Bab 7
Security

7.1 Tujuan

Pada akhir pembahasan bab ini, siswa diharapkan mampu :

- Memahami dasar security dan kriptografi
- Memahami proteksi domains dan permissions
- Bagaimana menambahkan permissions pada MIDlet Suite
- Bagaimana membuat MIDlet Suite menggunakan NetBeans Mobility Pack
- Bagaimana membuat message digest menggunakan SATSA
- Bagaimana melakukan enkripsi menggunakan symmetric keys

7.2 Dasar Security

Kriptografi

Kriptografi adalah cabang dari ilmu matematika yang memiliki banyak fungsi dalam pengamanan data. Kriptografi adalah proses mengambil message dan menggunakan beberapa fungsi untuk menggenerasi materi kriptografis (sebuah digest atau message terenkripsi).

Kriptografi adalah salah satu dari teknologi yang digunakan dalam layanan security seperti integrity, confidentiality, identity dan non repudiation.

Tipe Security Services

Sebelum kita memasuki bahasan dasar fungsi kriptografi, kita pelajari terlebih dahulu beberapa security services penting yang digunakan dalam sebuah aplikasi :

Authentification – Adalah proses verifikasi indentitas dari pengguna pada akhir jalur komunikasi.

Confidentiality – Jika kita mengirimkan data sensitive melalui sebuah jaringan, kita ingin memastikan bahwa hanya penerima yang dituju yang dapat membacanya.

Integrity – Kita ingin memastikan bahwa data yang kita terima tidak mengalami pengubahan, penambahan ataupun pemisahan.

Non-repudiation – Service ini dapat menunjukkan bukti bahwa pengirim telah mengirimkan message, atau penerima telah menerima message.

Authorization – Untuk memastikan bahwa user memiliki hak akses spesifik terhadap data penting maupun sumber data.

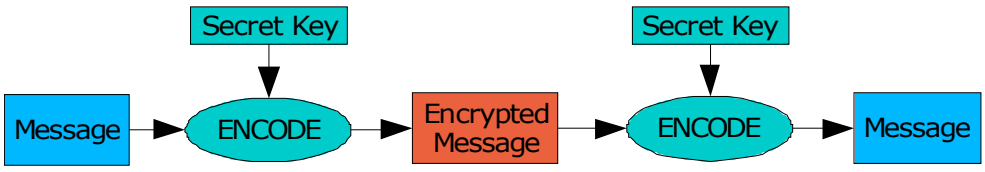
Message Digests

Sebuah message digest juga disebut sebagai digital fingerprint. Sebuah message digest adalah sebuah kesimpulan matematis dari sebuah message atau file. Hal ini untuk memastikan integritas dari message atau file. Sehingga dapat memberikan informasi bahwa sebuah message telah mengalami perubahan atau tidak. Mengubah satu karakter dari sebuah file atau message dapat menyebabkan perubahan drastis dari message digest. Digest terbuat melalui sebuah proses yang sangat menyulitkan untuk membuat dua file atau message yang berbeda dengan message digest yang sama.

Sebuah message digest berfungsi dalam satu alur fungsi. Sebuah message digest relatif mudah untuk diproses, namun sangat sulit jika dilakukan dengan cara sebaliknya. Dari sebuah message digest, sangat sulit untuk mengolah dan membuat sebuah message yang dapat menghasilkan message digest yang sama.

Kriptografi Symmetric Key

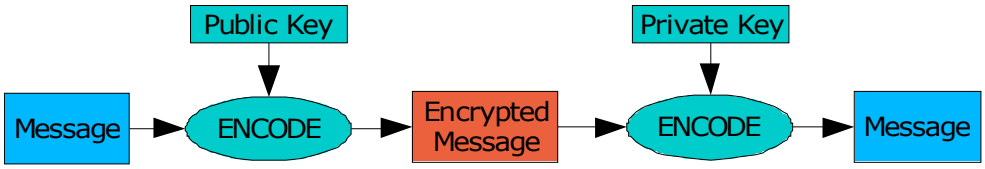
Dengan kriptografi symmetric key, sebuah message dapat terenkripsi dan terdekripsi menggunakan key yang sama. Baik pengirim maupun penerima message harus memiliki key yang sama supaya proses tersebut berjalan dengan sukses. Pengirim menggunakan key rahasia untuk mengenkripsi message, sedangkan penerima menggunakan kunci yang sama untuk mendekripsi message. Sekali message telah terenkripsi, message tersebut dapat dikirimkan melalui jaringan tanpa dipahami oleh penyadap.



