Studi Perbanding Penggunaan Algoritma Hash SHA 256 dengan Simetrik dan Asimetrik *Ciphers* dalam Perancangan *Secure* SWF *Rich Internet Application* (RIA)

Dody Dharma

Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Perumahan Sukamenak Indah Blok.I No.62 Jl.Jopo Sayati, Bandung, Jawa Barat
e-mail: dody.dharma@yahoo.com

Abstrak

Aplikasi berbasis Flash memiliki ekstensi SWF, sehingga aplikasi jenis ini dapat disebut juga sebagai aplikasi SWF. Untuk menghasilkan aplikasi SWF yang handal dan aman maka perlu terdapat suatu mekanisme pengamanan terhadap *Unauthorized access to data in transit* serta *unauthorized access to local data.*

Untuk mencapai hal ini, terdapat beberapa cara pengamanan melalui proses autentifikasi dengan dukungan kriptografi yang diantaranya adalah penggunaan algoritma Hash dan pengunaan *Cipher* Simetrik dan Asimetrik. Algoritma hash sangat berguna jika *user* ingin melakukan pengamanan pesan secara irreversible. Sedangakan metode *Cipher* simetrik dan asimetrik berguna unutk pengaman yang bersifat reversible.

Mengingat bahwa aplikasi SWF merupakan aplikasi yang dapat di-*decompiled.* Maka perlu strategi bijak untuk memanfaatkan kedua metode enkripsi diatas dalam merancang *Secure* SWF *Rich Internet Application.*

Algoritma Hash SHA 256 cukup sesuai dimanfaatkan untuk keperluan proses *secure autentification* pada aplikasi. Sedangkan penggunaan enkripsi c*ipher* simetrik dan asimetrik hanya cocok untuk proses pengenkripsian data dengan kunci publik berada disisi klien. Selain itu penggunaan *cipher* simetrik dan asimetrik masih membutuhkan konsiderasi terkait masalah performansi aplikasi SWF.

***Kata kunci**:* Flash, Aplikasi SWF, *actionscript*, algoritma Hash SHA 256, *Cipher* Simetrik dan Asimetrik.

1 Pendahuluan

Seiring dengan perkembangan dunia internet yang pesat mulai dari infrastruktur jaringan hingga perkembangan dalam hal rekayasa perangkat lunak, berbagai *platform* sebagai fasilitator aplikasi berbasis web juga berkembang begitu pesat. Dari sekian banyak vendor perangkat lunak yang mengembangkan teknologi untuk mendukung aplikasi berbasis web, salah satu yang cukup sukses adalah *Adobe Systems Incorporated* dengan teknologi *runtime* Adobe Flash. Berdasarkan survey yang dilakukan oleh Adobe, ditemukan bahwa 99% pengguna Intenet di dunia memiliki *runtime* flash player pada masing-masing *Web Browser* mereka. Hal ini bermakna bahwa secara umum *Web based application* yang dikembangkan dengan target *runtime* diatas *Flash player* dapat berjalan dengan baik pada setiap *web browser client.*

Pada kenyataannya memang tidak dapat dipungkiri bahwa Aplikasi berbasis Flash memang cukup populer di dunia. Mulai dari *mini internet games* hingga aplikasi bisnis, yang bermakna bahwa pengembangan Aplikasi berbasis flash ini memiliki nilai bisnis yang cukup menjanjikan.

Flash Platform merupakan seperangkat teknologi *cross platform* yang terdiri dari ***Adobe Flash*** dan ***Flex*** sebagai tools dan Framework proses pengembangan aplikasi, serta ***Adobe Flash Player*** dan ***Air*** sebagai runtime (*virtual machine)* yang memungkinkan aplikasi berjalan sebagai *Web Application* pada *Web Browser* manapun serta sebagai *Desktop Application* pada *Operating system* Apapun.

