersebut dapat didengarkan atau ditonton tanpa harus menyelesaikan pengunduhan arsip tersebut. Contoh lain adalah siaran televesi yang menggunakan satelit atau kabel. Siaran tersebut berlangsung terus-menerus dan harus segera ditayangkan tepat pada saat diterima. Tidak hanya untuk arsip video dan audio, *stream* juga digunakan untuk data seperti berita, *stock market quotes*, dan lain-lain. Internet dan industri pertelevisian juga menyediakan contoh lain, yaitu *applets*. *Applets* kebanyakan merupakan sebuah program besar yang diorganisasikan menjadi beberapa modul. Mesin pengguna awalnya mengunduh (*download*) dan mengekseskusi modul awal. Setelah berjalan, program tersebut akan mengunduh modul-modul yang dibutuhkan dan mengeksekusinya. Modul yang sudah tidak digunakan juga akan dibuang dari mesin pengguna. Struktur *applet* ini dibagi menhadi dua faktor, faktor pertama adalah jumlah storage yang tersedia pada mesin pengguna dapat terbatas. Faktor kedua adalah *bandwith* yang dapat terbatas. Appllet harus didesain agar tetap berjalan baik dengan dua faktor tersebut

Dari informasi di atas dapat dilihat bahwa arsip *stream* sangat berbeda jika dibandingkan dengan arsip biasa. Arsip jenis ini tidak pernah disimpan secara penuh di dalam mesin penerima, sebab arsip ini akan langsung dikonsumsi.

2. Permasalahan

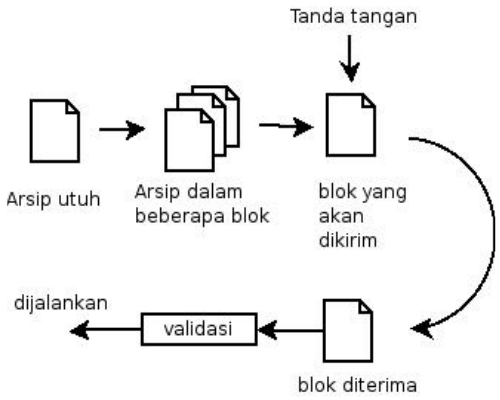
Jika melihat dari metode yang diterapkan untuk pemberian tanda tangan digital, metode ini tidak cocok untuk digunakan langsung ke arsip *stream*. Hal ini disebabkan arsip *stream* biasanya sangat panjang dan bahkan mungkin tidak akan selesai (misal: siaran berita), sehingga akan tidak mungkin jika penerima harus menunggu agar semua arsip dapat diunduh secara lengjap. Bahkan jika arsip dapat diunduh dengan lengkap, kelebihan arsip stream untuk dapat memperlihatkan data yang tidak lengkap akan hilang hanya dikarenakan pemasangan tanda tangan digital.

Permasalahan inilah yang muncul untuk arsip *stream.* Arsip jenis ini kebanyakan adalah arsip video dan audio yang tetap harus terjaga informasi penciptanya. Solusi yang mungkin untuk penjagaan informasi tersebut adalah dengan tanda tangan digital. Tapi bagaimana penerapan tanda tangan digital pada arsip *stream*?

Penerapan ini tentunya akan berbeda dengan penerapan pada arsip konvensional. Untuk itu pada pembahasan berikut disajikan beberapa alternatif dari penerapan tanda tangan tersebut.

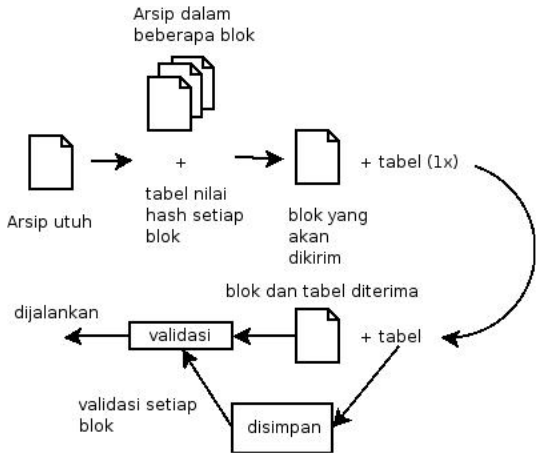
3. Alternatif Solusi

Untuk menyelesaikan masalah seperti yang telah dijelaskan sebelumnya, terdapat beberapa alternatif solusi yang dapat diberikan [GEN97].



Gambar 4 Alternatif pertama solusi tanda tangan digital

Alternatif pertama adalah membagi *stream* menjadi beberapa blok. Setiap blok tersebut diberikan tanda tangan digital oleh pengirim. Penerima nantinya akan menerima pesan tersebut dan melakukan verifikasi untuk setiap blok sebelum mengkonsumsinya. Solusi ini juga berlaku jika *stream* tidak terbatas. Sayangnya biaya komputasi untuk melakukan pemberian dan verifikasi tanda tangan digital saat ini masih terhitung mahal, sehingga jika dilakukan berulang-ulang akan membebani sistem. Selain itu, metode ini kemungkinan akan menimbulkan *bottleneck* karena waktu pengiriman dari pengirim ke penerima lebih sedikit dibandingkan proses verifikasi yang dilakukan penerima.

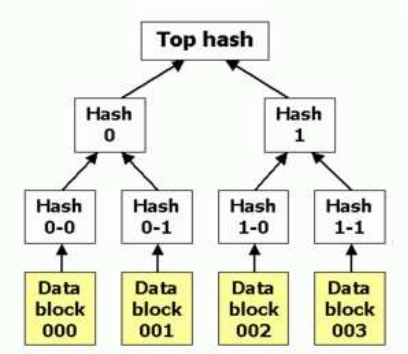


Gambar 5 Alternatif kedua solusi tanda tangan digital

Alternatif lainnya adalah dengan tetap membagi *stream* menjadi sejumlah blok, tapi dengan membuat tabel yang mendaftarkan hasil fungsi *hash* dari setiap blok dan menyimpan tabel ini. Alternatif ini hanya berlaku untuk arsip yang ukurannya diketahui. Jika penerima meminta untuk mengautentikasi *stream*, pengirim mengirimkan tabel tersebut baru dilanjutkan oleh *stream*. Penerima akan menerima tabel terlebih dahulu dan menyimpannya, lalu setiap blok yang diterima akan diverifikasi dengan melihat apakah nilai *hash*-nya cocok untuk setiap blok.