Kriptografi Asymmetric Key

Permasalahan dari symmetric key adalah kedua pihak harus memiliki key yang sama. Key tersebut harus dikirimkan secara aman menuju penerima dari beberapa sebab sehingga key dapat dicuri dan digunakan untuk mendekripsi sebuah message.

Dengan menggunakan asymmetric keys, pengirim mengenkripsi message dengan menggunakan public key penerima. Kemudian penerima mendekripsi message tersebut menggunakan private key. Private key hanya dimiliki oleh penerima. Antara private dan public key merupakan komplemen matematis sehingga message yang terenkripsi menggunakan public key dapat terdekripsi menggunakan private key. Hal tersebut secara komputasi juga sulit untuk membuat private key ulang menggunakan public key.



Kriptografi Public Key

Algoritma public key menuntut penggunaan complementary key secara terpisah dalam proses enkripsi dan dekripsi. Hal ini menunjukkan kepastian bahwa akan memakan waktu yang sangat lama untuk mengetahui private key melalui pengolahan public key. Tuntutan ini membuat distribusi public key menjadi mudah tanpa mengkhawatirkan kerahasiaan dari private key.

Algoritma public key yang amat popular adalah algortima RSA. Keamanan dari RSA telihat dari tingkat kesulitan faktorial numerik dalam cakupan yang besar.

Digital Signature

Sebuah digital signature mirip dengan message digest kecuali bahwa digest dihasilkan oleh private key dari sebagian personal atau entitas. Public key digunakan dalam verifikasi bahwa message yang ditandai berasal dari penanda.

Key Management

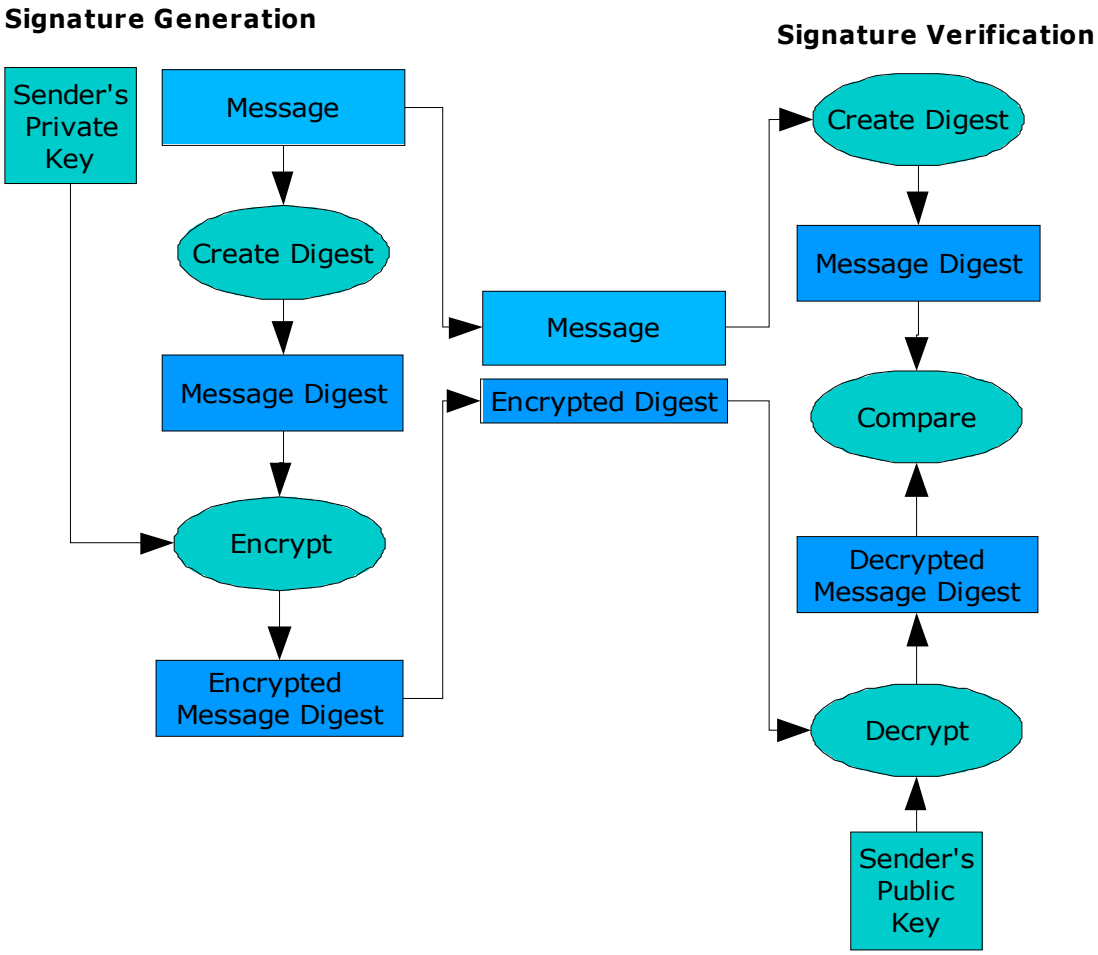
Salah satu permasalahan dari kriptografi public key adalah key management. Bagaimana anda mengetahui bahwa public key yang anda gunakan dalam verifikasi otentifikasi dari digital signature adalah public key asli yang dikirimkan oleh pengirim?

