Eksplorasi Java Cryptography Architecture (JCA) dan Implementasinya Pada Perangkat Android

Ni Made Satvika Iswari - 13508077[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]if18077@students.if.itb.ac.id

***Abstrak*—Java Cryptography Architecture (JCA) merupakan bagian utama dari platform Java. JCA terdiri dari arsitektur *provider* dan sekumpulan API untuk *digital signatures, message digests* (*hash),* sertifikasi dan validasi sertifikat, enkripsi (blok simetri/asimetri), *key generation* dan manajemen, dan *secure random number generation*. API tersebut memungkinkan pengembang untuk mengintegrasikan dengan mudah keamanan ke aplikasi yang dikembangkan. Pembahasan mengenai JCA akan dititikberatkan pada kelas Cipher, yang menyediakan fungsionalitas kriptografi yaitu enkripsi dan dekripsi pesan. Android adalah sistem operasi untuk telepon seluler yang berbasis Linux. Android menyediakan platform terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri yang digunakan oleh bermacam peranti bergerak. Untuk melakukan proses enkripsi/dekripsi pesan, Android menggunakan JCA, yaitu dengan menggunakan package javax.crypto.*. Pada makalah ini, penulis akan melakukan pembahasan mengenai JCA dan mengimplementasikannya pada perangkat Android. Adapun algoritma enkripsi yang digunakan adalah DES, mode ECB, padding PKCS5Padding. Kunci yang akan digunakan akan dibangkitkan dengan menggunakan fungsi hash MD5 terlebih dahulu.**

***Kata Kunci*—Java Cryptography Architecture, Kelas Cipher, Android**

I. PENDAHULUAN

Platform Java menekankan kuat pada keamanan, termasuk keamanan bahasa, kriptografi, infrastruktur *public key*, *authentication*, komunikasi yang aman, dan *access control*.

Java Cryptography Architecture (JCA) merupakan bagian utama dari platform Java. JCA terdiri dari arsitektur *provider* dan sekumpulan API untuk *digital signatures, message digests* (*hash),* sertifikasi dan validasi sertifikat, enkripsi (blok simetri/asimetri), *key generation* dan manajemen, dan *secure random number generation*. API tersebut memungkinkan pengembang untuk mengintegrasikan dengan mudah keamanan ke aplikasi yang dikembangkan.

Prinsip dari JCA adalah sebagai berikut :

1. *Implementation Independence*, yaitu aplikasi tidak perlu mengimplementasikan algoritma keamanan, melainkan dapat meminta layanan keamanan dari platform Java.
2. *Implementation Interoperability*, yaitu sebuah aplikasi tidak terikat pada *provider* yang spesifik, begitu pula sebalikanya.
3. *Algorithm Extensibility*, platform Java mendukung instalasi dari *custom providers* yang mengimplementasi layanan.

Android adalah sistem operasi untuk telepon seluler yang berbasis Linux. Android menyediakan platform terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri yang digunakan oleh bermacam peranti bergerak.

Untuk melakukan proses enkripsi/dekripsi pesan, Android menggunakan JCA, yaitu dengan menggunakan *package*:

```
javax.crypto.*
```

Package ini menyediakan *class* dan *interface* untuk aplikasi kriptografi yang mengimplementasikan algoritma untuk melakukan enkripsi, dekripsi, atau *key agreement*. Implementasi cipher yang berasal dari *provider* yang berbeda dapat diintegrasikan dengam menggunakan kelas abstrak SPI (*Service Provider Interface*). Dengan kelas *SealedObject*, pengembang dapat mengamankan sebuah objek dengan mengenkripsinya dengan sebuah cipher.

Mengembangkan aplikasi untuk perangkat Android akan difasilitasi oleh sekumpulan *tools* yang disediakan dengan *Software Development Kit* (SDK). Pengembang dapat mengakses *tools* ini melalui *plugin* Eclipse yang bernama ADT (*Android Development Tools*) atau melalui *command line*. Pengembangan dengan menggunakan Eclipse lebih dipilih karena dapat meminta langsung *tools* yang dibutuhkan ketika mengembangkan aplikasi. Namun, pengembang dapat memilih untuk mengembangkan aplikasi dengan menggunakan IDE yang lain atau *text editor* sederhana.

