Studi dan Implementasi Cryptography *API* pada Perangkat *BlackBerry*

Matthew – NIM: 13507012

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : matt_jcs@yahoo.com

Abstrak

Saat ini pengguna *BlackBerry* terus meningkat jumlahnya, terutama di pasar Indonesia. Hal ini tidak lepas karena kecanggihan teknologi koneksi internet yang disediakan *BlackBerry* itu sendiri, sehingga banyak pengguna yang puas ketika menggunakan *BlackBerry*. Tentunya koneksi terus-menerus ini harus mendapat pengamanan khusus, apalagi jika terdapat data penting yang hendak dikirim atau diterima. Sehingga di dalam pemrograman pada perangkat *BlackBerry*, telah disediakan sebuah *API* yang memfasilitasi fungsionalitas untuk kriptografi.

Pengenkripsian data menggunakan *DES* (Data Encryption Standard) sudah tidak dapat dipercaya lagi keamanannya. Oleh karena itu dibuat beberapa algoritma lain yang menggantikan *DES* dengan tingkat keamanan yang lebih tinggi. Salah satu algoritma yang masih dipakai sampai saat ini adalah *AES* (*Advanced Encryption Standard*). *AES* termasuk ke dalam kelompok algoritma kriptografi simetri berbasis *cipher block*. Tidak seperti *DES* yang berorientasi bit, *AES* sudah menggukanan orientasi byte. Dalam setiap putarannya *AES* menggunakan kunci internal yang berbeda.

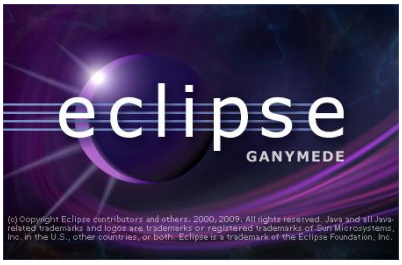
Pada makalah ini, penulis akan menganalisis dan mengimplementasikan beberapa fungsi yang ada pada Cryptography *API* pada pemrograman *BlackBerry* dengan bahasa Java. Dalam *API* ini, digunakan *AES symmetric key* untuk membuat kunci dari *encryptor engine*. Kemudian disiapkan sebuah *Digest* (*SHA-1*) untuk membuat standar data sebelum dienkripsi. Setelah itu, dapat dilakukan proses enkripsi dan mendapat ciphertext dari plaintext yang kita miliki. Jika hendak mengembalikan ke plaintext, maka dilakukan proses pendekripsian dengan metoda yang sama.

***Kata kunci**: BlackBerry, Cryptography API, Advanced Encryption Standard*

1. Pendahuluan

Sejak dua tahun terakhir ini, *BlackBerry* menjadi sangat populer di Indonesia. Hampir semua masyarakat di kalangan atas menggunakan *BlackBerry* sebagai nilai *prestidge* mereka. Demikian pula dunia pemrograman di perangkat *BlackBerry* menjadi populer, sehingga fasilitas untuk membuat program pada perangkat ini pun menjadi semakin baik.

Terdapat dua jenis platform untuk pengembangan aplikasi *BlackBerry*, yaitu berbasis bahasa Java (menggunakan *Java Mobile Edition for BlackBerry*) dan berbasis Web (menggunakan *Widgets for BlackBerry*). Pada makalah ini, penulis menggunakan aplikasi *BlackBerry* yang berbasis Java. Dalam pengembangan aplikasi ini, tentu banyak kemiripan dengan melakukan pengembangan di perangkat telepon seluler lainnya yang berbasis Java. Beberapa perbedaannya yaitu hasil keluaran dari source code yang dikompilasi berupa file berekstensi .cod, yaitu file khusus untuk diinstall pada perangkat *BlackBerry*. Sedangkan *IDE* yang digunakan adalah *eclipse GANYMEDE* dengan tambahan plugins *JDE* untuk *OS BlackBerry 4.6*.



Gambar 1 *IDE eclipse GANYMEDE* untuk membuat program Java pada *BlackBerry*

Dengan menggunakan plugins *JDE (Java Development Environtment*) ini, para *developer* dapat menggunakan banyak *API* yang disediakan. Adapun *API* yang dapat digunakan pada JDE ini berasal dari *"java", "javax", "javax.microedition", "net.rim. BlackBerry.API", "org.w3c.dom",* dan *"org.xml"*. Untuk *API* lengkap *BlackBerry* yang dapat digunakan terdapat di situs resmi *BlackBerry*, yaitu http://www.BlackBerry.com/developers/docs/4.6.0API/.

