Analisis Keamanan Penggunaan Kunci Publik dan Privat pada Digital Signature untuk Aplikasi Blackberry

Matthew – NIM: 13507012
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
matt_jcs@yahoo.com

***Abstrak*— Dalam pengiriman suatu data yang bersifat penting dan rahasia, diperlukan sebuah penanganan agar mengetahui bahwa data tersebut adalah data yang asli (sebenarnya) dan belum dilakukan perubahan-perubahan pada tersebut, sesedikit apapun perubahannya. Oleh karena itu, dalam pengiriman pesan penting seperti itu sangat diperlukan tandatangan digital (digital signature). Pesan penting tersebut pertama akan di signing dengan menggunakan kunci privat yang hanya dimiliki oleh sang pengirim. Setelah pesan diterima atau data tersebut diunduh oleh sang penerima, maka penerima tersebut cukup menggunakan kunci publik untuk memverifikasi keabsahan data / pesan tersebut, apakah data tersebut asli dan apakah sang pengirim merupakan pengirim yang sebenarnya.**

Saat ini aplikasi BlackBerry tidak hanya digunakan pada kalangan masyarakat umum saja. Namun berbagai pihak profesional pun sudah menggunakan BlackBerry untuk mengoptimalkan pekerjaannya. RIM (Research In Motion) sendiri saat ini sudah memfasilitasi BES atau BlackBerry Enterprise Server yang diperuntukkan ke perusahaan yang ingin memiliki jaringan BlackBerry untuk internal perusahaannya. Dalam penggunaan seperti ini, sangatlah diperlukan keamanan yang sangat tinggi dalam pentransmisian setiap data yang ada, karena rahasia perusahaan sangatlah penting untuk dijaga. Oleh karena itu RIM sudah menyediakan API untuk menggunakan kunci publik dan kunci privat pada digital signature.

Pada makalah ini penulis akan menganalisis penggunaan algoritma RSA dengan kunci publik dan privat untuk membuat sebuah digital signature pada sebuah pesan. Setelah itu, juga akan ditunjukan pengimplementasian API untuk algoritma ini dalam platform Java untuk sebuah aplikasi BlackBerry, bagaimana cara penggunaannya. Kemudian pada makalah ini juga akan dianalisa keamanan pesan yang dikirim setelah diberi digital signature dengan API dari RIM ini, yaitu dengan membandingkan hasil verifikasi data yang asli dan data yang sudah dirubah keasliannya.

***Kata kunci*: BlackBerry, kunci publik, kunci privat, RSA, implementasi API.**

I. PENDAHULUAN

Sejak zaman dahulu, tanda-tangan sudah digunakan untuk otentikasi dokumen cetak. Tanda-tangan sendiri mempunyai karakteristik sebagai berikut. Yang pertama adalah tanda-tangan merupakan bukti yang otentik. Kedua, tanda tangan tidak dapat dilupakan. Ketiga tanda-tangan tidak dapat dipindah untuk digunakan ulang. Keempat dokumen yang telah ditandatangani tidak dapat diubah. Dan kelima tanda-tangan dianggap tidak dapat disangkal (*repudiation*).

Dengan seiringnya perkembangan jaman kea rah dunia digital, maka fungsi tanda tangan pada dokumen kertas juga dapat diterapkan untuk otentikasi pada data digital (pesan, dokumen elektronik). Tanda-tangan untuk data digital ini dinamakan tanda-tangan digital (*digital signature*). Tanda-tangan digital bukan merupakan tulisan tanda-tangan yang di-digitisasi (di-scan), namun tanda-tangan digital ini adalah nilai kriptografis yang bergantung pada isi pesan dan kunci. Tanda-tangan pada dokumen cetak selalu sama, apa pun isi dokumennya, sedangkan tanda-tangan digital selalu berbeda-beda antara satu isi dokumen dengan dokumen lain.