II. ARSITEKTUR JCA

Cryptographic Service Providers (CSP)

Isitilah CSP dan *provider* akan digunakan bergantian

pada makalah ini, istilah ini mengacu pada *package* atau kumpulan *package* yang mengimplementasi satu atau lebih layanan kriptografi.

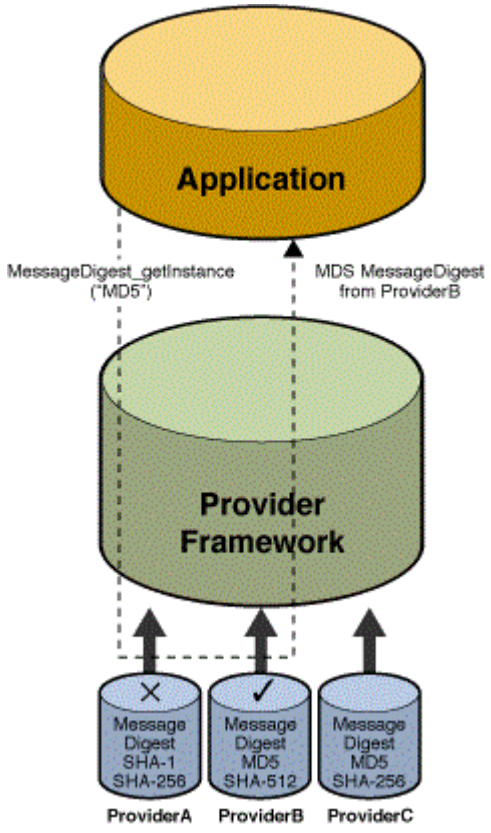
`java.security.Provider`

adalah kelas dasar untuk seluruh *provider* keamanan. Setiap CSP mengandung *instance* dari kelas ini yang mengandung nama *provider* dan daftar dari seluruh layanan keamanan/ algoritma yang diimplementasikan.

Setiap instalasi JDK akan memiliki satu atau lebih *provider* yang terinstall dan terkonfigurasi secara *default*. *Provider* tambahan dapat ditambahkan secara static maupun dinamik.

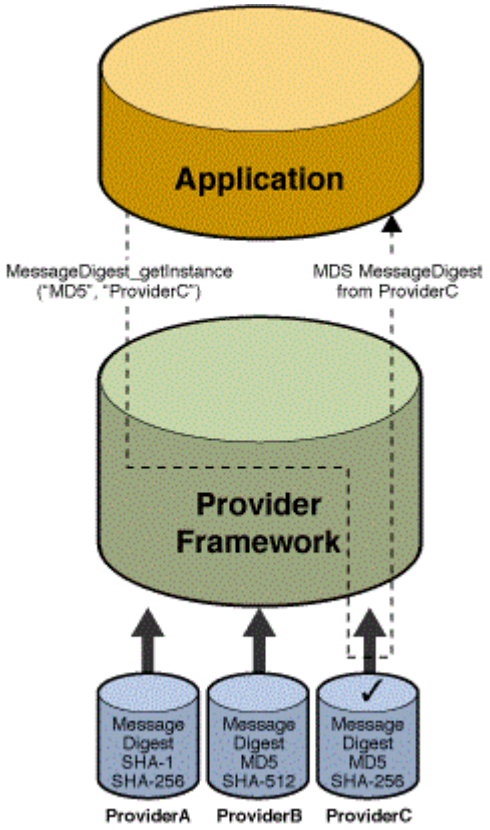
Untuk menggunakan JCA, sebuah aplikasi akan meminta tipe objek tertentu (misalnya `MessageDigest`) dan algoritma atau layanan tertentu (misalnya algoritma "MD5"), kemudian mendapatkan implementasi dari salah satu *provider* yang terinstall. Cara lainnya adalah, program, dapat meminta objek dari *provider* yang spesifik. Setiap *provider* memiliki nama yang digunakan untuk mengacunya.

```
md = MessageDigest.getInstance("MD5");
md = MessageDigest.getInstance("MD5",
"ProviderC");
```



Gambar 1 Implementasi Proses Meminta *Message Digest* "MD5" Tanpa Menspesifikkan Nama *Provider*

Gambar 1 menggambarkan implementasi proses meminta message digest "MD5". Gambar tersebut menunjukan tiga *provider* yang berbeda yang mengimplementasi algoritma *message digest* yang berbeda ("SHA-1", "MD5", "SHA-256", dan "SHA-512"). Pada gambar sebelah kiri, sebuah aplikasi meminta implementasi algoritma MD5 tanpa menspesifikkan nama *provider*. *Provider* akan dicari berdasarkan urutannya dan implementasi dari *provider* pertama yang menyediakan algoritma yang diminta, ProviderB, akan dikembalikan.