Untuk menggunakan *API* yang membantu fungsionalitas pada perangkat *BlackBerry*, *RIM* (Research of Motion) telah menyediakan sebuah *API*

bernama net.rim.device.api.crypto. Dengan *API* ini, maka fungsi-fungsi yang berhubungan dengan kriptografi seperti enkripsi dan dekripsi metode *AES* dapat langsung digunakan. Masih banyak fungsi-fungsi yang lebih mendalam di dalam *API* kriptografi ini, namun pada makalah ini penulis hanya akan membahas seputar enkripsi dan dekripsi secara *AES* dan cara mengimplementasikannya pada perangkat *BlackBerry*.

2. Standar Algoritma Kriptografi *AES*

AES atau *Advanced Encryption Standard* merupakan algoritma kriptografi yang ditemukan olehVincent Rijmen dan Joan Daemen. Standar ini disebut **Rijndael**. Berikut merupakan spesifikasi Rijndael (atau saat ini disebut *AES*):

- Mendukung panjang kunci 128 bit sampai 256 bit dengan step 32 bit.
- Panjang kunci dan ukuran blok dapat dipilih secara independen.
- Setiap blok dienkripsi dalam sejumlah putaran tertentu, sebagaimana halnya pada *DES*.
- Karena *AES* menetapkan panjang kunci adalah 128, 192, dan 256, maka dikenal *AES*-128, *AES*-192, dan *AES*-256.

Karena jumlah kunci dan ukuran blok dapat dipilih secara independen, maka jumlah putaran pun berubah-rubah sesuai panjang kunci dan ukuran bloknya. Berikut ini merupakan ketiga jenis *AES* beserta keterangannya.

Jenis *AES*	Panjang Kunci	Ukuran Blok	Jumlah Putaran
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Tabel 1 Ketiga jenis *AES* dan keterangannya

Panjang kunci, ukuran blok, dan jumlah putaran dalam satuan *word*, yaitu setara dengan 32 bit.

Secara de-fakto, hanya ada dua varian *AES*, yaitu *AES*-128 dan *AES*-256, karena akan sangat jarang pengguna menggunakan kunci yang panjangnya 192 bit.

Namun, varian *AES* yang tersederhana yaitu *AES*-128 pun sudah sangat kuat. *AES*-128 memilki panjang kunci sebesar 128-bit, maka akan terdapat:

$$2^{128} = 3{,}4 \times 10^{38} \text{ kemungkinan kunci}$$

Jika sebuat komputer tercepat dapat mencoba 1 juta kunci setiap detik, maka akan dibutuhkan waktu:

$$5{,}4 \times 10^{24} \text{ tahun untuk mencoba seluruh kunci}$$

Dengan kesimpulan sederhana seperti ini, kita sudah dapat cukup percaya dengan kekuatan *AES*, namun untuk mendapat keamanan yang lebih tinggi dapat digunakan *AES* dengan keamanan kunci 256-bit.

2.1. Cara Kerja Algoritma *AES*

Tidak seperti *DES* yang berorientasi bit, *AES* atau *Rijndael* beroperasi dalam orientasi *byte*. Pada setiap putarannya, *AES* mengunakan kunci internal yang berbeda (disebut *round key*). Sedangkan pada proses *Enciphering*, *AES* ini melibatkan operasi substitusi dan permutasi.

Garis besar Algoritma *Rijndael* yang beroperasi pada blok 128-bit dengan kunci 128-bit adalah sebagai berikut (di luar proses pembangkitan *round key*):

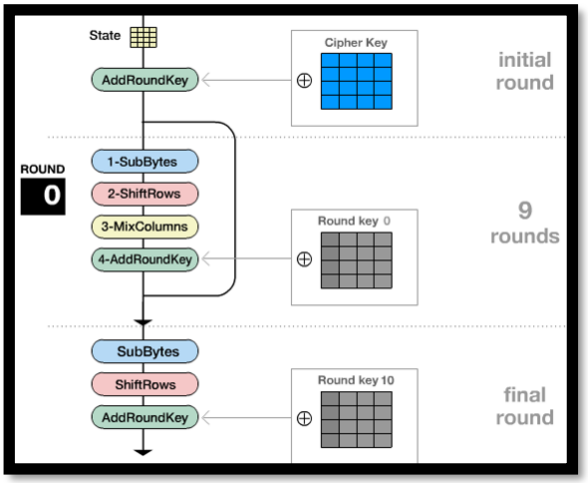
1. *AddRoundKey*: melakukan *XOR* antara *state* awal (plainteks) dengan *cipher key*. Tahap ini disebut juga *initial round.*