Masalah dari alternatif ini adalah kemungkinan dibutuhkan sebuah tabel dengan ukuran yang sangat besar, sehingga menyulitkan dalam penyimpanan tabel itu sendiri. Selain itu, masalah lain adalah jika tabel terlalu besar, proses pengunduhan tabel akan memakan waktu lama, dan akan menyebabkan terjadinya *delay* dalam melihat arsip *stream* yang cukup lama.

Kedua solusi di atas bisa dimodifikasi dengan menggunakan *authentication tree*. Setiap blok diletakkan sebagai daun (*leaf*) dari sebuah pohon binar dan setiap simpul (*node*) menyimpan nilai *hash* dari anaknya. Dengan cara ini, pengirim hanya perlu memberi tanda tangan dan mengirimkan akar dari pohon ini. Untuk melakukan otentikasi setiap blok, pengirim hanya perlu mengerimkan seluruh *path* otentikasi kepada penerima. Hal ini berarti jika stream memiliki blok sejumlah k, maka informasi otentikasi yang diasosikan dengan setiap blok adalah *0(log k)*.



Gambar 6 Alternatif ketiga dengan menggunakan hash tree

Solusi ini bisa juga disebut dengan *hash* tree. Solusi terakhir mengeliminasi masalah-masalah yang ada sebelumnya. Ide ini berlaku untuk *stream* yang terbatas dan tidak terbatas. Komputasi untuk membuat tanda tangan digital pun hanya dilakukan sekali. Selain itu tidak ada tabel berukuran besar yang harus disimpan, dan informasi otentikasi yang terasosiasi dengan setiap blok tidak tergantung dengan ukuran *stream*.

3.1. Anti Penyangkalan Dalam Solusi

Bagaimana dengan isu anti penyangkalan? Jika penerima hanya tertarik untuk mengetahui identitas dari pengirim, sebuah solusi yang didasarkan oleh MAC akan mencukupi. Meskipun setiap pengirim dan penerima saling berbagi kunci, *stream* dapat diotentikasi blok per blok dengan menggunakan komputasi MAC di atasnya. Sejak MAC selalu lebih cepat dari tanda tangan digital untuk dikomputasi dan diverifikasi, solusi ini akan tidak mendatangkan biaya komputasi yang sama denga solusi berbasi tanda tangan digital.

Namun, sebuah pendekatan berbasis MAC tidak akan memiliki properti anti penyangkalan. Kita akan dipusingkan karena kita membutuhkan properti ini untuk solusi. Selain itu, untuk menjadikan properti ini berarti dalam *stream* kita butuh agar setiap *prefix* dari *stream* menjadi anti peyangkalan. Misalkan $B = B_1, B_2, ...$ di mana setiap B_i adalah sebuah blok, kita membutuhkan setiap *prefix* $B_i = B_1,..,B_i$ menjadi anti penyangkalan. Aturan ini muncul disebabkan solusi di mana setiap pengirim harus mengirimkan sebuah MAC untuk tiap blok dan memberi tanda tangan untuk semua *stream* pada akhirnya.

Hal ini dilakukan untuk mencegah pengirim melakukan interupsi transmisi dari *stream* sebelum properti anti penyangkalan diterima. Selain itu, hal ini memberi garansi ke penerima. Misalkan saat penerima menggunakan *applet* terjadi kerusakan dari sistem. Maka penerima menghentikan proses *transfer* untuk mencegah kerusakan lebih lanjut, tapi pada waktu yang bersamaan, penerima juga masih butuh sejumlah bukti bahwa sebagian *stream* yang telah dia terima dari seorang pengirim yang diketahui.

Solusi lain adalah dengan membagi *stream* menjadi beberapa blok dan memasukkan informasi otentikasi pada *stream* itu sendiri. Informasi otentikasi pada blok ke-i akan bisa digunakan untuk mengautentikasi blok ke-(i+1). Dengan cara ini, pengirim hanya perlu memberi tanda tangan pada blok pertama lalu porperti dari tanda tangan ini menyebarkan ke *stream* sisanya dengan informasi otentikasi tersebut. Masalah utama dari kasus ini adalah bagaimana melakukan otentikasi dari blok internal menjadi cepat. Untuk ini akan dilihat dari dua kasus.

Kasus pertama adalah *stream* yang dialirkan terbatas dan diketahui keseluruhannya oleh pemberi tanda tangan. Dalam kasus ini sebuah komputasi *hash* akan mencukupi untuk mengautentikasi seluruh blok internal. Idenya adalah untuk menyimpan setiap blok nilai *hash* dari blok selanjutnya.

Kasus kedua adalah *stream* yang dialirkan tidak diketahui batasnya. Dalam kasus ini solusi yang diberikan menjadi kurang optimal karena membutuhkan sekumpulan komputasi *hash* untuk mengotentikasi sebuah blok, meskipun tergantung dengan mekanisme penyimpanan hasil komputasi *hash* bisa dilunasi dalam panjang dari blok. Ukuran dari informasi otentikasi juga merupakan isu dalam kasus ini. Idenya adalah bagaimana menggunakan skema tanda tangan cepat satu kali untuk mengotentikasi blok-blok internal, sehingga blok ke-i akan mengandung sebuah kunci publik yang bisa digunakan sekali dan tanda tangan yang digunakan sekali untuk menghargai kunci yang ada pada blok ke-(i-1). Tanda tangan ini mengautentikasi tidak hanya blok *stream* tapi juga kunci sekali pakai yang disimpan di sini.