Gambar 2 Implementasi Proses Meminta Message Digest "MD5" Dengan Menspesifikkan Nama Provider

Pada gambar 2, aplikasi meminta implementasi algoritma MD5 dari *provider* yang spesifik, ProviderC. Pada kasus ini implementasi dari ProviderC akan dikembalikan, walaupun *provider* dengan urutan lebih awal, ProviderB, juga menyediakan implementasi MD5.

JCA menawarkan sekumpulan API yang memungkinkan pengguna untuk menanyakan *provider* mana yang terinstall dan layanan apa yang disediakan. Arsitektur ini juga memudahkan pengguna untuk menambahkan *provider* yang diinginkan.

Bagaimana *Provider* Sebenarnya Diimplementasikan?

Instance dari *engine class* adalah yang sesuatu yang didukung oleh implementasi kelas yang memiliki *method signature* yang sama. Panggilan aplikasi akan diarahkan melalui *engine class* dan akan diantarkan ke implementasi dasar. Implementasi tersebut menangani permintaan dan mengembalikan hasil yang sesuai.

Method aplikasi API pada setiap *engine class* diarahkan pada implementasi *provider* melalui *class* yang mengimplementasikan *Service Provider Interface* (SPI) yang sesuai. Sehingga, untuk setiap *engine class*, terdapat

abstract SPI class yang sesuai yang menjelaskan method yang harus diimplementasikan oleh setiap algoritma CSP. Nama dari setiap kelas SPI adalah sama seperti *engine class* yang bersesuaian, diikuti dengan `Spi`. Sebagai contoh, *engine class* `Signature` menyediakan akses ke fungsionalitas dari algoritma *digital signature*. Implementasi *provider* yang sebenarnya disediakan pada subkelas dari `SignatureSpi`. Aplikasi akan memanggil method API *engine class*, yang saat mengembalikan akan memanggil method SPI pada implementasi yang sebenarnya.

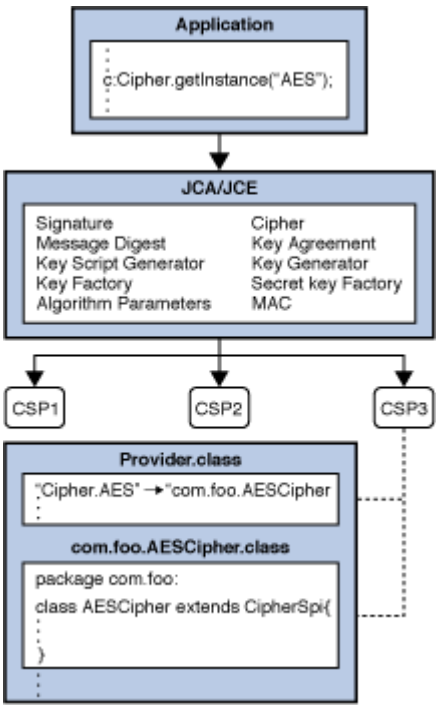
Setiap kelas SPI adalah abstrak. Untuk menyediakan implementasi jenis layanan tertentu untuk algoritma tertentu, *provider* harus membuat subkelas dari kelas SPI yang bersesuaian dan menyediakan implementasi untuk semua method abstrak.

Untuk setiap *engine class* pada API, implementasi *instance* diminta dan diinstansiasi dengan memanggil `getInstance()` pada *engine class*. Method tersebut adalah static dan mengembalikan *instance* dari kelas. *Engine class* menggunakan mekanisme pemilihan *framework provider* seperti yang telah dijelaskan sebelumnya untuk memperoleh dukungan implementasi sesungguhnya (SPI), lalu membuat objek *engine* yang sesungguhnya. Setiap instans dari *engine class* mengenkapsulasi (sebagai bagian private) instans dari kelas SPI yang bersesuaian, yang disebut sebagai objek SPI. Seluruh method API dari objek API dideklarasikan sebagai objek final.