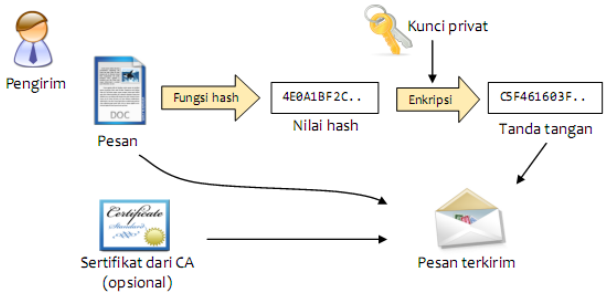
BlackBerry sendiri yang merupakan media pengiriman pesan, membutuhkan pengamanan data yang baik. Hal ini disebabkan BlackBerry tidak hanya

digunakan pada lingkungan pengguna biasa (*standard user*), namun BlackBerry kerap kali digunakan juga pada lingkungan perusahaan. Hal ini didukung RIM sendiri, perusahaan khusus pengembang piranti lunak pada BlackBerry, yaitu dengan ada nya BlackBerry Enterprise Server untuk digunakan dalam sebuah jaringan internal perusahaan.

Dalam hal ini, maka RIM telah menyediakan sebuah API, yakni Cryptography API untuk mendukung pemrograman pada perangkat BlackBerry yang membutuhkan tingkat keamanan lebihyaitu bernama net.rim.device.api.crypto. Dengan API ini, maka fungsi-fungsi yang berhubungan dengan kriptografi seperti enkripsi dan dekripsi dengan kunci public-privat dapat langsung digunakan. Begitu pula dengan algoritma RSA untuk penandatanganan digital dapat langsung digunakan, yaitu terdapat pada bagian Key Management API letaknya pada Cryptography API ini yang akan dijelaskan selanjutnya.

II. DIGITAL SIGNATURE STANDARD (DSS)

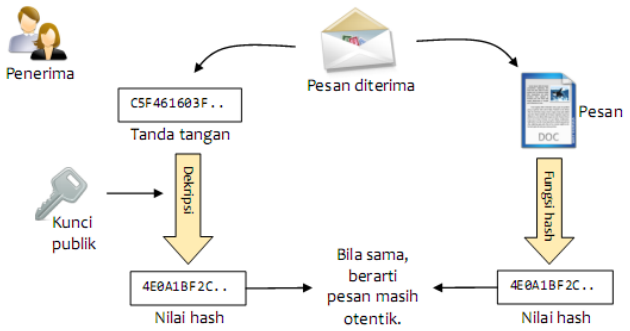
Digital signature standard atau tanda tangan digital adalah cara untuk menunjukkan bahwa pesan (*message*) yang dikirimkan adalah otentik, yaitu asli dari pengirim dan belum ada perubahan yang dilakukan terhadap pesan tersebut. Dengan cara ini maka penerima pesan dapat mempercayai bahwa pesan yang ia terima adalah pesan yang asli, tanpa pernah diubah isinya oleh pihak lain saat dalam perjalanan / pengiriman. Skema tanda tangan digital sendiri dapat diilustrasikan sebagai berikut:



Gambar 1. Skema pemberian tanda tangan digital pada sebuah dokumen / pesan

Pertama-tama, pengirim menyiapkan pesan yang hendak ia kirim. Setelah itu barulah dengan bantuan fungsi hash, pengirim membuat sepasang kunci privat dan kunci publik. Setelah itu, ia mengenkripsi pesan yang yang ia kirim dengan kunci privat yang ia miliki dan menjadi sebuah tanda tangan digital. Tanda tangan tersebut kemudian ia bubuhkan pada pesan digital yang hendak ia kirim.

Sedangkan pada sisi penerima, pertama-tama ia mendapat kunci publik yang dibagikan secara bebas oleh sang pengirim pesan. Setelah menerima pesan tersebut, makan sang penerima akan mengambil tanda tangan digitalnya untuk didekripsi dengan kunci publik yang ia miliki. Nilai hash ini kemudia disimpan, dan kemudian dicocokan dengan nilai hash yang diperoleh dari pesan yang dihitung nilai hashnya. Jika nilai hash yang diperoleh ternyata sama, maka dapat dipastikan keaslian dokumen beserta pengirimnya ada benar.