Aplikasi berbasis *runtime* Flash memiliki ekstensi SWF, sehingga aplikasi jenis ini dapat disebut juga sebagai aplikasi SWF. Suatu

aplikasi yang kompetitif haruslah handal (*reliable*). Untuk menghasilkan aplikasi SWF yang handal dan aman maka diperlukan suatu mekanisme pengamanan terhadap *Unauthorized access to data in transit* serta *unauthorized access to local data.*

Untuk mencapai hal ini, terdapat beberapa cara pengamanan melalui proses autentifikasi dengan dukungan kriptografi yang diantaranya adalah penggunaan algoritma Hash dan pengunaan *Cipher* Simetrik dan Asimetrik. Algoritma hash sangat berguna jika *user* ingin melakukan pengamanan pesan secara irreversible. Sedangakan metode *Cipher* simetrik dan asimetrik berguna unutk pengaman yang bersifat reversible.

Mengingat bahwa aplikasi SWF merupakan aplikasi yang dapat di-*decompiled.* Maka perlu strategi bijak untuk memanfaatkan kedua metode enkripsi diatas dalam merancang *Secure* SWF *Rich Internet Application.*

Pada makalah penulis akan memaparkan strategi penerapan kedua metode enkripsi diatas secara aplikatif pada aplikasi Flash. Selain itu penulis juga mencoba mengimlementasikan *libary* pada ***Flex*** *framework* yang dapat dimanfaatk untuk menghasilkan *secure SWF Application.*

Dalam makalah ini penulis mencoba membuat sebuah aplikasi tester sederhana unutuk menguji fungsionalitas ***library*** enkripsi dalam bahasa ***actionscript*** *3,* bahasa yang dipergunakan untuk pengembangan aplikasi di lingkungan flash. Sedangkan untuk pendefinisian layout antarmuka aplikasi, penulis mempergunakan sintaks ***mxml***, yang merupakan bahasa yang dikenal dilingkungan ***Adobe flex framework***.

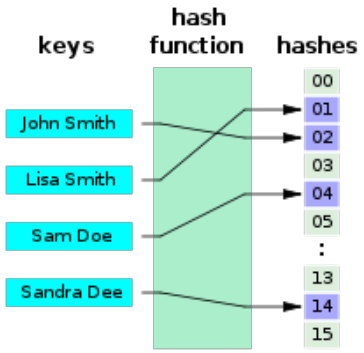
2 Teknik Enkripsi Data

2.1 Algoritma Hash

Fungsi Hash merupakan sebuah fungsi matematis yang dapat mengkonversi data berukuran besar maupun bervariasi kedalam sebuh datum berukuran kecil, biasanya sebuah dapat berupa bilangan integer yang dapat dijadikan sebagai indeks suatu array (larik). Nilai yang dikembalikan oleh suatu fungsi hash disebut nilai hash, kode hash, jumlah hash, atau hash.

Fungsi hash paling sering dipergunakan unutk mempercepat proses pembacaan data pada tabel atau proses perbandingan data – seperti pencarian data pada basis data, pendektekisan dulikasi *record* pada suatu file besar, pencarian kesamaan urutan pada DNA, dan lain-lain.

Fungsi hash bisa saja memetakan 2 atau lebih kunci ke nilai hash yang sama. Pada banyak aplikasi, sangat dituntut untuk meminimasi terjadinya hal semacam itu yang sering disebut sebagai *collision. Sejauh ini* desain fungsi hash yang baik memang masih menjadi topik hangat dalam penelitian.



Gambar 1 Fungsi Hash yang sempurna

Hashing merupakan suatu metode unutk melakukan enkripsi yang bersifat satu arah (*non-reversible*) yang mamapu mentransformasikan teks asli kesebuah string unik, atau hash. Karena hash bersifat *non-reversible*, maka tidak ada cara untuk mengkonversi balik nilai hash ke teks aslinya semula. Algoritma ini sangat bermanfaat unutk menagani proses tertentu seperti otentifikasi password. Pada banyak aplikasi, password diproses melalui algoritma hash dan kemudian nilai hash tersebut disimpan pada *storage*. Ketika seorang pengguna aplikasi hendak melakukan proses otentifikasi pada aplikasi tersebut, nilai hash dari password yang diberikan oleh user dibangkitkan untuk kemudian akan dibandingkan apakah bernilai sa,a dengan hash unik yang tersimpan pada *storage*. Jika bersesuaian, maka sipengguna dapat diberi hak akses terhadap aplikasi. Metodologi ini, memungkinkan administrator unutk menyimpan password dalam bentuk non-plaintext tanpa perlu khwatir terhadap resiko pencurian atau pendekripsian password .