Berikut adalah ilustrasinya:

```
import javax.crypto.*;

Cipher c = Cipher.getInstance("AES");
c.init(ENCRYPT_MODE, key);
```



Gambar 3 Implementasi Provider

Pada gambar di atas, sebuah aplikasi menginginkan instance dari `javax.crypto.Cipher` berupa "AES", dan tidak perduli provider mana yang digunakan. Aplikasi tersebut memanggil method `getInstance`() dari *engine class* Cipher, yang ketika mengembalikan akan bertanya pada framework JCA untuk menemukan instance *provider* pertama yang mendukung "AES". Framework akan berkonsultasi pada setiap *provider* yang terinstall, dan memperoleh instance *provider* dari kelas `Provider`. (Mengingat kembali bahwa kelas `Provider` adalah basis data dari algoritma yang tersedia) Framework mencari setiap *provider*, dan akhirnya menemukan entri yang cocok pada CSP3. Entri basis data ini menunjuk pada kelas implementasi `com.foo.AESCipher` yang mengekstensi `CipherSpi`, dan dengan demikian cocok untuk digunakan oleh *engine class* `Cipher`. Instance dari `com.foo.AESCipher` akan terbentuk, dan dienkapsulasi pada instance baru dari `javax.crypto.Cipher`, yang akan dikembalikan ke aplikasi. Ketika aplikasi melakukan operasi `init()` pada instance `Cipher`, *engine class* `Cipher` akan menunjuk permintaan tersebut ke method `engineInit()` yang sesuai pada kelas `com.foo.AESCipher`.

III. KONSEP JCA

Engine Class menyediakan interface untuk layanan kriptografi yang spesifik, tidak bergantung pada algoritma kriptografi atau provider. Kelas ini menyediakan :

- Operasi kriptografi (enkripsi, *digital signature*, *message digest*, dsb),
- Generator atau converter dari material kriptografi (kunci dan parameter algoritma), atau
- Objek (kunci yang tersimpan atau sertifikat) yang mengenkapsulasi data kriptografi dan dapat digunakan pada layer lebih atas pada abstraksi.

Kelas – kelas yang tersedia adalah sebagai berikut :

- `SecureRandom` : digunakan untuk membangkitkan nomor acak atau semi-acak
- `MessageDigest` : digunakan untuk mengkalkulasi *message digest* (hash) dari data tertentu.
- `Signature` : diinisialisasi dengn kunci, digunakan untuk menandai data dan memeriksa *digital signature*
- `Cipher` : diinisialisasi dengan kunci, digunakan untuk mengenkripsi/mendekripsi data
- Message Authentication Codes (MAC) : seperti MessageDigest, juga membangkitkan nilai hash, namun pertama kali diinisialisasi dengan kunci untuk melindungi integritasi pesan
- `KeyFactor` : digunakan untuk mengkonversi kunci kriptografi yang acak dengan tipe Key ke kunci yang ditetapkan, dan sebalikanya
- `SecretKeyFactory` : digunakan untuk mengkonversi kunci kriptografi yang acak dengan tipe SecretKey ke kunci yang ditetapkan, dan sebaliknya.

- `KeyPairGenerator` : digunakan untuk membangkitkan pasangan baru kunci public dan private yang cocok untuk digunakan dengan algoritma yang ditentukan
- `KeyGenerator` : digunakan untuk membangkitkan kunci rahasia baru untuk digunakan dengan algoritma yang ditentukan
- `KeyAgreement` : digunakan oleh lebih dari satu pihak untuk menyepakati dan menentukan sebuah kunci khusu untuk digunakan untuk operasi kriptografi tertentu
- `AlgorithmParameters` : digunakan untuk menyimpan parameter untuk algoritma tertentu, termasuk parameter encoding dan decoding
- `KeyStore` : digunakan untuk membuat dan mengatur keystore
- `CertificateFactory` : digunakan untuk membuat sertifikat public key dam Certificate Revocation Lists (CRLs)
- `CertificatePathBuilder` : digunakan untuk membangun rantai sertifikat (juga dikenal sebagai certification paths)
- `CertPathValidator` : digunakan untuk memeriksa rantai sertifikat
- `CertStore` : digunakan untuk mendapatkan kembali `Certifiacte` dam `CRL` dari repository