Digital Certificates adalah messages yang dibuat oleh Certification Authority (CA) yang menyertakan keabsahan entitas dari public key. Pada dasarnya, Digital Certificates adalah container dari Public Keys.

Untuk mendapatkan sertifikat dari CA, sebuah entitas menyertakan dokumentasi yang membuktikan eksistensi dari identitas. Setelah melalui proses verifikasi identifikasi, CA kemudian menandai public key dari identitas yang menggunakan private key dari CA.

Namun kita telah menciptakan problem yang lain. Bagaimana kita memverifikasi bahwa public key dari CA adalah asli? Anda akan mengetahui bahwa kita hanya membuat rangkaian dari keabsahan.

Solusinya adalah penandaan certificate secara pribadi. CA menandai public key menggunakan private key yang sesuai. Kemudian certificate yang telah ditandai dan mengandung public key dari CA akan didistribusikan secara bebas. Hal ini dikenal sebagai root certificate. Certificate yang ditandai secara pribadi dapat dibuat dengan mudah oleh siapapun. Aplikasi seperti web browser dan email umumnya disertai dengan root certificates dari Certificate Authorities yang diterima secara luas.



7.3 J2ME Security

Protection Domains – Sebuah protection domains mendefinisikan rangkaian permissions yang disertakan pada MIDlet Suite. MIDP 2.0 menjelaskan bahwa paling tidak terdapat dua buah protection domains : untrusted dan trusted domains. Untrusted domains adalah pembatasan dimana akses terhadap protected API pada kondisi default tidak diijinkan. Seorang user secara eksplisit harus mengatur tipe akses MIDlet Suite terhadap API. Untrusted MIDlets (berjalan di untrusted domains) tidak memerlukan user permission dalam mengakses protected API.

Untrusted dan trusted domain menyediakan akses yang tak terbatas pada Record Management, MIDlet life cycle, LDCUI, Game dan Multimedia API. Bagaimanapun, API untuk koneksi HTTP dan HTTPS menuntut kejelasan permissions dari user jika MIDlet Suite berjalan pada untrusted domain.

Sebuah protection domain adalah rangkaian dari ”Allowed” dan ”User” permissions yang diberikan kepada MIDlet Suite.

Permissions

Terdapat dua tipe mode interaksi permissions, mode Allowed dan User. Pada mode Allowed, user tidak diminta melakukan pengaturan permission saat MIDlet mengakses sebuah API yang terproteksi. Sebuah aplikasi secara otomatis memberikan hak akses terhadap resource dan interaksi dari user tidak diperlukan.

Dalam mode User, device menanyakan apakah user menginginkan untuk mencabut atau memberikan hak akses MIDlet terhadap resource. Frekuensi dari pertanyaan bergantung pada mode interaksi yang dipilih oleh user.

Mode Interaksi User

Sebuah user permission memiliki salah satu dari 3 mode interaksi berikut :

Blanket – User memberikan permission pada MIDlet Suite untuk mengakses resource atau API secara permanen. User tidak akan lagi diminta melakukan pengaturan setiap MIDlet Suite berjalan. Permission yang ada akan tetap eksis hingga MIDlet Suite dihapus dari device atau user merubah permission tersebut.

Membuat sebuah permission adalah salah satu dari cara pengamanan akses terhadap restricted APIs. Dalam MIDP, nama dari permission menggunakan nama dari package dari API tersebut sebagai prefix dan bersifat case sensitive. Jika permission tersebut ditujukan kepada sebuah class, maka penamaan permission harus mengandung nama class dan package.