4. Prasyarat

Dalam menerapkan solusi dari pemberian tanda tangan digital pada arsip *stream*, dibutuhkan beberapa prasyarat sebagai berikut

4.1. Fungsi *Hash Collision-Resistant*

Seperti telah dijelaskan sebelumnya, Fungsi ini merupakan fungsi *hash* yang memetakan sebuah *string* panjang menjadi sebuah elemen yang jumlahnya *fixed*. Sebuah fungsi dapat dikatakan sebagi fungsi *hash Collision-Resistant* jika setiap *poynomial tim algorithm* yang diberikan sebagai input H(x) dalam beberapa nilai x akan menghasilkan nilai yang sama, dengan probabilitas yang sangat kecil, dengan kata lain, tidak terdapat beberapa pesan berbeda yang memiliki pesan ringkas yang sama. Beberapa contoh dari fungsi ini adalah MD5 dan SHA-1.

4.2. Skema Tanda Tangan

Skema tanda tangan adalah *triplet (G,S,V)* dari *probabilistic polynomial-time algorithms* yang memenuhi properti-properti sebagai berikut.

- *G* adalah algoritma pembangkit kunci. Dengan input 1n, akan menghasilkan sepasang *(S.K, P.K)* □ *{0,1}*^{2n}. *S.K.* merupakan singkatan dari *secret key* (kunci privat) dan *P.K.* adalah *public key* (kunci publik).
- *S* adalah algoritma penandatanganan, dengan input pesan P dan dengan kunci privat a menghasilkan *signature σ*.
- V adalah algoritma verifikasi. Setiap *(P.K, S.K.) = G(1*^{n}*)* dan *σ = S(SK,P)* maka terdapat *V(PK,σ,M) = 1*

4.3. Tanda Tangan *Stream*

Stream didefinisikan sebagai sekumpulan blok *B = B1, B2, ...* di mana setiap *Bi* □ *{0,1}*^{c} untuk setiap konstanta *c*.

Dalam solusi, permasalahan dibagi atas dua kasus. Dalam kasus pertama diasumsikan *stream* terbatas dan diketahui panjangnya oleh pengirim. Kasus ini disebut kasus *off-line*. Sementara kasus kedua adalah *stream* tidak terbatas atau tidak diketahui kapan berakhirnya, disebut kasus *online*.

5. Penjabaran Solusi

Dalam penjabaran solusi, akan dibedakan antara solusi *online* dan solusi *off-line*

5.1. Solusi *Off-line*

Untuk *stream* yang terbatas, *stream* dibagi secara logikal menjadi sekumpulan blok yang berukuran c. Penerima memiliki sebuah *buffer* dengan ukuran c. Penerima menerima tanda tangan dalam bentuk 20 *byte* karakter hasil *hash* dari blok yang pertama. Setelah veriifikasi dari tanda tangan, penerima mengetahui apa *hash* dari blok pertama seharusnya dan mulai menerima *stream* lalu mengkomputasi *hash* blok per blok. Saat menerima blok pertama, dilakukan pemeriksaan *hash* dibandingkan dengan tanda tangan yang telah diverifikasi. Jika cocok, blok tersebut dimainkan atau digunakan. Jika tidak, maka blok akan dibuang sekaligus *stream* yang diunduh.

Jika verifikasi berhasil, blok lain akan dilanjutkan untuk diverifikasi. Faktor kunci dalam poin ini adalah blok pertama mengandung 20 *byte* nilai *hash* dari blok kedua, blok kedua mengandung 20 *byte* nilai *hash* dari blok ketiga, dan seterusnya. Jadi setelah pemeriksaan tanda tangan pada awal, pemeriksaan yang dilakujan hanya memeriksa nilai *hash* dari tiap blok.

Secara lebih detail, solusi ini dijelaskan sebagai berikut [GEN97]. Misalkan *(G,S,V)* adalah skema tanda tangan. Pengirim memiliki sepasang kunci publik dan kunci privat. *{SK,PK} = G(1*^{n}*)* dari skema tanda tangan. H adalah fungsi *collision resistant hash*. Jika *stream* awal adalah

$$B = B_1, B_2, ..., B_k$$

dan hasil *stream* hasilnya adalah

$$B' = B'_0, B'_1, B'_2, ..., B'_k$$

pemrosesan dikerjakan dengan *backward* dari *stream* asalnya sebagai

$$B'_k = <B_k,00..0>$$

$$B'_i = <B_i, H(B'_{i+1})> \text{ untuk setiap } i = 1,..., k\text{-}1$$

$$B'_0 = <H(B'_1,k), S(SK, H(B'_1,k))>$$

Dari sisi pengirim, mengkomputasi tanda tangan dan memasukkan nilai *hash* membutuhkan sebuah *backwards pass* dalam *stream*, dikarenakan *stream* diketahui panjangnya. Di atas juga diperlihatkan bahwa blok pertama B'_0 dari *stream* yang dimasukkan mengandumh panjang dari *stream* yaitu k.

Penerima melakukan verfikasi sebagai berikut. Pada saat menerima blok *B'i = <Bi,A0,>* penerima memeriksa apakah

$$V(PK, A_0,B) = 1$$

dan mengekstraksi panjang k dari *stream*. Lalu menerima *B'i = <Bi, Ai>* untuk *(1≤i≤k)* penerima menerima Bi jika

$$H(B'_i) = A_{i-1}$$

Selanjutnya penerima harus melakukan komputasi untuk sebuah operasi konci publik di awal, dan hanya satu fungsi *hash* per blok. Tidak dibutuhkan tabel yang besar di dalam memori.

5.2. Solusi On*line*

Seperti telah dijelaskan sebelumnya, pada *stream* tidak terbatas pengirim tidak tahu panjang *stream* tersebut. Untuk memfasilitasi penandatanganan dalam kasus ini, digunakan model tanda tangan khusus, yang lebih cepat untuk dikomputasi dan diverifikasi daripada tanda tangan biasa karena berbasis fungsi satu arah dan tidak butuh fungsi *trapdoor*. Fungsi

seperti DES atau SHA-1 lebih efiisien daripadafungsi *trapdoor* terkaan seperti RSA. Walaubagaimanapun, skema ini tidak bisa digunakan untuk memberi tanda tangan sejumlah besar pesan tapi sekumpulan pesan awal (biasanya berjumlah satu).