IV. KELAS UTAMA DAN *INTERFACE*

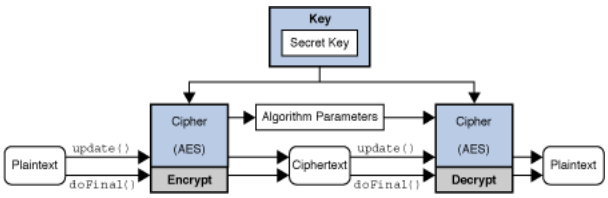
Berikut adalah daftar kelas – kelas utama dan *interface* yang disediakan pada JCA:

- Kelas : `Provider` dan `Security`
- *Engine Class* : `SecureRandom`, `MessageDigest`, `Signature`, `Cipher`, `Mac`, `KeyFactory`, `SecretKeyFactory`, `KeyPairGenerator`, `KeyGenerator`, `KeyAgreement`, `AlgorithmParameters`, `AlgorithmParameterGenerator` , `KeyStore`, dan `CertificateFactory`
- Kelas dan *Interface* : `Key`

Pada makalah ini, penulis akan membatasi pembahasan hanya pada *engine class* `Cipher`.

V. KELAS CIPHER

Kelas `Cipher` menyediakan fungsionalitas dari cipher kriprografi yang digunakan untuk enkripsi dan dekripsi. Enkripsi adalah proses mengambil data (disebut *plaintext*) dan kunci, dan memproduksi data (*ciphertext*) yang tidak berarti kepada pihak ketiga yang tidak mengetahui kuncinya. Dekripsi adalah proses inversi : mengambil *ciphertext* dan kunci untuk memproduksi *plaintext*.



Gambar 4 Proses Enkripsi dan Dekripsi Oleh Kelas Cipher

Membuat Objek Cipher

Objek Cipher diperoleh dengan menggunakan method `getInstance()`. Disini, nama algoritma sedikit berbeda dengan engine class yang lain, tidak hanya menyebutkan nama algoritma, namun juga "transformasi". Transformasi adalah string yang menjelaskan operasi (atau kumpulan operasi) untuk dilaksanakan pada input untuk menghasilkan output. Transformasi selalu mengandung nama dari algoritma kriptografi (misalnya DES), dan dapat disertai dengan mode dan skema padding.

Bentuk transformasi :

- "algoritma/mode/padding", atau
- "algortima"

Inisialisasi Objek Cipher

Objek Cipher yang diperoleh melalui method `getInstance` harus diinisialisasi untuk salah satu mode, yang didefinisikan sebagai konstanta final integer pada kelas `Cipher`. Mode tersebut diacu dengan nama simboliknya, seperti yang ditampilkan berikut :

- ENCRYPT_MODE : enkripsi data
- DECRYPT_MODE : dekripsi data
- WRAP_MODE : membungkus `java.security.Key` menjadi bytes sehingga kunci tersebut aman untuk diangkut.
- UNWRAP_MODE : membuka bungkus dari kunci yang dibungkus menjadi objek `java.security.Key`

Setiap inisialisasi Cipher akan mengambil parameter mode operasional (`opmode`), dan menginisialisasi objek Cipher untuk mode tersebut. Parameter lainnya adalah kunci (`key`) atau certificate yang mengandung kunci (`certificate`), parameter algoritma, dan sumber acak (`random`).

Apabila objek Cipher yang membutuhkan parameter (misalnya inisialisasi vector) diinisialisasikan untuk enkripsi, dan tidak ada parameter pada method `init`, maka implementasi cipher akan memenuhi parameter tersebut sendiri, dapat berupa menghasilkan parameter acak atau menggunakan parameter *default*.

Sementara, jika objek Cipher yang membutuhkan parameter diinisialisasi untuk dekripsi, dan tidak ada parameter pada method `init`, maka akan muncul `InvalidKeyException` atau `InvalidAlgorithmParameterException`, bergantung pada method init yang digunakan.

Proses Enkripsi dan Dekripsi Data

Data dapat dienkripsi dalam satu tahap (*single-part operation*) atau dalam multitahap (*multiple-part operation*). Operasi multitahapakan berguna jika pengembang tidak mengetahui berapa panjang dari data yang akan digunakan, atau jika data terlalu panjang untuk disimpan pada memori pada suatu waktu.