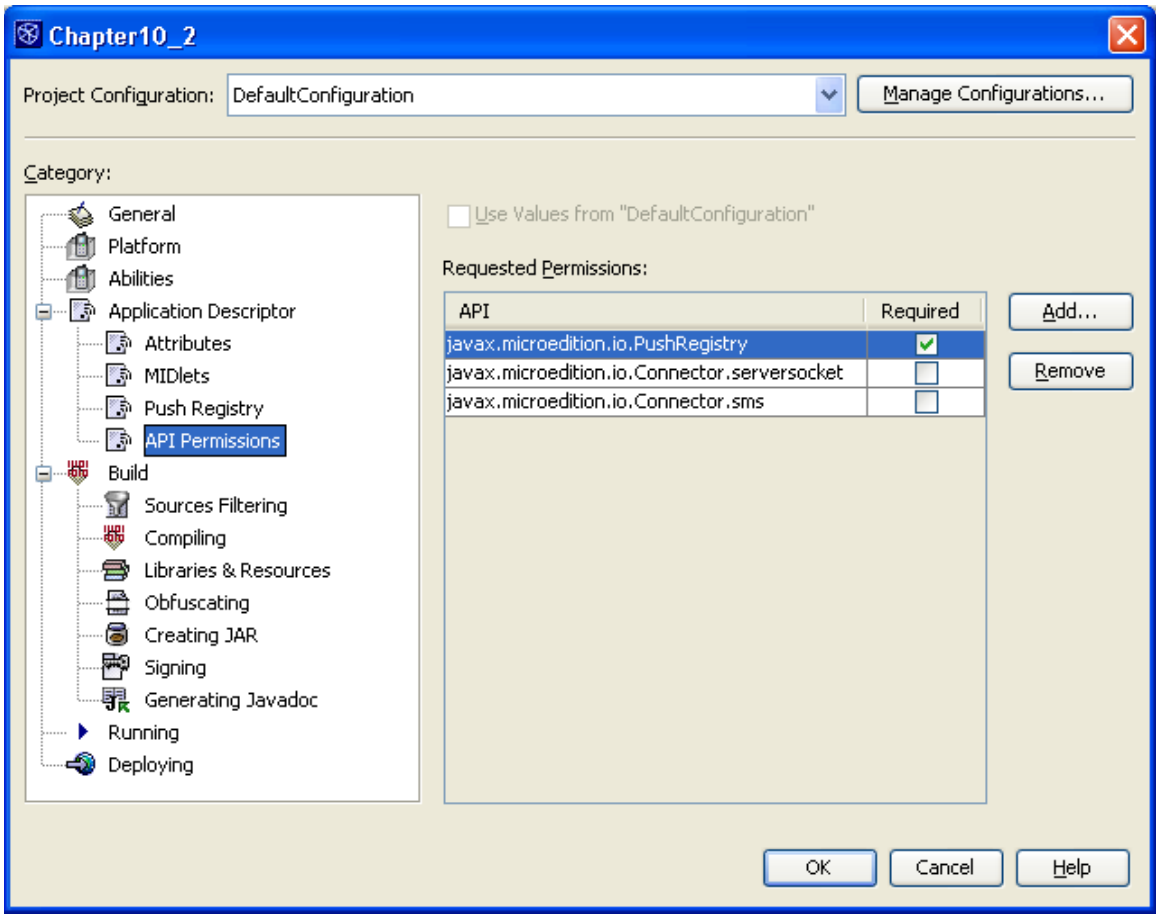
Sebuah MIDlet dapat menuntut adanya permission dengan mendeklarasikan MIDlet-Permissions ataupun atribut MIDlet-Permissions-Opt pada application descriptor. Jika MIDlet Suite menyertakan atribut MIDlet-Permissions, atribut permission tersebut

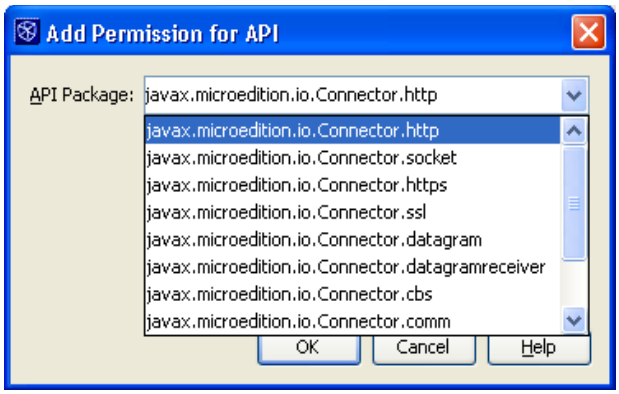
harus diberi hak akses terhadap protection domain. Jika hak akses tidak diberikan, maka proses instalasi akan dibatalkan.