Gambar 2. Skema pemverifikasian tanda tangan digital pada sebuah dokumen / pesan

A. Algoritma RSA

RSA adalah algoritma enkripsi dekripsi yang digunakan dalam penandatanganan digital. RSA adalah algoritma kunci publik atau kriptografi asimetri, dimana kunci untuk mengenkripsi pesan berbeda dengan kunci untuk mendekripsi pesan. Kunci enkripsi merupakan kunci yang bersifat pribadi, yaitu hanya dimiliki sang pengirim pesan dan harus dijaga dari jangkauan orang lain, maka kunci tersebut disebut **kunci privat**. Sedangkan kunci untuk mendeskripsi pesan yang telah ditandatangani sang pengirim tidak dirahasiakan, melainkan dipublikasikan secara umum oleh sang penerima, maka disebut kunci publik. Algoritma pembangkitan pasangan kunci tersebut adalah sebagai berikut:

1. Pilih dua bilangan prima yang besar, sebut saja *p* dan *q*, dan jangan sama angkanya. (rahasia)
2. Hitung $n = pq$. (tidak rahasia)
3. Hitung $\phi(n) = (p - 1)(q - 1)$. (rahasia)
4. Pilih satu bilangan *e* yang *coprime* terhadap $\phi(n)$. *Coprime* atau relatif prima artinya

faktor pem-bagi terbesarnya adalah 1. Bilangan *e* dan *n* ini dipublikasikan sebagai *kunci publik*, tidak rahasia.

5. Hitung kunci privat *d* yang memenuhi persamaan $ed \equiv 1 \, (mod \, \phi(n))$. Cara lainnya adalah:

$$d = \frac{1 + k \, \phi(n)}{e}$$

Dengan mencoba-coba nilai *k* mulai dari 1, 2, 3, …, akan terdapat *d* yang bulat (bukan pecahan). Nilai *d* ini adalah *kunci privat* dan tentu saja bersifat rahasia.

Cara enkripsi adalah sebagai berikut. Bagi pesan dalam blok-blok *m*1, m2, dan deterusnya. Setiap blok menyatakan nilai angka di antara 0 dan $n - 1$. Kemudian blok cipherteks didapat dari persamaan berikut:

$$c_i = {m_i}^e \bmod n$$

Dekripsi menggunakan rumus yang sama, hanya kita menggunakan kunci privat *d*.

$$m_i = {c_i}^d \bmod n$$

Namun untuk tanda tangan digital, kita mempertukarkan *e* dan *d* pada kasus enkripsi dan dekripsi. *e* dan *d* memang berkoresponden, namun mereka hanyalah sepasang angka. Maka tidak masalah bila kita mempertukarkan *e* untuk dekripsi dan *d* untuk enkripsi.

B. Algoritma SHA-1

SHA-1 atau *secure hash algorithm* adalah salah satu algoritma fungsi *hash* satu arah dari keluarga SHA. Fungsi *hash* adalah fungsi yang menghasilkan nilai yang panjangnya tetap, tak peduli berapa panjang pesan yang menjadi inputnya. Fungsi *hash* bekerja satu arah, sehingga sekali nilai *hash* didapat maka (seharusnya) tidak mungkin mencari plainteks asalnya dari nilai *hash* ini. Serangan terhadap fungsi *hash* disebut *collision attack*, yaitu usaha mencari plainteks lain yang memiliki nilai *hash* yang sama.

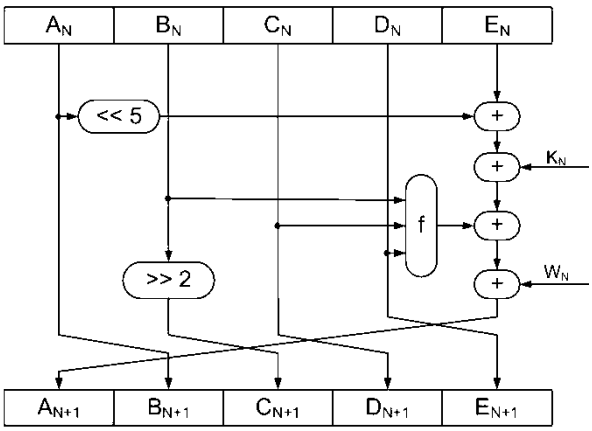
SHA-1 menghasilkan nilai *hash* — disebut *message digest* — yang panjangnya 160 bit. Ukuran plain-teks maksimum yang bisa ia terima adalah 2^{64} bit.