2. Putaran sebanyak *Nr* – 1 kali. Proses yang dilakukan pada setiap putaran adalah:

   a. *SubBytes*: substitusi *byte* dengan menggunakan tabel substitusi (*S-box*).
   b. *ShiftRows*: pergeseran baris-baris *array state* secara *wrapping*.
   c. *MixColumns*: mengacak data di masing-masing kolom *array state.*
   d. *AddRoundKey*: melakukan XOR antara *state* sekarang *round key*.

3. *Final round*: proses untuk putaran terakhir:

   a. SubBytes
   b. ShiftRows
   c. AddRoundKey



Gambar 2 Skema operasi algoritma Rijndael

Untuk memperjelas cara kerja algoritma *AES* ini, dapat memperhatikan contoh pseudo-code dalam bahasa C berikut:

```
#define LENGTH 16               /* Jumlah byte di
dalam blok atau kunci */
#define NROWS 4                 /* Jumlah baris
di dalam state */
#define NCOLS 4                 /* Jumlah kolom
di dalam state */
#define ROUNDS 10               /* Jumlah putaran
*/
typedef unsigned char byte;  /* unsigned 8-bit
integer */

rijndael (byte plaintext[LENGTH], byte
ciphertext[LENGTH],
          byte key[LENGTH])
{
 int r;                        /* pencacah
pengulangan */
 byte state[NROWS][NCOLS];  /* state sekarang
*/
 struct{byte k[NROWS][NCOLS];} rk[ROUNDS + 1];
/* kunci pada setiap putaran */

 KeyExpansion(key, rk);         /* bangkitkan
kunci setiap putaran */
 CopyPlaintextToState(state, plaintext);  /*
inisialisasi

state sekarang */
 AddRoundKey(state, rk[0]);  /* XOR key ke
dalam state */

 for (r = 1; r<= ROUNDS - 1; r++)
 {
  SubBytes(state);          /* substitusi setiap
byte dengan S-box */
  ShiftRows(state);         /* rotasikan baris i
sejauh i byte */
  MixColumns(state);        /* acak masing-
masing kolom */
  AddRoundKey(state, rk[r]);   /* XOR key ke
dalam state */
 }
 SubBytes(state);          /* substitusi setiap
byte dengan S-box */
 ShiftRows(state);         /* rotasikan baris i
sejauh i byte */
 AddRoundKey(state, rk[ROUNDS]);   /* XOR key
ke dalam state */

 CopyStateToCiphertext(ciphertext, state);  /*
blok cipherteks yang dihasilkan */
}
```

Algoritma *Rijndael* mempunyai 3 parameter sebagai berikut:

1. **`plainteks`**: *array* yang berukuran 16 *byte*, yang berisi data masukan.
2. **`cipherteks`**: *array* yang berukuran 16 *byte*, yang berisi hasil enkripsi.
3. **`key`**: *array* yang berukuran 16 *byte*, yang berisi kunci ciphering (disebut juga *cipher key*).

3. Cryptography *API* pada *BlackBerry*

API kriptografi ini dapat diakses pada JDE *BlackBerry* dengan cara melakukan import komponen dari *net.rim.device.API.crypto*, atau dengan cara

```
Import net.rim.device.api.crypto.*;
```