Dalam kasus ini, akan digunakan skema *one-time signature*. Skema ini memungkinkan tanda tangan hanya pada sebuah pesan dengan menggunakan sekumpulan informasi privat dan publik. Salah satu keuntungan dari skema ini adalah kecepatannya. Implementasi efisien dari skema ini adalah Merkle Tree Signature Scheme yang tidak butuh pasangan kunci baru setiap pesan.

Dalam kasus ini *stream* juga dibagi menjadi beberapa blok. Awalnya pengirim mengirimkan kunci publik *one-time signature scheme*.Lalu ia mengirimkan blok pertama bersamaan dengan *one-time signature* dalam nilai *hash* berdasarkan pada kunci publik satu kali yang dikirim di blok sebelumnya. Blok pertama juga mengandung kunci *one-time* publik baru yang digunakan untuk memverifikasi tanda tangan pada blok kedua dan struktur ini terus berulang dalam blok-blok berikutnya.

Dalam lebih detailnya, misalkan kita ketahui *(G,S,V)* adalah skema tanda tangan digital dan *(g,s,v) one time signature scheme*. Dengan H adalah funsgi *hash collision resistant*, pengirimnya memiliki sepasang kunci $(SK,PK) = G(1^n)$, maka:

$$B = B_1, B_2, ...$$

adalah *stream* asli yang diasumsikan tidak terbatas, dan

$$B' = B'_0, B'_1, B'_2, ...$$

adalah *stream* yang telah diberi tandatangan. Setiap $i \geq 1$ merupakan $(sk_i,pk_i) = g(1^n)$ keluaran dari sebuah eksekusi algoritma *g*, maka

$$B'_0 = <pk_0,S(SK,pk_0)>$$

Dikarenakan kunci publik *one time signature* selalu pendek, maka mereka tidak perlu diberikan fungsi *hash* sebelum diberikan.

$$B'_i = < B_i, pk_i,s(sk_{i-1},H(B_i,pki))> \text{ untuk } i \geq 1$$

Dapat dilihat sebagian dari tanda tangan biasa pada blok pertama. Tanda tangan berikut merukpakan satu kali penggunaan, yang memungkinkan komputasi lebih cepat, termasuk pembangkitan kunci.

Penerima menerima *stream* sebagai berikut. Sewaktu menerima $B'_0 = <pk_0,A_0>$ penerima akan memeriksa apakah

$$V(PK,A_0,pk_0) = 1$$

dan pada saat menerima B1, ia akan memeriksa apakah

$$v(pk_{i-1},A_i,H(B_i,pk_i)) = 1$$

Jika salah satu dari pemeriksaan ini gagal, penerima akan berhenti menerima *stream*. Lalu penerima harus mengkomputasi sebuah kunci publik pada awal, lalu hanya *one-time signature* per blok.

6. Isu Keamanan

Untuk isu mengenai keamanan dari kedua metode ini, akan dibahas mengenai bukti-bukti yang telah dilampirkan [GEN97]

6.1. Untuk Kasus *Stream* Terbatas

Misalkan *(G',S',V')* adalah tanda tangan *stream* yang didefinisikan pada bagian sebelumnya. *(G,S,V)* adalah skema tanda tangan "*regular*" dan H adalah fungsi *hash* yang digunakan pada saat konstruksi. Maka berlaku teorema berikut

Teorema 1 Jika *(G,S,V)* adalah skema tanda tangan yang aman dan H adalah fungsi *hash collision-resistant* maka skema tanda tangan *(G',S',V')* juga aman.

6.2. Untuk Kasus *Stream* Tidak Terbatas

Untuk kasus stream yang tidak terbatas, dimisalkan *(G',S',V')* adalah skema tanda tangan yang didefinisikan pada bagian sebelumnya. *(G,S,V)* adalah skema tanda tangan "*regular*" dan H adalah fungsi *hash* yang digunakan pada saat konstruksi. Maka berlaku teorema berikut.

Teorema 2 Jika *(G,S,V)* dan *(g,s,v)* merupakan skema tanda tangan yang aman dan H adalah fungsi *hash collision-resistant* maka hasil skema tanda tangan *(G',S',V')* juga aman

Kedua teorema ini dijelaskan pembuktiannya pada [GEN97] dan telah membuktikan bahwa penyelesaian dengan metode ini akan memberikan keamanan.

7. Isu Implementasi

Untuk isu yang terkait masalah implementasi [GEN97] juga dijelaskan mengenai alasan pemilihan *one time signature scheme*, anti

penyangkalan, *hybrid schemes*, dan *probabilistic one-time signatures*.

7.1 *One Time Signature Scheme*

Dalam memilih skema tanda tangan digital, parameter yang dilihat adalah panjang tanda tangan waktu yang dibutuhkan untuk verifikasi. Di dalam solusi yang telah dijelaskan sebelumnya, parameter-parameter ini menentukan kebutuhan yang dapat menyebabkan konflik. Misalnya jika kita menginginkan tanda tangan yang pendek, dibutuhkan waktu verifikasi yang lama, sebaliknya tanda tangan yang panjang lebih cepat diverifikasi. Meskipun verifikasi seharusnya cukup cepat untuk memungkinkan penerima mengkonsumsi blok-blok *stream* dengan rating inpput yang telah didefinisikan. Pada waktu yang sama, dengan dimasukkannya tanda tangan di dalam *stream*, ukuran yang kecil menjadi penting sehingga tidak mempengaruhi ukuran *stream*.