Cara kerja SHA-1 adalah sebagai berikut. Plainteks dihitung panjangnya, lalu ditambah bit-bit pengganjal (*padding bits*) supaya panjangnya $\equiv 448 \pmod{512}$. Bila panjang pesan tepat sama dengan 448 bit, maka tetap ditambahi bit pengganjal sebanyak 512 bit sehingga panjang pesan 960 bit. For-mat bit pengganjal adalah satu buah bit 1 diikuti banyak bit 0 seperlunya. Contoh 1000000000000… . Setelah itu, panjang keseluruhan pesan dihitung. Hasil perhitungan ini sejumlah 64-bit ditambahkan ke ekor rangkaian bit pengganjal tadi. Sampai di sini, panjang pesan sudah menjadi kelipatan 512 bit.

Pesan sepanjang tadi dibagi-bagi menjadi blok-blok 512 bit. Setiap blok dilewatkan ke fungsi H_{SHA} un-tuk dihitung *hash*nya. Fungsi H_{SHA} menerima masukan blok 512 bit itu dan nilai *hash* dari H_{SHA} blok sebelumnya. Nilai *hash* yang final, yaitu message digest dari plainteks, adalah hasil keluaran dari H_{SHA} blok terakhir. Lalu bagaimana H_{SHA} blok pertama? Dia tidak memiliki nilai *hash* blok sebelumnya. Maka dari itu SHA-1 menggunakan 5 buah buffer, diberi nama A B C D E, panjangnya masing-masing 32-bit. Isi dari buffer sebelum SHA-1 dihitung sudah ditetapkan sebagai berikut:

A = 67452301_{hex}
B = $EFCDAB89_{hex}$
C = $98BADCFE_{hex}$
D = 10325476_{hex}
E = $C3D2E1F0_{hex}$

Setiap blok H_{SHA} dapat digambarkan sebagai berikut:



Gambar 3. Skema fungsi hash SHA-1

Operasi "<<" dan ">>" pada gambar adalah ope-rasi rotasi bit. Operasi '>> 2' ekivalen dengan '<< 30'. Juga operasi "+" pada gambar adalah ope-rasi penjumlahan lalu di-mod 2^{32}.

Satu gambar ini diulang sebanyak 80 putaran. Pada gambar di samping, terdapat simbol W, K, dan fungsi

f. Ketiga entitas ini bergantung pada putaran keberapa saat ini.

Nilai W dihitung dari blok pesan 512-bit yang sedang diproses oleh H_{SHA} saat ini. Untuk putaran per-tama, W_1 adalah 32 bit pertama dari blok 512 bit. Putaran kedua, W_2 adalah 32 bit berikutnya. Begitu seterusnya sampai W_{16}. Mulai putaran ke-17, nilai W_i dihitung dari rumus berikut:

$$W_i = (W_{i-16} \oplus W_{i-14} \oplus W_{i-8} \oplus W_{i-3}) \text{ lalu rotasi 1 bit ke kiri.}$$

Nilai K adalah konstanta penambah. Nilainya sudah ditetapkan sebagai berikut:

Putaran ke-00 ≤ x ≤ 19; K = $5A827999_{hex}$
Putaran ke-20 ≤ x ≤ 39; K = $6ED9EBA1_{hex}$
Putaran ke-40 ≤ x ≤ 59; K = $8F1BBCDC_{hex}$
Putaran ke-60 ≤ x ≤ 79; K = $CA62C1D6_{hex}$

Terakhir ada fungsi *f*, apa yang ia kerjakan juga bergantung pada putaran. Isi fungsi *f* adalah:

Putaran	*f* (B, C, D) =
0 ≤ x ≤ 19	(B ∧ C) ∨ (~B ∧ D)
20 ≤ x ≤ 39	B ⊕ C ⊕ D
40 ≤ x ≤ 59	(B ∧ C) ∨ (B ∧ D) ∨ (C ∧ D)
60 ≤ x ≤ 79	B ⊕ C ⊕ D

Tabel 1. Nilai dari fungsi f tergantung pada urutan putaran saat ini

Setelah semua 80 putaran ini selesai, isi dari buffer A, B, C, D, dan E menjadi input bagi H_{SHA} blok berikutnya. Nilai *hash* final adalah gabungan rangkaian bit di dalam A-B-C-D-E hasil H_{SHA} blok terakhir.

III. Implementasi DSS pada Aplikasi BlackBerry Berbasis Java

Untuk tahap implementasi, penulis membuat sebuah aplikasi berbasis Java untuk *BlackBerry* dengan *IDE* ecplise *GANYMEDE*. Dalam membangun aplikasi ini, source code dibagi ke dua file utama, yaitu Controller dan Screen.

Controller adalah file yang mengatur fungsi fungsi untuk melakukan enkripsi, dekripsi, dan bertugas untuk menampilkan screen ke layar perangkat *BlackBerry*. Sedangkan Screen adalah file yang bertugas mengatur komponen tampilan pada layar, dan mengatur event apa yang akan dilakukan jika suatu tombol ditekan, suatu text dimasukan, dan hal-hal yang berubungan dengan tampilan lainnya. Di bawah ini merupakan potongan source code untuk aplikasi **BB Digital Signature**.

File: MattController.java

```
/**
 *
 */
package com.peekatree.controller;

import net.rim.device.api.crypto.*;
import net.rim.device.api.ui.UiApplication;

import com.peekatree.view.MattScreen;

/**
 * MainMenu Controller
 * @author Matthew
 */
public class MattController {

    // Screen -------------------------
        public MattScreen screen;

        /**
         * Constructor
         * @throws
UnsupportedCryptoSystemException
         * @throws
CryptoUnsupportedOperationException
         * @throws CryptoTokenException
         */
        public MattController() throws
CryptoTokenException,
CryptoUnsupportedOperationException,
UnsupportedCryptoSystemException {
                screen = new MattScreen(this);
        }

        /**
         * Run the LoginController
         */
        public void run() {

        UiApplication.getUiApplication().pushScre
en(screen);
        }

    /**
     * Dengan Menggunakan data dan kunci privat,
dapat dibuat signature untuk
     * menyediakan data yang dapat terpercaya
dan diautentikasi
     * @param privateKey : yaitu kunci privat
untuk menandatangani data
     * @param data : data yang akan
ditandatangani
     * @return : mengembalikan tanda tangan
dalam bentuk byte
     */
    public byte[] sign( RSAPrivateKey
privateKey, byte[] data ) throws CryptoException
    {
        // Membuat PKCS1 signature signer
        PKCS1SignatureSigner signer = new
PKCS1SignatureSigner( privateKey );
        signer.update( data );

        byte[] signature = new byte[
signer.getLength() ];
        signer.sign( signature, 0 );
```

```
        return signature;
    }

    /**
     * Menggunakan data dan kunci publik untuk
memverifikasi keabsahan data dan keaslian tanda
tangan
     * @param publicKey : kunci publik untuk
verifikasi*
     * @param data : data yang diterima
     * @param signature : tanda tangan digital
di dalam data
     * @return : boolean, menyatakan apakah
tanda tangan valid atau tidak
     */
    public boolean verify( RSAPublicKey
publicKey, byte[] data, byte[] signature )
throws CryptoException
    {
        PKCS1SignatureVerifier verifier = new
PKCS1SignatureVerifier( publicKey, signature, 0
);
        verifier.update( data );
        return verifier.verify();
    }
}
```