Ada beberapa macam algoritma hash, algoritma hash yang direkomendasikan oleh Adobe untuk dimanfaatkan adalah SHA 256 Karena dinilai cukup aman terhadap serangan kriptografi.

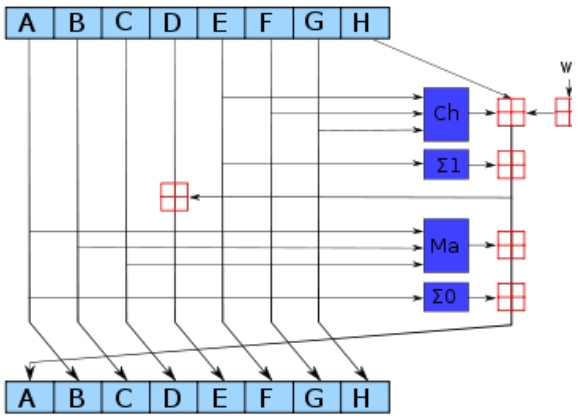
Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)
SHA-0		160	160	512	$2^{64} - 1$
SHA-1		160	160	512	$2^{64} - 1$
SHA-2	*SHA-256/224*	256/224	256	512	$2^{64} - 1$
	SHA-512/384	512/384	512	1024	$2^{128} - 1$

Algorithm and variant		Word size (bits)	Rounds	Operations	Collisions found
SHA-0		32	80	+,and,or, xor,rot	Yes
SHA-1		32	80	+,and,or, xor,rot	None (2^{52}[10] attack)
SHA-2	*SHA-256/224*	32	64	+,and,or,xor,shr,rot	None
	SHA-512/384	64	80 +	+,and,or,xor,shr,rot	None

Bagan 1Tabel perbandingan beberapa Fungis Hash

2.1.1 Algoritma Hash SHA 256

As one example, as3corelib started as an Adobe Labs project and is now part of the Google Code project. It provides several utilities for JSON, web services, and cryptography. The cryptography library includes methods for supporting SHA256, SHA224, SHA1, MD5, and the WSSEUsernameToken. Adobe recommends using SHA256 in place of SHA1 or MD5 whenever possible—due to known cryptographic attacks against the hashing algorithms. Corelib requires libraries from the free Flex 3 SDK to perform the base-64 encoding:



Gambar 2 Satu Iterasi dalam sebuah keluarga fungsi kompresi *SHA-2*

SHA-256 mempergunakan 6 fungsi lojik, yang setiap fungsi tersebut beroperasi pada 32-bit *words* yang direperesentasikan sebagai *x*, *y*, and *z*. Hasil dari setipa fungsi merupakan sebuha 32-bit *word* baru.

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$
$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$
$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$
$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

Menurut definisi *secure hash standard* yang dikeluarkan oleh ***Federal Information Processing Standards Publication 180-2***, ROTL adalah

The rotate left (cicrular left shift) operation, ROTL $^n(x)$, where x is a w-bit word and n is an integer with $0 \le n < w$, is define by

$$ROTL^n(x) = (x << n) \vee (x >> w - n).$$

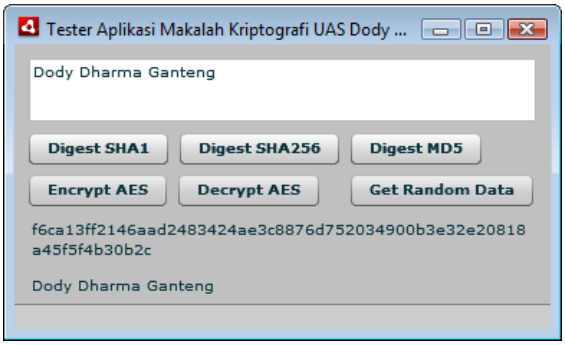
Thus ROTL $^n(x)$ is wquivalent to a circular shift(rotation) of x by n position to the left.