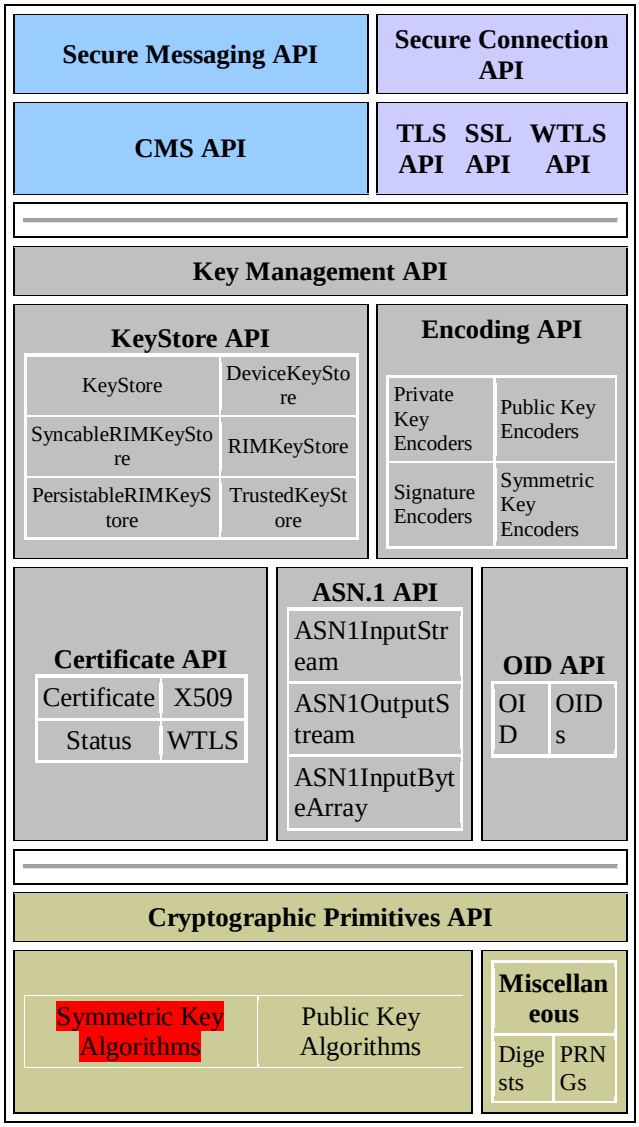
Setelah itu, maka seluruh fungsi yang dibawah *API* ini dapat diakses.

Adapun *API* kriptografi dari *RIM* ini merupakan koleksi kelas-kelas yang menyediakan keamanan yang efektif untuk aplikasi *BlackBerry* yang hendak dikembangkan, terlepas serumit apapun aplikasi yang sedang dikembangkan. *API* ini sangat dapat diandalkan dan juga fleksibel, sehingga *API* ini dapat mengerjakan setiap fungsinya dengan cara yang berbeda-beda tergantung kebutuhan sang *developer*.

Dengan *API* kriptografi ini, secara singkat memiliki kemampuan untuk mengerjakan fungsi fungsi berikut:

- Enkripsi dan dekripsi data
- Digitally sign dan verifikasi data (mengamankan integritas data)
- Autentikasi data

API kriptografi ini sangat fleksibel, sehingga dapat digunakan secara efektif pada tingkat yang berbeda sesuai kebutuhan. Berikut ini merupakan komponen-komponen dari *API* kriptografi dari RIM:

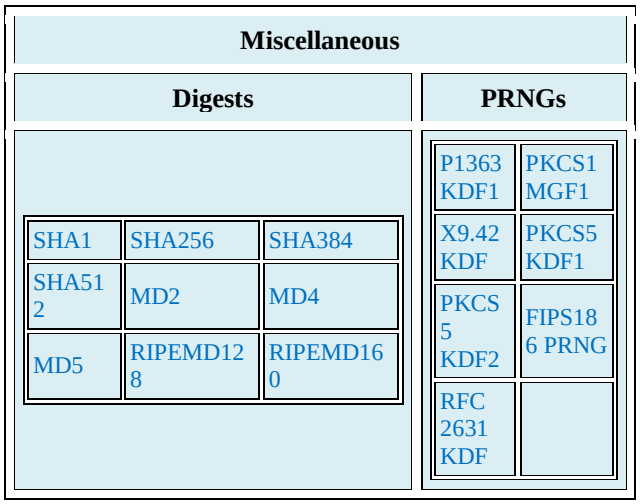


Tabel 2 *API* kriptografi dari RIM sebagai penyederhanaan dari keseluruhan sistem *API*