Skema o*ne-time signature* merupakan algoritma yang dapat mengkompromikan hal ini. Skema dibuat berdasarkan atas fungsi satu arah F pada domain D, yang menggunakan fungsi *hash collision resistant* H. Skema ini memungkinkan pembubuhan pada sebuah pesan m-*bit.* Skema ini dijelaskan sebagai berikut.

Pembangkitan Kunci

Pilih $m + log\ m$ elemen dalam D, jadikan menjadi $a_1,.., a_{m+logm}$. Ini adalah kunci rahasia. Kunci publiknya adalah:

$$pk = H(F(a_1), ... , F(a_{m+logm}))$$

Algoritma Penandatanganan

Jika M adalah pesan yang akan ditandatangan, masukkan M ke dalam representasi biner dari angka-angka representasi biner M kosong, dan ini disebut M' sebagai *binary string* hasil. Tanda tangan M adalah s1,...,sm dengan si = a, jika bit ke-i dari M adalah 1 maka s1 = F(ai).

Algoritma Verfikasi

Algoritma verifikasinya adalah sebagai berikut

$$H(t_1, ... ,t_{m+logm}) = pk$$

di mana $t_i = s_i$ jika bit ke-i dari M' adalah 0, jika tidak $t_i = F(s_i)$

Keamanan

Secara intuitif skema ini aman dikarenakan tidak mungkin untuk mengubah 0 menjadi 1 dalam representasi biner dari pesan *M* tanpa harus melakukan invers fungsi *F*. Perbuahan dari 1 menjadi 0 mungkin dilakukan, tapi dapat meningkatkan jumlah 0 pada representasi biner *M* menyebabkan sebuah bit membalik dari ke 1 pada sekumpulan *log m* bit dari *M'* bagian terakhir, dan juga menyebabkan penyerang menginversi fungsi *F*.

Parameter

Skema ini memiliki panjang tanda tangan $|D|(m+log\ m)$ di mana |D| adalah sekumpulan bit yang dibutuhkan untuk merepresentasikan elemen D. Penerima harus menghitung satu komputasi *hash* H dan rata-rata $(m+log\ m)/2$ komputasi dari F.

Secara praktis kita bisa mengasumsikan bahwa F memetakan string 64 bit ke *long string*. Sejak *collision-resistance* tidak dibutuhkan dari F kita dapat mempercayai bahwa parameter ini cukup. Perkriaan fungsi F yang baik dapat dikonstruksi secara mudah dari blok *chiper* seperti DES dari fungsi *hash* seperti MD5 atau SHA-1. H juga dapat dinstansiasi ke MD5 atau SHA-1. Secara umum kita bisa mengasumsikan m mejadi 128 atau 160 jika pesan yang diberikan terlebih dahulu di-hash dengan menggunakan MD5 atau SHA-1.

Implementasi SHA-1 menghasilkan tanda tangan dengan panjang 1344 byte. Penerima harus mengkomputasi F sekitar 84 kali rata-rata. Sementara pada MD5 jumlahnya menjadi 1080 byte dan 68 komputasi. Jika digunakan skema offline *(stream* terbatas*)* kita juga perlu menambahkan 16 byte untuk memasukkan kunci publik dalam *stream*.

7.2 Anti Penyangkalan

Dalam kasus pembantahan secara legal mengenai isi dari *stream* yang ditandatangan, penerima harus membawa sesuatu sebagai bukti. Jika penerima menyimpan seluruh stream tidak ada masalah. Tapi dalam beberapa kasus, misalkan karena keterbatasan memori di komputer, data *stream* dibuang setelah dikonsumsi. Masalah yang muncul adalah bagaimana penerima bisa melindungi dirinya untuk pengajuan kasus sebagai bukti.

Dalam kasus *stream* terbatas, misalnya diasumsikan blok terakhir dari *stream* yang ditandatangan memiliki nilai khusus untuk nilai *hash*. Contohnya adalah semuanya 0. Penerima hanya butuh menyimpan blok yang pertama. Dengan ini ia bisa membuktikan bahwa dia menerima sesuatu dari pengirim. Selanjutnya adalah tinggal memaksa pengirim menyusun

ulang keseluruhan *stream* yang cocok blok awalnya dan berakhir dengan blok terakhir yang memiliki nilai *hash* khusus tersebut.

Untuk kasus *stream* yang tidak terbatas, setidaknya penerima harus menyimpan blok pertama dan semua *one-time signature* lalu mengharuskan pengirim mengkonstruksi ulang seluruh *stream*. Namun dalam praktisnya, hal yang disimpan masih tetap terlalu banyak. Sebuah *stream* online dapat dipecah menjadi beberapa bagian besar yang merepresentasikan unit logis, misalnya program TV atau *live broadcast* dari game. Idenya adalah ketika sebuah unit logis dipilih, sebuah *upper bound* dari jumlah total blok bisa diestimasi dan semua kunci sekali pakai yang dibutuhkan untuk bagian besar bisa dikomputasi oleh pengirim. Tambahan, dengan menggunakan teknik *offline* ini, pengirim bisa mengkomputasi nilai *hash* tunggal yang saat diberi bisa digunakan untuk mengautentikasi setiap kunci publik sekali pakai jika bisa dikirimkan sebagai sekumpulan kunci, di mana satu kunci untuk setiap blok *stream*. Skema *online* bekerja sebagaimana mestinya. Pada awalnya pengirim mengirim tanda tangan digital dalam nilai hash yang bisa digunakan untuk mengautentikasi sebuah stream dari kunci publik sekali pakai. Stream dari sebuah kunci sekali pakai lalu dimasukkan ke dalam data *stream* online dan data *stream* tersebut diautentikasi dengan *one time signature* didasarkan oleh kunci sekali pakai dimana setiap *one time signature* adalah sebuah *hash* dari semua data yang dikirimkan dalam *stream*. Cara ini penerima hanya butuh untuk menyimpan tanda tangan digital *regular* dan *one time signature* terakhir yang telah diterima dan diverifikasi. Untuk tujuan anti penyangkalan, didasarkan atas tanda tangan digital regular, pengirim bisa dipaksa untuk memperlihatkan seluruh *stream* atau kunci publik sekali pakai dan *one time signature* valid terakhir yang disimpan sama penerima. Pengrim bisa diminta untuk memproduksi sebuah data *stream* dengan nilai *hash* yang sama sebagaimana dibubuthkan pada *one time signature* terakhir yang disimpan oleh penerima.