Berikut Contoh pemanfaatan SHA256 pada kode *ActionScript3* untuk pengembangan SWF Application dengan mempergunakan *framework* ***Adobe flex 3*** dengan memanfaatkan *library* AS3Corelib :

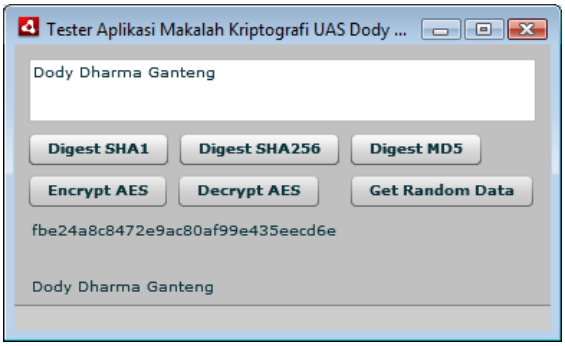
```
// Import library  SHA256 dari CoreLib
import com.adobe.crypto.SHA256;

// Inisialisasi string yang akan di-hash
var myString:String = "HashMe";

// Ciptakan  sebuah representasi hex dari
// Hash  SHA256
var hexHash:String = SHA256.hash
(myString);

// Ciptakan sebuah represensi basis 64
// dari hash 256
var base64Hash:String =
SHA256.hashToBase64(myString);
```

Bahkan pada release terakhir, sebuah proyek dari Adobe labs , *alchemy*, memungkinkan para developer untuk melakukan *porting* aplikasi dengan bahas C ke *actionscript*. Porting ini juga mendukung SHA, SHA2, dan MD5.

Berikut adalah contoh aplikasi yang Saya rancang sendiri unutk melakukan pengujian fungsi-fungsi enkripsi dari *library AS3Lib.* Aplikasi berikut dikembangkan dengan framework ***Adobe Flex*** dan berjalan diatas *runtime* **Adobe Air.**

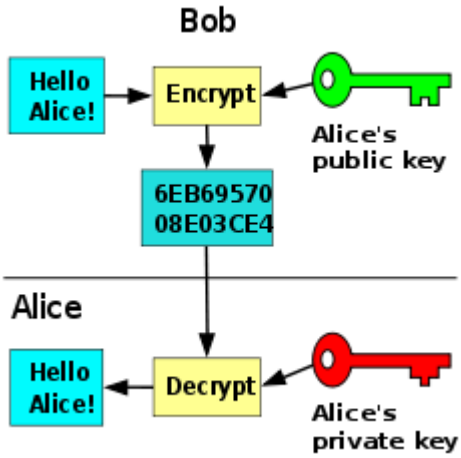


Gambar 3 Aplikasi testing yang mengkonversi string "Dody Dharma Ganteng" ke nilai Hash SHA 256



Gambar 4 Aplikasi testing yang mengkonversi string "Dody Dharma Ganteng" ke MD5

2.2 Chiper Simetrik dan Asimetrik



Gambar 5ilustrasi Enkripsi Kunci Asimetrik

Berbeda dengaAlgoritma Hash, *Cipher* Simetrik dan asimetrik berguna ketika Kita hendak membuat sebuah enkripsi data yang bersifat *reversible* . Beberapa implementasi dari *Cipher* simetrik dan asimetrik yang ditulis dalam bahasa *actionscript* dapat ditemukan di web. Hal ini sangat berguna untuk dimanfaatkan, namun di sisi lain para *developer* harus tetap waspada, berhubung *file* SWF dapat di-*decompile*. Sehingga rahasia serta kunci private untuk aplikasi ini tidak boleh disimpan di dalam *file* itu sendiri. Setidaknya, kunci rahasia simetrik dan kunci privat asimetrik dibuat sedemikian rupa sehingga baru dapat diperoleh dengan mekanisme *download* melalui *secure internet protocol* atau dibangkitkan oleh apliaksi SWF pada mesin *client* untuk mencegah kebocoran.