Secure Messaging API berisi *API* dari CMS (Cryptographic Message Syntax) dan menyediakan seluruh fungsionalitas untuk membuat aplikasi messaging yang aman. **Secure Connection *API*** berisi *API* untuk TLS (Trasnsport Layer Security), WTLS (Wireless Transport Layer Security), dan SSL (Secure Sockets Layer) dan menyediakan fungsionalitas yang dibutuhkan untuk membuat dan mengatur koneksi yang aman antara client dan server. Bersamaan dengan Secure Messaging *API*, *API* ini menyediakan fungsionalitas sebuah protokol, karena kedua *API* ini berisi kode yang dibutuhkan untuk mengimplementasikan komunikasi yang aman

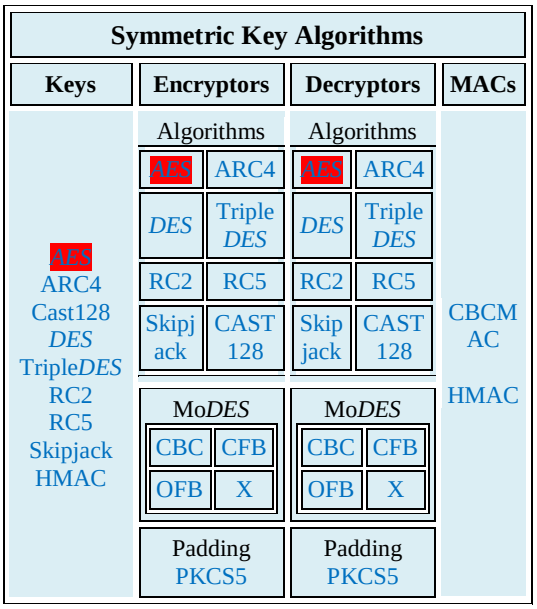
Key Management API berisi framework kriptografi dasar yang dibutuhkan utnuk membuat aplikasi yang aman. Pengaturan dan penyebaran key dipegang oleh **KeyStore API**, sedangkan key di encoding oleh **Encoding API**. **Certificate API** berisi fungsionalitas untuk mengatur sertifikasi kriptografi. **OID (Object Identifier) API** berisi fungsi yang dibutuhkan untuk mengatasi dan menggunakan beberapa Object ID yang umum. **ASN.1 (Abstract Syntax Notation) API** menyediakan mekanisme untuk memformat dan memparse data yang sering kali dibutuhkan bersamaan dengan skema dan protocol kriptografi.

Pada bagian yang terbawah, terdapat **Cryptographic Primitives API** yang berisi tools yang paling umum dibutuhkan untuk mengimplementasikan kebutuhan mengenai kriptografi. *API* ini berisi keys, MACS (Message Authentication Co*DES*), ciphers, dan fungsionalitas lainnya yang berasosiasi baik dengan kriptografi simetrik maupun kriptografi public. Sebagai tambahan, *API* ini mengandung algoritma "unkeyed" (tidak berkunci) seperti digests dan PRNGs (Pseudo Random Number Generators) yang dibutuhkan *API* lainnya.



Tabel 3 Beberapa fungsi umum kriptografi pada *API BlackBerry*

AES sendiri termasuk kedalam algoritma dengan kunci simetrik, yaitu menggunakan kunci yang sama dalam proses enkripsi maupun proses dekripsinya. Oleh karena itu, pada *API* ini *AES* termasuk ke golongan **Symmetrc Key Algorithms**.



Tabel 4 Keys, Encryptor, Decryptor, dan MAC dari algoritma Symmetric Key

3.1. Penggunaan Keys, Encryptor, dan Decryptor untuk *AES*

Pada *API BlackBerry*, keys, encryptor, dan decryptor untuk *AES* dipisahkan masing-masing menjadi satu kelas terpisah. Sehingga untuk menggunakan ketiganya, dibutuhkan ketiga kelas yang saling bergantung ini.

Berikut ini merupakan detil dari kelas *AES*EncryptorEngine, sedangkan untuk Decryptor dan Key memiliki komponen yang mirip dengan kelas enkriptornya.

Field Summary		
	`static int`	**`BLOCK_LENGTH_DEFAULT`** This is the default block length of a *AES* key

Constructor Summary	
	`AESEncryptorEngine``(AESKey key)` Creates an instance of the `AESEncryptorEngine` class given the *AES* key with a default block length of 16 bytes.
	`AESEncryptorEngine``(AESKey key, int blockLength)` Creates an instance of the `AESEncryptorEngine` class given the *AES* key.
	`AESEncryptorEngine``(AESKey key, int blockLength, boolean inECMMode)` Creates an instance of the `AESEncryptorEngine` class given the *AES* key.