7.3. *Hybrid Schemes*

Dalam skema online, panjang dari informasi otentikasi yang tersimpan diperhatikan sebagaimana bisa memotong keluaran strea, Untuk mengurangi *hybrid schemes* bisa dipertimbangkan. Dalam kasus ini diasumsikan bahwa beberapa asinkron antara pengirim dan penerima dapat diterima.

Misalkan pengirim bisa memproses sekumpulan blok *stream* dalam suatu waktu sebelum mengirimkan mereka, dengan proses yang tersalurkan hal ini cukup hanya menambahkan sebuah delay awal sebelum *stream* disampaikan. Pengirim akan menmberi nilai dengan sebuah nilai sekali pakai hanya satu blok saja dari keseluruhan. Dua puluh blok antara dua blok yang ditandatangani bisa diautentikasi menggunakan skema *offline*. Cara ini *one time signature* dan waktu verfikasi bisa dilunasi dalam keseluruhan blok.

Fitur yang berguna dari skema *one time signature* yang diberikan adalah memungkinkan veriikasi sekumpulan pesan dengan bit per bit. Ini memungkinkan kita untuk secara langsung mengembangkan bit tanda tangan dan waktu verifikasi. Jika kita mengasumsikan penerima dibolehkan untuk memainkan keseluruhan blok yang memiliki informasi terautentukasi sebelum menghentikan jika kerusakan terdeksi kita bisa melakukan hal-hal berikut. Kita bisa mendistribusikan bit-bit tanda tangan ke sepanjang dua puluh blok dan memverfikasi *hash* dari blok pertama bit demi bit sebagaimana bit tanda tangan tidaerima. Hal ini menjaga rate *stream* stabil karena kita tidak punya tanda tangan yang dikirimkan dalam sebuah blok tunggal dan verifikasi akan menghabiskan tiga sampai empat komputasi per blok. dalam setiap blok

Pembuangan *constraint* dalam memainkan keseluruhan blok yang tidak terautentikasi juga dimungkinkan sebelum kerusakan terdeteksi. Hal ini membutuhkan modifikasi sederhana untuk skema *online*. Daripada memasukkan blok Bi *one-time signature* milikinya, tandatangan dimasukkan dalam blok berikutnya. Hal ini berarti dalam skema online blok harus bisa diproses dua kali dalam satu waktu. Saat modifikasi diaplikasikan pada skema *hybrid*, tanda tangan bit di dalam 20 blok digunakan untuk mengautentikasi dua blok berikutnya sehinggan informasi autentikasi tidak pernah dimainkan. Walaubagaimanapun, hal ini berarti bahwa sekarang pengirim harus memproses dua kali lebih besar dalam skema *hybrid*.

7.4. *One Time Signature* Probabilistik

Panjang dan waktu komputasi dari tanda tangan sekali pakai dihitung dari panjang pesan yang dikirim. Pesan pertama kali di-hash, lalu panjang

dari fungsi *hash collision-resistant* adalah parameter krusial di sini. Walau bagaimanapun dalam memastikan fungsi ini adalah fungsi *collision-resistant* yang kuat, dubutuhkan asumsi dengan *range* yang panjang. Contohnya adalah MD5 dengan 128 bit atau SHA-1 dengan 160 bit.

8. Contoh Aplikasi

Arsip *stream* sangat jamak digunakan saat ini. Berikut penjabaran contoh-contoh penggunakan tanda tangan digital pada arsip-arsip *stream* yang ada. Contoh-contoh yang dibahas pada bagian ini adalah untuk arsip audio dan video, *applet*, *broadcast*, penghematan biaya koneksi, dan peluang penggunaannya pada *proxy server*.

8.1. Audio dan Video

Untuk contoh mengenai arsip video dan audio [GEN97] mencontohkannya dalam arsip MPEG. Sebab dalam kasus tersebut, arsip MPEG sudah mewakili audio dan video.

Untuk arsip jenis MPEG ini ada beberapa metode untuk memasukkan data autentikasi, yaitu:

1. Memasukkan data ke dalam *section* USER-DATA
2. Memungkinkan MPEG *system layer* membolehkan data-data elementer pada data *stream* dipecah secara sinkron dengan *stream* audio dan video yang terpaket
3. Menggunakan teknik yang diambil dari *digital watermarking* untuk menyimpan informasi di dalam video dengan resiko kehilangan kualitas.

Dalam kasus ini dikarenakan video MPEG berukuran sangat besar dan penerima harus minimal memiliki *buffer* sebesar 1.8Mbit, kedua solusi *(offline* dan *online)* bisa diimplementasikan tanpa terlalu mempengaruhi kualitas gambar. Hal serupa juga terjadi pada audio. Misalkan pada kasus ekstrim kualitas suara audio sangat kecil, yakni 32Kbits/s dan setiap *audio frame*nya juga kecil (milsakan 1000 bytes) dan *buffer* untuk menerima audio juga kecil (misalkan 2 Kbytes), memang membutuhkan penurunan kualitas suara. Sebab metode *online* membutuhkan sekitar 1000 byte untuk informasi autentikasi per blok. Untuk kasus ekstrim ini, strategi *online* lebih baik untuk mengirimkan informasi autentikasi dengan memisahkan tapi *multiplexed stream* data MPEG. Namun jika *buffer*-nya cukup, blok audio besar, skema online dapat membuat delay awal sekitar 5-6 detik dan kualitas degradasi kira-kira 5%. Jika *buffer* penerima kecil tapi tidak sangat kecil, skema *hybrid* akan bisa bekerja. Misalkan ada sebuah skem yang bisa dibuat sekali tapi menggunakan blok berurutan dengan panjang masing-masing 1000 byte. Ini akan menyebabkan degradasi sekitar 5% dan delay awal sekitar 20 detik.