2.2.1 Resiko celah Keamanan dari Kunci Enkripsi pada file SWF

Terdapat beberapa resiko celah keamanan jika kita mempergunakan kunci rahasia simetrik maupun asimetrik dalam file SWF. Jika kunci simetrik disimpan pada sebuah *shared object*, maka kita harus ingat bahwa *shared object* merupakan objek yang tidak terenkripsi dan bisa saja dibaca oleh *user* atau oleh *malicious software* yang ada pada mesin *user*. Kehati-hatian juaga diperlukan sebelum melakukan perubahan pada konfigurasi *default localpath* karena hal ini dapat semakin meningkatkan jumlah file SWF yang dapat membaca data dari *shared object*. Kita dapat men-*setting seure flag* pada dengan nilai true untuk membatasi akses terhadap *shared object* hanya terbatas pada Aplikasi SWF yang *secure*. Suatu File SWF yang di-*load* melalui protocol HTTP dapat dikategorikan kedalam aplikasi beresiko *middle-attack* dan jika suatu *attacker* berhasil me-*replace* file SWF saat sedang transit, maka *file* SWF yang bersifat *malicious* akan mampu mencuri kunci dari *shared object. File* eksternal yang di-host pada domain yang sama dengan aplikasi yang memanfaatkannya, yang diimpor kedalam file SWF terenkripsi maupun file SWF eksternal lain yang diberikan hak *permission* melalaui setting `Security.allowDomain()`, dapat juga diberikan *cross-scripting permissions* untuk memperolehdari file SWF tersebut.

Jika Kita hendak memanfaatkan cipher asimetrik, sangat memungkinkan uutk menyimpan kunci publik dalam SWF file dengan aman. Kunci publik dapat dimanfaatkan untuk mengenkripsi data terkait dengan lebih dulu mengirimkan kunci publik ke server aplikasi yang berkorespondensi dengan kunci privat. Pendekatan ini memungkinkan komunikasi satu arah dari data terenkripsi menuju server serta dapat juga dipergunakan unutk melakukan verifikasi data bertanda tangan digital yang dikirimkan dari server ke file SWF klien. Satu contoh yang mungkin berguna, yaitu dalam situasi seperti saat proses pengumpulan data yang bersifat sensitf seperti *password*,dengan

dilanjutkan dengan pengiriman *password* tersebut ke *server*. Karena file SWF dapt berisi kunci publik dan kunci publik hanya dipergunakan unutk proses enkripsi, maka tidaklah menjadi masalah jika seseorang melakukan proses *decompile* terhadap file SWF dalam rangka mencuri kunci tersebut. Para *attacker* bisa saja mengirinkan data *malicious* yang terenkripsi ke server, namun seorang *server programmer* tetap harus melakukan validasi data walaupun proses enkripsi dipergunakan.

Terdapat pertimbangan masalah performa aplikasi ketika memanfaatkan kriptografi, sehingga dalam proses konsiderasi, diperlukan pengujian performa terhadap aplikasi jika hendak berniat mempergunakan kriptografi secara ekstensif. Sebagai ilustrasi, enkripsi RSA dinilai lebih cepat ketimbang dekrisi RSA. Dalam banyak kasus, dinilai lebihmudah dan aman untuk memanfaatkan protokol HTTPS unutk mengirimkan data terenkripsi ketimbang melakukan enkripsi data secara langsung pada kode ***actionscript***.

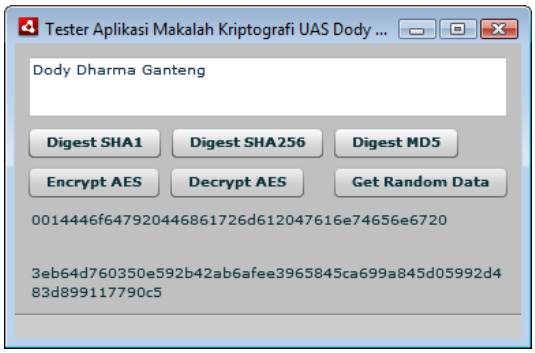
Library enkripsi AES dan RSA dapat ditemukan pada beberapa *framework* , termasuk pada ***Alchemy ActionScript 3 Crypto Wrapper library*** yang dapat diperoleh di ***Adobe Labs*** dan ***AS3Crypto Framework*** *open-source*.