Method Summary		
	`void`	**`encrypt`**`(byte[] plaintext, int plaintextOffset, byte[] ciphertext, int ciphertextOffset)` Encrypts the data given a byte array containing the plaintext, a byte array to hold the ciphertext, and their associated offsets.
	`String`	**`getAlgorithm`**`()` Returns the name of the algorithm with with key and block lengths, ie "*AES*_128_128" (`"AES_" + keyBitLength + "_" + blockBitLength`).
	`int`	**`getBlockLength`**`()` Returns the block length of the encrypted data.
	`int`	**`getKeyLength`**`()` Returns the length of the key in bytes.

Tabel 5 Summary dari kelas *AES*EncryptorEngine

Ketiga kelas *AES* ini merupakan kelas yang terkategori *Signed*. Artinya element yang terkategori *Signed* hanya dapat diakses pada perangkat *BlackBerry* jika menggunakan aplikasi(*IDE*) yang telah diverifikasi (signed) saat melakukan proses pengkompilasian.

Untuk mendapat seperangkat code signing keys, dapat melakukan pemesanan ke http://www.*BlackBerry*.com/go/co*DES*igning seharga $20. Code signing ini hanya dibutuhkan jika aplikasi berjalan pada perangkat *BlackBerry* yang sebenarnya, namun pada pengembangan di simulator *code signing keys* ini tidak dibutuhkan.

4. Implementasi enkripsi dan dekripsi data dengan metode *AES* pada *BlackBerry*

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 **KriptoBB**.

File: MattController.java

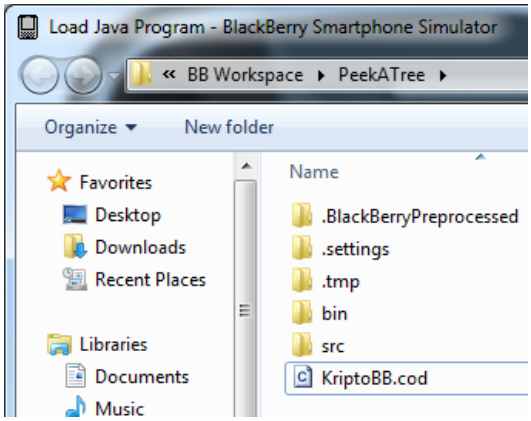
```
public byte[] encrypt( byte[] keyData, byte[]
data ) throws CryptoException, IOException {
// membuat kunci AES
AESKey key = new AESKey( keyData );

// menyiapkan engine untuk enkripsi
AESEncryptorEngine engine = new
AESEncryptorEngine( key );

// data dibuat sesuai ukuran block
PKCS5FormatterEngine fengine = new
PKCS5FormatterEngine( engine );

// menyiapkan block yang dienkripsi
ByteArrayOutputStream output = new
ByteArrayOutputStream();
BlockEncryptor encryptor = new BlockEncryptor(
fengine, output );

// kalkulasi fungsi hash dari data (diperoleh
dari SHA1)
SHA1Digest digest = new SHA1Digest();
digest.update( data );
byte[] hash = digest.getDigest();

//proses enkripsi
encryptor.write( data );
encryptor.write( hash );
encryptor.close();
output.close();

// return dalam byte
return output.toByteArray();
}


public byte[] decrypt( byte[] keyData, byte[]
ciphertext ) throws CryptoException,
IOException {
// membuat kunci AES
AESKey key = new AESKey( keyData );

// menyiapkan engine untuk dekripsi
AESDecryptorEngine engine = new
AESDecryptorEngine( key );

// data dibuat sesuai ukuran block
PKCS5UnformatterEngine uengine = new
PKCS5UnformatterEngine( engine );

// menyiapkan block yang didekripsi
ByteArrayInputStream input = new
ByteArrayInputStream( ciphertext );
BlockDecryptor decryptor = new BlockDecryptor(
uengine, input );

//membaca data untuk didekripsi
byte[] temp = new byte[ 100 ];
DataBuffer buffer = new DataBuffer();

for( ;; ) {
        int bytesRead = decryptor.read( temp
);
        buffer.write( temp, 0, bytesRead );
        if( bytesRead < 100 ) {
                //data sudah habis
                break;
        }
}

byte[] plaintextAndHash = buffer.getArray();

int plaintextLength = plaintextAndHash.length
- SHA1Digest.DIGEST_LENGTH;

byte[] plaintext = new byte[ plaintextLength
];
byte[] hash = new byte[
SHA1Digest.DIGEST_LENGTH ];

System.arraycopy( plaintextAndHash, 0,
plaintext, 0, plaintextLength );
System.arraycopy( plaintextAndHash,
plaintextLength, hash, 0,
SHA1Digest.DIGEST_LENGTH );

// mencocokan fungsi hash hasil dengan yang
diperoleh dari plaintext
SHA1Digest digest = new SHA1Digest();
digest.update( plaintext );
byte[] hash2 = digest.getDigest();

if( !Arrays.equals( hash, hash2 )) {
        // TODO - Throw an exception?
        throw new RuntimeException();
}

// return dalam bite
return plaintext;
}
```