8.2. Applet

Dalam contoh aplikasi diambil *applet* Java. *Applet* ini dijalankan melalui jaringan. Cara kerjanya adalah sebagai berikut. Pada saat *applet* dipanggil, sebuah kelas *startup* dipanggil. Nantinya setelah dibutuhkan kelas-kelas lain, maka kelas tersebut akan diunduh dan dijalankan.

Bila dicocokkan dengan pembahasan mengenai tanda tangan digital pada arsip *stream* ini, diterapkan dengan menandatangani kelas *startup* dan setiap kelas yang diunduh harus memiliki nilai *hash* dari kelas-kelas tambahan yang langsung diunduh.

Metode ini sendiri sudah diterapkan sejak JDK1.1, dengan setiap *applet* dikomposisi menjadi satu atau sekumpulan arsip Java, masing-masingnya mengandung tabel *hash* dari komponen. Namun jika *applet* Java tersebut berukuran sangat besar dan kompleks, hal ini akan menyebabkan masalah. Sebab:

1. Ukuran *hash table* yang besar ke dalam kelas akan mengganggu. Tabel ini harus diekstraksi dan diautentikasi sebelum kelas diautentikasi.
2. Ongkos komputasional untuk pembubuhan setiap manifest jika sebuah *applet* dikomposisi dari beberapa arsip.
3. Mengakomodasi kelas atau data yang dibuat saat jalan oleh aplikasi berbasi server sesuai permintaan *client*.

Permasalahan ini dapat diselesaikan dengan teknik-teknik yang ada di dalam makalah ini.

8.3. Aplikasi *Broadcast*

Skema online dan offline dapat dengan mudah dimodifikasi untuk menyesuaikan dengan skenario *broadcast*. Misalkan diasumsikan bahwa *stream* dikirimkan ke sebuah *broadcast channel* dengan beberapa penerima yang secara dinamik bergabung atau meninggalkan *channel*. Dalam kasus ini penerima yang bergabung

dengan transmisi telah memulai tidak bisa untuk mengautentikasi *stream* sejak kehilangan blok pertama yang mengandung isi *stream*. Kedua skema walaubagaimanapun bisa dimodifikasi sehingga setiap sebuah bagian merupakan bagian dari informasi berurutan. Selain itu juga ada tanda tangan digital regular dalam sebuah blok yang tersimpan di dalam *stream*. Penerima yang telah memverifikasi *stream* melalui mekanisme *chaining* bisa melupakan tanda tangan ini sebagaimana penerima mengaturnya pada bermacam-macam waktu.

8.4. Arsip Panjang saat Berkomunikasi dengan Biaya

Solusi ini bisa digunakan untuk mengaitentikasi arsip panjang dalam usahanya untuk mengurangi biaya komunikasi jika terdapat kerusakan. Misalkan peneria melakukan pengunduhan terhadap sebuah arsip panjang yang ada di web. Tidak ada kebutuhan yang harus dipenuhi untuk mengunduh arsip tersebut, sehingga penerima dapat dengan mudah menerima seluruh arsip dan memeriksa tanda tangan pada akhirnya. Tapi jika arsip tersebut mengalami kerusakan, penerima baru akan mengetahuinya setelah mengambil akhir arsip. Sementara komunikasi membutuhkan biaya, yaitu misalnya waktu, dan *bandwith*, solusi ini sama sekali tidak efektif. Dengan solusi *offline*, penerima dapat menginterupsi transmisi jika kerusakan terdeteksi sejak awal.

8.4. *Content-Filtering* Berbasis Tanda Tangan pada Proxy

Tanda tangan digital juga bisa digunakan sebagai penyaring *content* yang diterima oleh *proxy server* melalui *firewall*. Biasanya, jika terdapat sebuah *firewall* dan sebuah komputer ingin terhubung ke sana, koneksi baru bisa dilakukan dengan *proxy server*. Komputer tersebut akan membuat koneksi ke *proxy server* dan jika diizinkan masuk, *proxy server* akan meneruskan koneksinya ke *server* yang dituju. *Proxy* selanjutnya akan mensimulasikan koneksi antara mesin internal dan eksternal dengan mnduplikasi data antara kedua koneksi. Proxy dapat dimodifikasi sehingga hanya memperbolehkan penandatanganan data untuk mengalir dari server eksternal ke mesin internal. Tapi, dikarenakan *proxy* hanya mennduplikasi data dari koneksi eksternal tersebut ke koneksi internal dan tidak bisa menyimpan semua data yang masuk sebelum memindahkannya, *proxy* tersebut tidak bisa menggunakan skema tanda tangan umum untuk menyelesaikan masalah ini. Tapi, dapat dilihat bahwa dari sudut pandang *proxy*, data adalah sebuh *stream*. Karena itu jika ada sebuah cara terstandardisasi untuk memasukkan data yang terautentikasi dalam setiap stream dan teknik untuk ini, maka metode-metode di atas dapat terpakai.

9. Kesimpulan

Dari hasil studi mengenai tanda tangan digital dan penerapannya dalam arsip *stream*, maka dapat diambil beberapa kesimpulan yang dipaparkan sebagai berikut.