Kode dibawah ini mencoba mendemonstrasikan pengguna algoritma Enkripsi RSA untuk mengenkripsi data dengan memanfaatkan *library* ***AS3Crypto Framework*** yang bersifat *open source*. ***AS3Crypto Framework*** mendukung enkripsi RSA, AES, 3DES, Blowfish, SHA, MD5, HMAC, dan beberapa solusi kriptografi. Kode ini ditujukan untuk mendemonstrasikan bahwa kriptografi kunci publik sangat mungkin dilakukan walupun dengan solusi *open-source*.

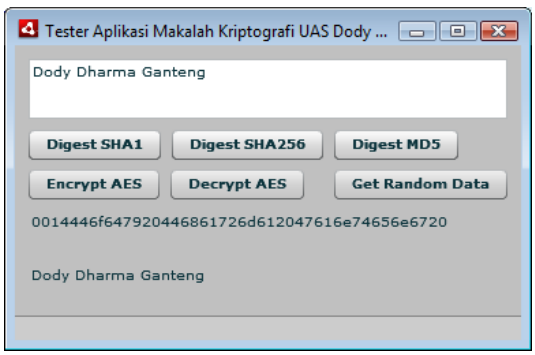
Catatan: *Library* ***AS3Crypto*** bukanlah milik ***Adobe Incorporated*** sehingga ***Adobe*** tidak punya tanggung jawa untuk mengembangkan *library* tersebut. Para *Developer* harus mampu memilih *library* yang cocok untuk keperluan mereka masing-masing.

```
import com.hurlant.crypto.rsa.RSAKey;
import flash.utils.ByteArray;
import com.hurlant.util.Hex;
import com.hurlant.util.der.PEM;

// Kunci public tanpa spasi atau
// carriage returns
// sebuah kunci publik
// openssl dibangkitkan oleh :
// openssl rsa -pubout –dalam
// private_key.pem -out public_key.pem
// harus bekerja pada baseline testing
```

```
var myPEMPublicKeyString = "-----BEGIN
PUBLIC KEY-----{BASE64 DATA with no
spaces or return characters}-----END
PUBLIC KEY-----";

// masukkan data yang akan dienkripsi
// dalam array byte

var data:ByteArray =
Hex.toArray(Hex.fromString("MyInputString
"));

// Tujuan ByteArray yang akan mengandung
// data terenkripsi
var encryptedResult:ByteArray = new
ByteArray;

// Set RSAKey dan encrypt data data
var rsa:RSAKey = PEM.
readRSAPublicKey(myPEMPublicKeyString);
rsa.encrypt(data, encryptedResult,
data.length);

// Konversi data terenkripsi kedalam hex
// encoded string untuk ditransportasikan
// pada client tujuan, sisi lain koneksi
// dapat mengkonversi hex tersbut
// kembali keda;am binary sebelum
// didekripsi

hexEncryptedResult =
Hex.fromArray(encryptedResult);
```



Gambar 6 Contoh Proses Enkripsi Asimetrik String "Dody Dharma Ganteng"



Gambar 7 Contoh Dekripsi String Dody Dharma Ganteng Dengan Kunci "3eb64d760350e592b42ab6afee3965845ca 699a845d05992d483d899117790c".