File: MattScreen.java

```
public class MattScreen extends MainScreen {

// This screen's controller
MattController controller;

// Components declaration
Background editBg;
HorizontalFieldManager hfm1;
LabelField ptLabel;
PATEditField ptEdit;
PATButtonField EncryptBtn;
HorizontalFieldManager hfm2;
LabelField cpLabel;
PATEditField cpEdit;
PATButtonField DecryptBtn;
HorizontalFieldManager hfm3;
LabelField oriLabel;
PATEditField oriEdit;
byte[] keyData = RandomSource.getBytes( 256 );
byte[] cipherData;

/**
* Constructor
*/
public MattScreen(MattController mattctr) {
this.controller = mattctr;

// Components initialization
hfm1 = new
HorizontalFieldManager(HorizontalFieldManager.
FIELD_LEFT);
ptLabel =  new LabelField("Plaintext: ");
```

```
ptEdit = new PATEditField();
EncryptBtn = new PATButtonField("Encrypt!") {
protected boolean navigationClick(int status,
int time) {
byte data[] = ptEdit.getText().getBytes();
try {
cipherData = controller.encrypt(keyData,
data);
} catch (CryptoException e) {
// catch block
e.printStackTrace();
} catch (IOException e) {
// catch block
e.printStackTrace();
}
String tempCipher = new String(cipherData);
cpEdit.setText(tempCipher);
return super.navigationClick(status, time);
}
};
hfm1.add(ptLabel);
hfm1.add(ptEdit);
hfm1.add(EncryptBtn);

hfm2 = new
HorizontalFieldManager(HorizontalFieldManager.
FIELD_LEFT);
cpLabel =  new LabelField("Ciphertext: ");
cpEdit = new PATEditField();
DecryptBtn = new PATButtonField("Decrypt!") {
protected boolean navigationClick(int status,
int time) {
byte tempData[] = null;
try {
tempData = controller.decrypt(keyData,
cipherData);
} catch (CryptoException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
String tempOri = new String(tempData);
oriEdit.setText(tempOri);
return super.navigationClick(status, time);
}
};
hfm2.add(cpLabel);
hfm2.add(cpEdit);
hfm2.add(DecryptBtn);

hfm3 = new
HorizontalFieldManager(HorizontalFieldManager.
FIELD_LEFT);
oriLabel =  new LabelField("Decrypted Text:
");
oriEdit = new PATEditField();
hfm3.add(oriLabel);
hfm3.add(oriEdit);

// Adding of components to this screen
add(hfm1);
add(hfm2);
add(hfm3);

// Setting screen
getMainManager().setBackground(BackgroundFacto
ry.createSolidBackground(0x00FDF9DE));
}
}
```

Setelah selesai dengan membuat source code, maka selanjutnya adalah melakukan kompilasi, kemudian mendownload hasil kompilasi (file .cod) ke dalam perangkat *BlackBerry*.



Gambar 3 Cara download KriptoBB.cod

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

Gambar 4 Aplikasi KriptoBB pada Home sebuah perangkat *BlackBerry*

Setelah itu, dapat juga diperiksa detail aplikasi yang dibuat melalui:
Options > Advanced Options > Applications > View Properties
Sedangkan untuk membuka aplikasi, cukup menekan tombol enter atau trackball pada aplikasi KriptoBB ini.