1. Tanda tangan digital adalah sebuah metode untuk memberikan tanda tangan pada dokument digital dengan mengikuti sifat-sifat yang dimiliki oleh tanda tangan pada dokumen biasa.
2. Tanda tangan digital memfasilitasi adanya verifikasi dan anti penyangkalan. Hal ini memenuhi dua dari empat kebutuhan utama dari kritografi, yaitu Kerahasiaan pesan (*confidentiality*), Otentikasi (*authentication*), Keaslian pesan (*data integrity*), dan Anti penyangkalan (*non repudiation*).
3. Metode pemberian tanda tangan itu sendiri terdiri atas dua jenis, yaitu Menggunakan enkripsi pesan dan menggunakan fungsi *hash* dan kriptografi kunci publik.
4. Dalam metode pemberian tanda tangan yang menggunakan fungsi *hash*, terdapat penggunaan kunci publik (*public key*) dan kunci privat *(private key).* Kunci publik adalah kunci milik seseorang yang diberitahukan kepada publik. Dapat digunakan untuk autentikasi ataupun enkripsi pesan. Tergantung pada skema kriptografi yang digunakan. Sementara kunci privat adalah kunci milik seseorang yang tidak diketahui oleh publik. Penggunaannya juga tergantung pada skema kriptografi yang digunakan.
5. MAC juga merupakan sebuah fungsi satu arah, tapi fungsi ini menggunakan kunci rahasia (kunci privat/*secret key*) dalam pembangkitan nilai *hash*.
6. Arsip *stream* adalah arsip yang terdiri dari sekumpulan bit yang sangat

panjang (mungkin tanpa batas) yang dikirimkan pengirim ke penerima. Kelebihan dari arsip ini adalah dapat dikonsumsi tanpa harus menunggu arsip peuh dikirimkan, dan penerima tidak perlu menyimpan semua data yang dikirimkan untuk mengkonsumsi.

7. Untuk implementasi pemberian tanda tangan digital pada arsip *stream*, terdapat beberapa prasyarat, yatiu, fungsi *hash collision resistant* seperti MD5 dan SHA-1, skema tanda tangan digital, dan metode pendekatan pemberian tanda tangan pada arsip *stream*. Terdapat dua pendekatan berbeda terhadap dua jenis arsip *stream* yang ada. Kedua jenis arsip *stream* ini adalah arsip *stream* terbatas, yang diketahui kapan berakhirnya dan arsip *stream* tidak terbatas, yang tidak diketahui kapan berakhirnya.

8. Pendekatan untuk arsip yang terbatas disebut solusi *off-line*. Pada skema *offline* ini, arsip *stream* dibagi menjadi beberapa blok kecil, dan setiap blok memiliki nilai *hash*. Penerima menerima nilai *hash* sebuah terlebih dahulu baru blok yang akan dilihat. Nantinya akan dilakukan pencocokan antara nilai hash yang diterima dengan komputasi oleh pengguna sendiri atas nilai *hash* dari blok yang diterimanya.

9. Pendekatan untuk *stream* yang terbatas disebut solusi *online*. Pada solusi ini digunakan konsep *one time signature*. Arsip *stream* dibagi menjadi beberapa blok. Awalnya pengirim mengirim kunci publik *one time*, lalu selanjutnya mengirim blok pertama bersama kunci publik *one time* berikutnya, sehingga skema ini bisa berlanjut secara terus menerus.

10. Isu keamanan mengenai implementasi ini terjamin seperti yang dijelaskan pada [GEN97] dengan adanya dua teorema yaitu

    - Jika *(G,S,V)* adalah skema tanda tangan yang aman dan H adalah fungsi *hash collision-resistant* maka skema tanda tangan *(G'S'V')* juga aman. Teorema ini berlaku untuk solusi *off-line*
    - Jika *(G,S,V)* dan *(g,s,v)* merupakan skema tanda tangan yang aman dan H adalah fungsi *hash collision-resistant* maka hasil skema tanda tangan (G',S',V') juga aman. Teorema ini berlaku untuk solusi *online.*

11. Terkait masalah implementasi [GEN97] juga dijelaskan mengenai alasan pemilihan *one time signature scheme*, anti penyangkalan, *hybrid schemes*, dan *probabilistic one-time signatures*.

12. Pemilihan *one time signature* didasarkan kekuatannya dalam mengkompromikan panjang tanda tangan waktu yang dibutuhkan untuk verifikasi.

13. Anti penyangkalan dapat dilakukan pada kedua metode, baik *offline* maupun *online*. Penerima hanya butuh menyimpan blok yang pertama. Hal ini membuktikan penerima menerima sesuatu dari pengirim. Sementara untuk kasus *stream* yang tidak terbatas, setidaknya penerima harus menyimpan blok pertama dan semua *one-time signature*, sehingga mengharuskan pengirim mengkonstruksi arsip kirimannya berdasarkan blok-blok dan *one time signature* tersebut.

14. Contoh penggunaan tanda tangan digital pada arsip *stream* dapat dilihat dari pengunduhan arsip audio dan video, *applet* Java, dan aplikasi *broadcast*, komunikasi yang digunakan untuk menghemat biaya, dan peluang untuk implementasi pada *proxy server*.

15. Solusi pemberian tanda tangan pada arsip *stream* membutuhkan metode pendekatan yang berbeda, yaitu *skema online* dan *offline* yang keduanya menerapkan penyimpanan nilai *hash* atau kunci sekali pakai di setiap bloknya. Dengan kedua ini, penerima tidak perlu mengambil seluruh arsip untuk melakukan validasi data, dan mempermudah pengguna untuk dapat memutuskan koneksi jika diketahui sebagian dari data yang diperolehnya ternyata tidak valid

7. Daftar Pustaka

1. [GEN97] Gennaro, Rosario dan Pankaj Rohatgi, *How To Sign Digital Stream*, I.B.M. T.J. Watson Research Center, New York, 1997
2. [GOR03] Gorreri, Roberto dkk, *Composional Integrity for Digital Stream Signature Protocols*, Dipartimento di Scienze dell'Informatzione, Universita di Bologna, Italia, 2003.
3. [MUN06] Munir, Rinaldi, M.T. *Diktat Kuliah IF5054 Kriptografi*. Program Studi Teknik Informatika, Sekolah Tinggi Elektro dan Informatika, Institut Teknologi Bandung, 2006.
4. [SCH96] Schneier, Bruce, *Applied Cryptography 2nd*, John Wiley and Sons, 1996.
5. [RSA06] *RSA Secirity , What is One Time Digital Signature Scheme?* http://www.rsasecurity.com/rsalabs/node.asp?id=2343, diakses terakhir 1 Januari 2007, pukul 23:00.