3 Hasil dan Pembahasan

Dari uraian sebelumnya, Algoritma Hash dinilai cukup efektif dipergunakan untuk keperluan proses autentifikasi pengguna sistem. Dalam hal ini Algoritma hash yang cukup sesuai dan aman adalah SHA 256. Karena pada algoritma ini tidak terdapat kemungkinan untuk terjadi *collision* sehingga dapat dipastikan setiap proses hashing dari data yang berbeda akan menghasilkan kode hash yang unik. Kerahasiaan data yang terenkrispi juga cukup terjamin, karena proses enkripsi ini tidak dapat di-*reverse.* Namun metode semacam ini tidak cocok untuk proses transmisi data rahasia dari suatu mesin ke mesin lain, mengingat data yang telah dienkripsi tidak dapat dikembalikan lagi. Proses ini hanya memungkinkan proses perbandingan kesamaan suatu data dengan membandingkan nilai hashnya.

Untuk mekanisme cipher simetrik dan asimetrik terlihat bahwa cukup riskan jika para developer menyimpan data-data rahasia, kunci privat, dan data sensitif lainnya di dalam suatu *file* SWF. Mengingat bahwa *file* semacam ini dapat didekompilasi, sehingga memungkinkan pihak-pihak yang tidak berhak untuk mengakses data rahasia tersebut. Hal yang paling setidaknya dianggap aman adalah menyimpan kunci publik di dalam aplikasi untuk sekedar keperluan enkripsi data. Selain itu penggunaan kriptografi semacam ini secara ekstensif akan cukup mempengaruhi performansi aplikasi. Sehingga penggunaan metode kriptografi seperti ini membutuhkan konsiderasi secara serius oleh developer. Cara yang mungkin sejauh ini dianggap cukup sederhana dan mudah adalah dengan memanfaatkan protokol *HTTP secure (*HTTPS*)*.

4 Simpulan

Algoritma Hash SHA 256 cukup sesuai dimanfaatkan untuk keperluan proses *secure autentification* pada aplikasi. Sedangkan penggunaan enkripsi c*ipher* simetrik dan asimetrik hanya cocok untuk proses pengenkripsian data dengan kunci publik berada disisi klien. Selain itu penggunaan *cipher* simetrik dan asimetrik masih membutuhkan konsiderasi terkait masalah performansi aplikasi.

5 Daftar Pustaka

[1] *Cole, Alaric*. 2008. ***Learning Flex 3-Getting up Speed with Rich Intenet Application***. Adobe Developer Library.

[2] (2002).***Secure Hash Standard***.Federal Information Processing Standard Publication.

[3] *http://opensource.adobe.com/wiki/display/flexsdk/Flex+SDK*

[4] *http://labs.adobe.com/wiki/index.php/Alchemy:Libraries*

[5] *http://www.adobe.com/devnet/*

6 Lampiran

Aplikasi berbasis Flash platform yang dicoba penulis dengan *framework* ***Adobe Flex*** memanfaatkan *library* **AS3CoreLib**.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
<mx:Script>
	<![CDATA[

		import alchemy.samples.as3_crypto_wrapper;

		private var aes128key:ByteArray = null;
		private var aes128iv:ByteArray = null;

		private function toBase16(input:ByteArray):String
		{
			var output:String = new String;
			const hex:String = "0123456789abcdef";
			const length:int = input.length;
			for(var i:int = 0; i < length; i++) {
				output += hex.charAt((input[i] & 0xF0) >> 4);
				output += hex.charAt(input[i] & 0x0F);
			}
			return output;
		}

		private function fromBase16(input:String):ByteArray
		{
			var output:ByteArray = new ByteArray;
			const hex:Object =
{'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9,'a':10,'b':11,'c':12,'d':13,'e':1
4,'f':15,'A':10,'B':11,'C':12,'D':13,'E':14,'F':15};
			const length:int = input.length;
			var i:int = 0;
			while(i < length) {
				var byte:uint = (hex[input.charAt(i)] << 4);
				i++;
				if(i < length) {
					byte += hex[input.charAt(i)];
					i++;
				}
				output.writeByte(byte);
			}
			return output;
		}

		private function initKeyAndIV():void
		{
			as3_crypto_wrapper.prng_seed(null);
			if(null == aes128key)
				aes128key = as3_crypto_wrapper.prng_random(16);
			if(null == aes128iv)
				aes128iv = as3_crypto_wrapper.prng_random(16);
		}