File: MattScreen.java

```
package com.peekatree.view;

import com.peekatree.controller.MattController;

import net.rim.device.api.crypto.*;
import net.rim.device.api.ui.Field;
import
net.rim.device.api.ui.FieldChangeListener;
import
net.rim.device.api.ui.component.BasicEditField;
import
net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.component.Dialog;
import
net.rim.device.api.ui.component.LabelField;
import
net.rim.device.api.ui.container.HorizontalFieldM
anager;
import
net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.decor.Background;
import
net.rim.device.api.ui.decor.BackgroundFactory;

/**
 * MattScreen
 * @author Matthew
 *
 */
public class MattScreen extends MainScreen {

        // This screen's controller ------------
-------------
        MattController controller;

        // Components declaration ---------------
-------------
        Background editBg;
        HorizontalFieldManager hfm1;
                LabelField lblData;
                BasicEditField textData;
                ButtonField btnSign;
        HorizontalFieldManager hfm2;
                LabelField lblSign;
                BasicEditField textSign;
                ButtonField btnVerify;
        RSAKeyPair senderKeyPair;



        // MenuItem declaration -----------------
-------------

        /**
         * Constructor
         * @throws
UnsupportedCryptoSystemException
         * @throws
CryptoUnsupportedOperationException
         * @throws CryptoTokenException
         */
        public MattScreen(MattController mattctr)
throws CryptoTokenException,
CryptoUnsupportedOperationException,
UnsupportedCryptoSystemException {
                this.controller = mattctr;

                // Membuat pasangan kunci RSA
untuk digunakan pada DSS berikut
                senderKeyPair = new RSAKeyPair( new
RSACryptoSystem( 1024 ));


                // Components initialization ----
----------------------
                hfm1 = new
HorizontalFieldManager(HorizontalFieldManager.FI
ELD_LEFT);
                        lblData =  new
LabelField("Original Message: ");
                        textData= new
BasicEditField();
                        btnSign = new
ButtonField("Sign!");

        btnSign.setChangeListener(new
FieldChangeListener() {
                                public void
fieldChanged(Field arg0, int arg1) {
                                        byte data[]
= textData.getText().getBytes();
                                byte[] signature;
                                        try {

        signature =
controller.sign(senderKeyPair.getRSAPrivateKey()
, data );

textSign.setText(new String(signature));
                                        } catch
(CryptoException e) {
                                                //
TODO Auto-generated catch block

        e.printStackTrace();
                                        }
                                }
                        });
                        hfm1.add(lblData);
                        hfm1.add(textData);
```

```
                hfm2 = new
HorizontalFieldManager(HorizontalFieldManager.FI
ELD_LEFT);
                        lblSign =  new
LabelField("Signature: ");
                        textSign = new
BasicEditField();
                        btnVerify = new
ButtonField("Verify!");

        btnVerify.setChangeListener(new
FieldChangeListener() {
                                public void
fieldChanged(Field arg0, int arg1) {
                                        boolean
verified;
                                        try {

        verified =
controller.verify(senderKeyPair.getRSAPublicKey(
), textData.getText().getBytes(),
textSign.getText().getBytes());
                                                if(
verified ) {

        Dialog.alert("Data and signature are
still authentic ^_^");
                                                }
else {

        Dialog.alert("Data or signature has been
modified! be careful >.<");
                                                }
                                        } catch
(CryptoException e) {
                                                //
TODO Auto-generated catch block

        e.printStackTrace();
                                        }
                                }
                        });
                        hfm2.add(lblSign);
                        hfm2.add(textSign);
                // Adding of components to this
screen ---------------
                add(hfm1);
                add(btnSign);
                add(hfm2);
                add(btnVerify);
                // MenuItem initialization ------
----------------------

                // Adding of MenuItem to this
screen -----------------

                // Setting screen ---------------
----------------------

        getMainManager().setBackground(Background
Factory.createSolidBackground(0x00EEFFDE));
        }

}
```

Setelah selesai dengan membuat source code, maka selanjutnya adalah melakukan kompilasi, kemudian menjalankan simulator BlackBerry melalui IDE Eclipse GANYMEDE.