Untuk mengenkripsi atau mendekripsi suatu data dalam satu tahap, pengembang dapat menggunakan salah satu dari method `doFinal` berikut :

```
public byte[] doFinal(byte[] input);

public byte[] doFinal(byte[] input, int
inputOffset, int inputLen);

public int doFinal(byte[] input, int
inputOffset, int inputLen, byte[] output);

public int doFinal(byte[] input, int
inputOffset, int inputLen, byte[] output,
int outputOffset)
```

Sedangkan untuk mengenkripsi atau mendekripsi data dengan multitahap, pengembang dapat menggunakan salah satu method `upate` berikut :

```
public byte[] update(byte[] input);

public byte[] update(byte[] input, int
inputOffset, int inputLen);

public int update(byte[] input, int
inputOffset, int inputLen,
byte[] output);

public int update(byte[] input, int
inputOffset, int inputLen,
byte[] output, int outputOffset)
```

Operasi multitahap di atas harus diakhiri dengan salah satu metho `doFinal` di atas (jika masih tersisa beberapa data masukan untuk tahap terakhir), atau dengan memanggil method `doFinal` berikut (jika tidak ada data masukan tersisa untuk step terakhir):

```
public byte[] doFinal();

public int doFinal(byte[] output, int
outputOffset);
```

Seluruh method doFinal akan memperhatikan padding (atau unpadding) yang dibutuhkan, jika padding (atau unpadding) telah direquest sebagai bagian dari transformasi yang telah ditetapkan.

Pemanggilan `doFinal` akan mereset objek Cipher ke state dimana objek tersebut diinisialisasi dengan pemanggilan method `init`. Artinya, objek Cipher akan ter-reset dan siap untuk melakukan proses enkripsi atau dekripsi (bergantung pada mode operasi yang ditentukan pada saat pemanggilan `init`) data selanjutnya.

VI. Implementasi Pada Perangkat Android

Mengembangkan aplikasi untuk perangkat Android difasilitasi dengan sekumpulan *tools* yang disediakan pada *Software Development Kit* (SDK). Pengembang dapat mengakses *tools* tersebut melalui *plugin* Eclipse yang bernama *Android Development Tools* (ADT) atau melalu *command line*. Pengembangan dengan menggunakan Eclipse lebih dipilih karena dapat langsung memanggil *tools* yang diinginkan selama membangun aplikasi.

Implementasi Java Cryptography Architecture (JCA) yang akan dilakukan penulis pada perangkat Android ini akan menggunakan IDE Eclipse Galileo. Adapun source code implementasi JCA akan dijelaskan sebagai berikut.

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.TextView;
import android.widget.EditText;

import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class Kriptografi extends Activity
implements OnClickListener {
        TextView txt_plain;
        TextView txt_cipher;
        EditText edit_plain;
        EditText edit_cipher;
        RadioButton radio_enkripsi;
        RadioButton radio_dekripsi;
        Button button_proses;
        String passphrase =
"kriptografi";
        byte[] ba;

       /** Called when the activity is
first created. */
   @Override
   public void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        edit_plain =
(EditText)this.findViewById(R.id.edit_pla
in);
        edit_cipher =
(EditText)this.findViewById(R.id.edit_cip
her);

        radio_enkripsi =
(RadioButton)this.findViewById(R.id.radio
_enkripsi);
        radio_enkripsi.setChecked(true);
        radio_dekripsi =
(RadioButton)this.findViewById(R.id.radio
_dekripsi);

        button_proses =
(Button)this.findViewById(R.id.button_pro
ses);

button_proses.setOnClickListener(this);

   }

       public void onClick(View v) {
              if
(radio_enkripsi.isChecked()) {
```

```
                    Enkripsi();
                }
                if
(radio_dekripsi.isChecked()) {
                    Dekripsi();
                }
        }

        public  byte [] encrypt (String
passphrase, byte [] data) throws
Exception {

            byte [] dataTemp;

            try
            {
               //Fungsi hash yang digunakan
untuk meng-encode kata kunci adalah MD5
                    MessageDigest md =
MessageDigest.getInstance("MD5");

md.update(passphrase.getBytes());

                    //Membuat kunci untuk
enkripsi dengan algoritma DES
                    DESKeySpec key = new
DESKeySpec (md.digest());
                    SecretKeySpec DESKey =
new SecretKeySpec (key.getKey(), "DES");

                    //Instansiasi objek
Cipher dengan Algoritma DES, mode ECB,
dan skema padding PKCS5Padding
                    Cipher cipher =
Cipher.getInstance("DES/ECB/PKCS5Padding"
);