Gambar 5 Detil dari aplikasi KriptoBB

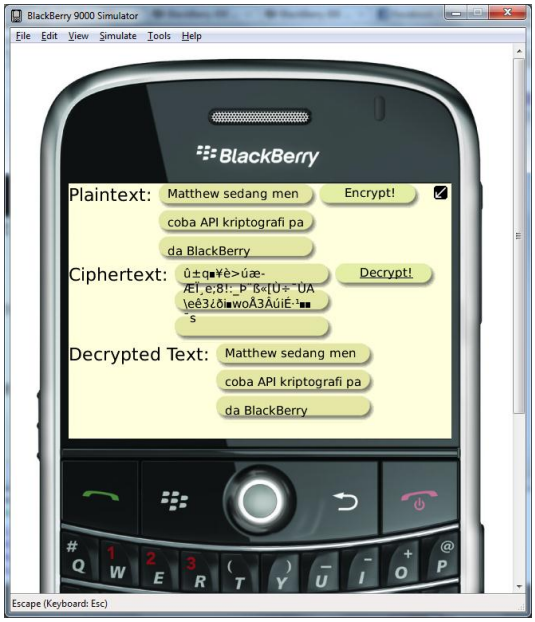
Berikut ini adalah langkah-langkah dari awal membuka aplikasi KriptoBB hingga selesai mendekripsi dan mendapat pesan awal yang dienkripsi

Gambar 6 Tampilan awal KriptoBB

Aplikasi ini akan menerima input Plaintext (berupa string agar mudah dicoba), yang kemudian di enkripsi. Untuk melakukan enkripsi cukup menekan tombol Encrypt!, sedangkan kunci sudah otomatis dibangkitkan pada source code secara random.

Gambar 7 Plaintext dimasukan

Gambar 8 Ciphertext setelah tombol Encrypt! ditekan



Gambar 9 Hasil text yang didekripsi kembali, mendapatkan plaintext yang awal

Setelah didekripsi kembali, maka plaintext awal diperoleh kembali. Pada aplikasi ini dibuat hanya mengatasi file berupa text untuk mempermudah tampilan, namun fungsi enkripsi dan dekripsi sudah mendukung setiap jenis file karena proses enkripsi-dekripsi dilakukan pada level *array of byte.*

KESIMPULAN

Seiring dengan kemajuan pasar penjualan *BlackBerry* saat ini, pada lingkungan pemrograman *BlackBerry* pun terus mengalami kemajuan. Untuk itu sangatlah diperlukan teknik kriptografi untuk menjaga setiap unsur yang bersifat *confidential* pada *BlackBerry*, baik pesan standar, data, maupun koneksi.

Oleh karena itu *RIM* sebagai pihak pengembang API pada pemrograman aplikasi BlackBery mengembangkan API kriptografi untuk memudahkan *developer* dalam menjaga keamanan data pada aplikasi yang dibangunnya.

Salah satu API sederhana yang disediakan RIM adalah *Advanced Encryption Standard* (*AES*). AES ini merupakan salah satu solusi yang baik untuk mengatasi masalah keamanan dan kerahasiaan data yang pada umumnya diterapkan dalam pengiriman dan penyimpanan data melalui media elektronik. Sehingga untuk skala perangkat *BlackBerry*, keamanan data yang dienkripsi dengan *AES* sudah sangat terjamin.

Dengan aplikasi berbasis Java, penulis berhasil mengimplementasikan sebuah aplikasi untuk perangkat *BlackBerry* bernama "**KriptoBB**". Aplikasi ini menggunakan *API* kriptografi yang disediakan *RIM*, dan ditampilkan bersama dengan *GUI* secukupnya. Dengan aplikasi ini kita dapat mengenkripsi suatu text dengan metode AES dan mendekripsi kembali ciphertext menjadi plaintext pada awalnya.

DAFTAR PUSTAKA

[1]. *BlackBerry JDE API Reference*, < http://www.*BlackBerry*.com/*developer*s/docs/4.6.0api/>, diakses pada 23Maret 2009

[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 2 Maret 2010.

[4]. *BlackBerry Support Community Forum*, <http://supportforums.*BlackBerry*.com/t5/*BlackBerry*-Internet-Service/Encryption-on-*BlackBerry*-BIS/m-p/262014>, diakses pada 3 Maret 2010.

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