III. Analisis dan Ujicoba Keamanan pada Aplikasi BB Digital Signature

Setelah simulator dijalankan, maka aplikasi ini secara otomatis akan masuk kedalam handset simulator BlackBerry.

Gambar 4. Aplikasi BB Digital Signature folder download di sebuah perangkat BlackBerry

Gambar 5 Detil dari aplikasi BB Digital Signature

Untuk menjalankan aplikasi, cukup menekan tombol enter atau trackball pada aplikasi BB Digital Signature ini.

Berikut ini adalah langkah-langkah dari awal membuka aplikasi BB Digital Signature hingga selesai mendekripsi dan mendapat pesan awal yang dienkripsi.

Gambar 6 Tampilan awal BB Digital Signature

Aplikasi ini akan menerima input Plaintext (berupa string agar mudah dicoba), yang kemudian diberi tanda tangan digital. Untuk mendapat tanda tangan digital dari data yang dikehendaki cukup menekan tombol Sign!

Pada test case berikut, pesan yang hendak dikirim adalah "Ini merupakan pesan rahasia yang hendak dikirim dan diberi tanda tangan digital"

Gambar 7 Signature diperoleh dalam format byte

Gambar 8. Hasil dari penekanan tombol Verify! Ketika data dan tanda tangan belum dirubah.

Gambar 9 Hasil dari penekanan tombol Verify! ketika ada bagian di tanda tangan yang dirubah (angka 9 → 8)

Gambar 10 Hasil dari penekanan tombol Verify! ketika ada bagian di data yang dirubah (tanda ? di akhir pesan)

Pada aplikasi ini, setelah tanda tangan dibuat, maka jika terdapat sedikit saja perubahan, baik di data maupun pada tanda tangan itu sendiri, maka akan terdeteksi bahwa sudah terdapat perubahan pada pesan.

IV. KESIMPULAN

Tanda tangan digital merupakan salah satu fitur keamanan yang baik dalam pengiriman data atau pesan secara digital. Hal ini disebabkan digital signature berbeda dengan tanda tangan biasa (di atas kertas) maupun tanda tangan hasil *scan*, tanda tangan digital lebih sulit dipalsukan dan data pun tidak dapat diubah, hal ini disebabkan untuk benar-benar sempurna memalsukan tandatangan digital, seseorang harus mendapat kunci privat yang dimiliki sang pengirim. Adalah tanggung jawab si pengirim untuk menjaga baik-baik kunci privatnya. Karena selain dengan kehilangan kunci privat, keamanan DSS ini sudah sangat terjamin.

BlackBerry telah menyediakan API-API untuk meningkatkan keamanan dalam berkirim data. Salah satunya adalah fitur Digital Signature ini. Dalam implementasi di dunia nyata, API ini dapat digunakan untuk lingkungan perusahaan, untuk menjamin keaslian data yang dikirim oleh direktur misalnya, maka para karyawan cukup memiliki kunci publiknya dan keaslian data dapat diverifikasi dengan mudah dan aman. Selain itu, API untuk Digital Signature ini pun dapat digunakan untuk berkirim pesan yang sangat penting.

V. REFERENSI

[1]. *BlackBerry JDE API Reference*, < http://www.*BlackBerry*.com/*developer*s/docs/4.6.0api/ >, diakses pada 14 Mei 2010

[2]. Rinaldi Munir, "*Diktat Kuliah IF5054*", Departemen Teknik Informatika, Institut Teknologi Bandung,2005.

[3]. *BlackBerry Java Development Environment Labs*, <http://na.*BlackBerry*.com/eng/*developer*s/resources/*developer*_labs.jsp#tab_tab_jde>, diakses pada 14 Mei 2010.

[4]. *BlackBerry* Java Development Environment version 4.6.0. *Development Guide*. Canada: Research In Motion Limited. Modifikasi terakhir file: 5 Desember 2009

[5]. Info Security US, <http://infosecurity.us/>, diakses pada 15 Mei 2010.

PERNYATAAN

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

Bandung, 17 Meil 2010

Matthew Wangsadiredja
13507012