                    //Proses Enkripsi

cipher.init(Cipher.ENCRYPT_MODE, DESKey);
                    dataTemp =
cipher.doFinal (data);
            }
            catch (Exception e)
            {
               throw e;
            }
               return dataTemp;
        }

        public  byte [] decrypt (String
passphrase, byte [] data) throws
Exception {

            byte [] dataTemp;

            try
            {
               //Fungsi hash yang digunakan
untuk meng-encode kata kunci adalah MD5
                    MessageDigest md =
MessageDigest.getInstance("MD5");

md.update(passphrase.getBytes());

                    //Membuat kunci untuk
enkripsi dengan algoritma DES
                    DESKeySpec key = new
DESKeySpec (md.digest());
                    SecretKeySpec DESKey =
new SecretKeySpec (key.getKey(), "DES");
```

```
                    //Instansiasi objek
Cipher dengan Algoritma DES, mode ECB,
dan skema padding PKCS5Padding
                    Cipher cipher =
Cipher.getInstance("DES/ECB/PKCS5Padding"
);

                    //Proses Dekripsi

cipher.init(Cipher.DECRYPT_MODE, DESKey);
                    dataTemp =
cipher.doFinal (data);
            }
            catch (Exception e)
            {
               throw e;
            }
            return dataTemp;
    }

        protected void Enkripsi() {
                String plaintext =
edit_plain.getText().toString();
                ba = plaintext.getBytes();

                try
                {
                        //Memanggil fungsi
enkripsi
                        ba = encrypt
(passphrase, ba);
                }
                catch( Exception e)
                {

        System.out.println(e.getMessage())
;
                }
                edit_cipher.setText(new
String(ba));
        }

        protected void Dekripsi() {

                try
                {
                        //Memanggil fungsi
dekripsi
                        ba = decrypt
(passphrase, ba);
                }
                catch (Exception e)
                {