		private function onSHA1():void
		{
			var buffer:ByteArray = new ByteArray;
			buffer.writeUTF(dataText.text);
			var hdl:uint = as3_crypto_wrapper.sha1_begin();
			as3_crypto_wrapper.sha1_update(hdl, buffer);
			var digest:ByteArray = as3_crypto_wrapper.sha1_finish(hdl);
			outputText.text = toBase16(digest);
		}

		private function onSHA256():void
		{
			var buffer:ByteArray = new ByteArray;
			buffer.writeUTF(dataText.text);
			var hdl:uint = as3_crypto_wrapper.sha256_begin();
			as3_crypto_wrapper.sha256_update(hdl, buffer);
			var digest:ByteArray = as3_crypto_wrapper.sha256_finish(hdl);
			outputText.text = toBase16(digest);
		}
```

```
                private function onMD5():void
                {
                        var buffer:ByteArray = new ByteArray;
                        buffer.writeUTF(dataText.text);
                        var hdl:uint = as3_crypto_wrapper.md5_begin();
                        as3_crypto_wrapper.md5_update(hdl, buffer);
                        var digest:ByteArray = as3_crypto_wrapper.md5_finish(hdl);
                        outputText.text = toBase16(digest);
                }

                private function onEncryptAES():void
                {
                        var buffer:ByteArray = new ByteArray;
                        buffer.writeUTF(dataText.text);
                        initKeyAndIV();
                        var hdl:uint = as3_crypto_wrapper.aes128_cbc_encrypt_begin(aes128key,
aes128iv);
                        var encrypted:ByteArray =
as3_crypto_wrapper.symmetric_encrypt_update(hdl, buffer);
                        encrypted.writeBytes(as3_crypto_wrapper.symmetric_encrypt_finish(hdl));
                        outputText.text = toBase16(buffer);
                        encrypted.position = 0;
                        outputText2.text = toBase16(encrypted);
                }

                private function onDecryptAES():void
                {
                        var buffer:ByteArray = fromBase16(outputText2.text);
                        var hdl:uint = as3_crypto_wrapper.aes128_cbc_decrypt_begin(aes128key,
aes128iv);
                        var decrypted:ByteArray =
as3_crypto_wrapper.symmetric_decrypt_update(hdl, buffer);
                        decrypted.writeBytes(as3_crypto_wrapper.symmetric_decrypt_finish(hdl));
                        outputText.text = toBase16(decrypted);
                        decrypted.position = 0;
                        outputText2.text = decrypted.readUTF();
                }

                private function onGetRandomData():void
                {
                        as3_crypto_wrapper.prng_seed(null);
                        var buffer:ByteArray = as3_crypto_wrapper.prng_random(20);
                        dataText.text = toBase16(buffer);
                }

        ]]>
</mx:Script>

        <mx:TextArea id="dataText" right="10" left="10" top="10"/>
        <mx:Text id="outputText" text="&lt;digest&gt;" right="10" y="121" left="10"/>
        <mx:Text id="outputText2" text="&lt;digest&gt;" right="10" y="160" left="10"/>
        <mx:Button x="10" y="62" label="Digest SHA1" id="sha1" click="onSHA1()"/>
        <mx:Button x="115" y="62" label="Digest SHA256" id="sha256" click="onSHA256()"/>
        <mx:Button x="234" y="62" label="Digest MD5" id="md5" click="onMD5()"/>
        <mx:Button x="10" y="62" label="Digest SHA1" click="onSHA1()"/>
        <mx:Button x="115" y="62" label="Digest SHA256" click="onSHA256()"/>
        <mx:Button x="234" y="62" label="Digest MD5" click="onMD5()"/>
        <mx:Button x="10" y="91" label="Encrypt AES" click="onEncryptAES()"/>
        <mx:Button x="114" y="91" label="Decrypt AES" click="onDecryptAES()"/>
        <mx:Button x="234" y="91" label="Get Random Data" click="onGetRandomData()"/>

</mx:WindowedApplication>
```