        System.out.println(e.getMessage())
;
                }
                edit_plain.setText(new
String(ba));
        }
}
```

Pada implementasi ini, kata kunci yang digunakan untuk proses enkripsi dan dekripsi sudah didefinisikan terlebih dahulu di awal yaitu “kriptografi”.

Proses enkripsi diawali dengan meng-encode kata kunci dengan fungsi hash MD5. Fungsi hash adalah prosedur deterministic yang akan mengambil blok data dengan cara yang berubah – ubah dan akan mengembalikan bit string dalam ukuran yang tetap. Pergantian sekecil apapun dalam

sumber masukan, maka akan mengubah secara drastis hasil keluarannya. Setelah itu, program akan membuat kunci dari hasil fungsi hash tadi untuk proses enkripsi dengan menggunakan algoritma DES. Instansiasi objek `DESKeySpec` akan membuat objek dari 8 bytes pertama dari data yang menjadi input parameternya. Setelah itu, kunci `DESKeySpec` akan digunakan untuk membuat objek `SecretKey` yang akan digunakan untuk mengenkripsi pesan oleh objek `Cipher`.

Proses enkripsi kemudian dilakukan dengan mebuat instanst dari kelas `Cipher`. Kelas Cipher tidak dapat langsung diinstansiasi melainkan harus memanggi method `getInstance` terlebih dahulu. Method ini memiliki 3 buah parameter, yaitu algoritma yang digunakan, mode yang digunakan, dam skema padding. Apabila pemanggilan method tidak menggunakan parameter mode dan/atau skema padding, maka standar nilai oleh provider lah yang akan digunakan. Ketika menggunakan algoritma blok cipher pada objek Cipher, maka jumlah bit yang akan diproses pada satu waktu dapat dicantumkan dengan menambahkannya pada nama mode, misalnya "AES/CFB8/NoPadding". Jika jumlah tersebut tidak dicantumkam, maka standard nilai oleh provider lah yang akan digunakan.

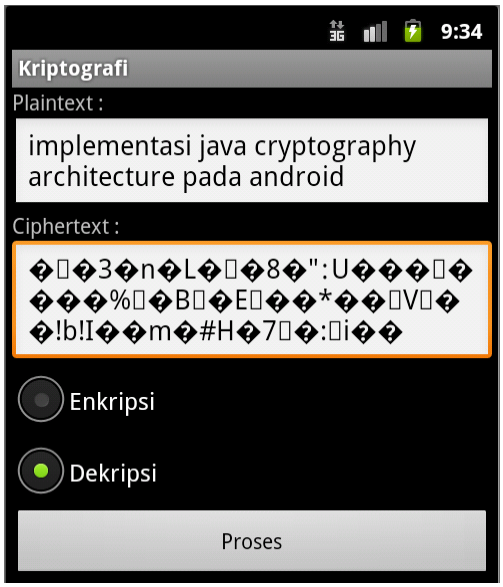
Setelah itu, barulah objek Cipher diinisialisasi dengan mode (enkripsi atau dekripsi) dan kunci yang telah dibangkitkan sebelumnya. Dan yang terakhir adalah memanggil method `doFinal`, yaitu akhir dari proses dan mengembalikan data yang telah dienkripsi berupa array of bytes.

Pada proses dekripsi, proses pembangkitan kunci tidak berbeda dengan pada proses enkripsi. Perbedaan antara proses enkripsi dengan proses dekripsi adalah pada inisialisasi objek Cipher. Pada proses enkripsi, mode yang digunakan adalah ENCRYPT_MODE, sedangkan mode yang digunakan pada proses dekripsi adalah DECRYPT_MODE.

Adapun *screenshot* dari program yang dijalankan pada emulator adalah sebagai berikut.

Gambar 5 Tampilan Program Kriptografi Pada Emulator Android

Gambar 6 Proses Enkripsi Pada Program Kriptografi



Gambar 7 Proses Dekripsi Pada Program Kriptografi

Package `javax.crypto.spec` yang tersedia menyediakan kelas dan interface yang dibutuhkan untuk menspesifikasikan kunci dan parameter untuk proses enkripsi. Adapun standard yang didukung adalah sebagai berikut :

- PKCS#3 Diffie-Hellman Key Agreement standard;
- FIPS-46-2 Data Encryption Standard (DES);
- PKCS#5 Password-Based Cryptography standard.

Kunci dapat dispesifikasikan dengan algoritma atau dengan cara yang lebih abstrak dan umum dengan ASN 1.

Parameter kunci dan algoritma ditentukan untu beberapa prosedur sebagai berikut :

- DH
- DES
- TripleDES

- PBE
- RC2
- RC5

VII. PENUTUP

Java Cryptography Architecture (JCA) merupakan bagian utama dari platform Java. JCA terdiri dari arsitektur *provider* dan sekumpulan API untuk *digital signatures, message digests (hash)*, sertifikasi dan validasi sertifikasi, enkripsi (blok simetri/asimetri), *key generation* dan manajemen, dan *secure random number generation*. API tersebut memungkinkan pengembang untuk mengintegrasikan dengan mudah keamanan ke aplikasi yang dikembangkan.

Adapun prinsi dari JCA adalah sebagai berikut :

1. *Implementation Independence*
2. *Implementation Interoperability*
3. *Algorithm Extemsibility*

Android adalah sistem operasi untuk telepon seluler yang berbasis Linux. Android menyediakan platform terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri yang digunakan oleh bermacam peranti bergerak.

Untuk melakukan proses enkripsi/dekripsi pesan, Android menggunakan JCA, yaitu dengan menggunakan *package* `javax.crypto.*`.

Standard algoritma yang didukung pada kelas `javax.crypto.spec` adalah :

- PKCS#3 Diffie-Hellman Key Agreement standard;
- FIPS-46-2 Data Encryption Standard (DES);
- PKCS#5 Password-Based Cryptography standard.

VIII. DAFTAR PUSTAKA

[1] javax.crypto | Android Developer http://developer.android.com/reference/javax/crypto/package-summary.html, diakses pada 28 Februari 2011 pukul 21.00

[2] Intoduction | Android Developer http://developer.android.com/guide/developing/index.html, diakses pada 28 Februari 2011 pukul 21.30

[3] Java Cryptography Architecture (JCA) http://download.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html#Introduction, diakses pada 28 Februari 2011 pukul 20.00

[4] Understanding Java Cryptography Architecture http://www.javascriptdownload.net/site/page.asp?dsy_id=66, diakses pada 22 Maret pukul 20.00

[5]Mulyadi. 2010. *Membuat Aplikasi Untuk Android*. Yogyakarta : Multimedia Center Publishing