Multiple permissions dituliskan menggunakan tanda koma (,) sebagai pemisah.

```
MIDlet-Permissions: javax.microedition.io.Connector.http

MIDlet-Permissions-Opt: javax.wireless.messaging.sms.receive,
 javax.wireless.messaging.sms.send
```

Membuat permissions pada MIDlet Suite menggunakan NetBeans Mobility Pack :

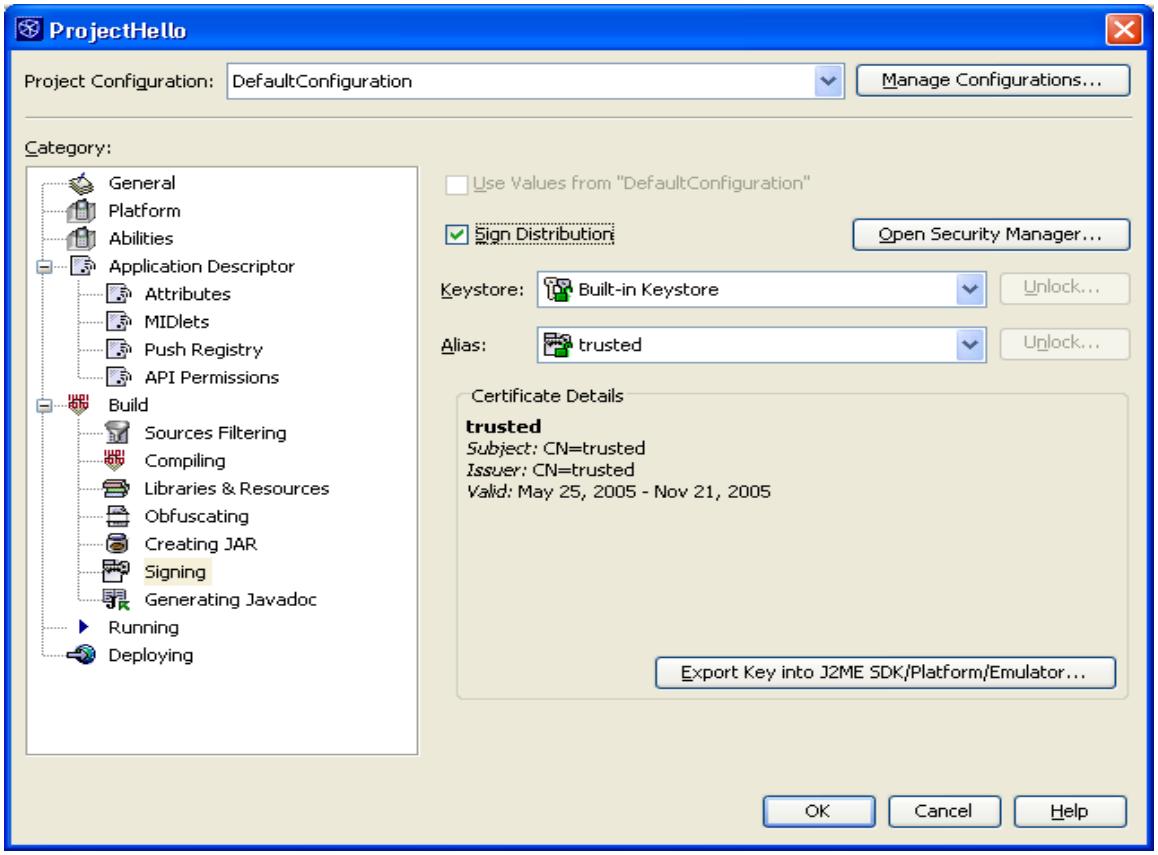




Trusted MIDlets – Sebuah MIDlet dapat diputuskan sebagai trusted application jika authentifikasi dan integritas dari file JAR dapat terverifikasi oleh device dan terbatas pada sebuah protection domain. Proses verifikasi dilakukan oleh device menggunakan certificates.

Menandai MIDlet pada NetBeans Mobility Pack :

Untuk memberi tanda pada MIDlet Suite menggunakan NetBeans Mobility Pack, buka project properties (klik kanan project name pada projects tab dan pilih Properties). Periksa bagian ”Sign Distribution” :



7.4 Menggunakan Security dan Trust Services API (SATSA)

Security and Trust Services API (SATSA) terdefinisi dalam Java Spesification Request (JSR) 177. SATSA adalah sebuah pilihan package yang menyediakan APIs untuk fungsi – fungsi security seperti manajemen digital signatures, pembuatan message digest dan digital signatures, berhubungan dengan Java Cards dan operasi kriptografi lainnya.

Contoh berikut ini menunjukkan cara pembuatan message digest dan enkripsi sebuah message menggunakan symmetric keys :

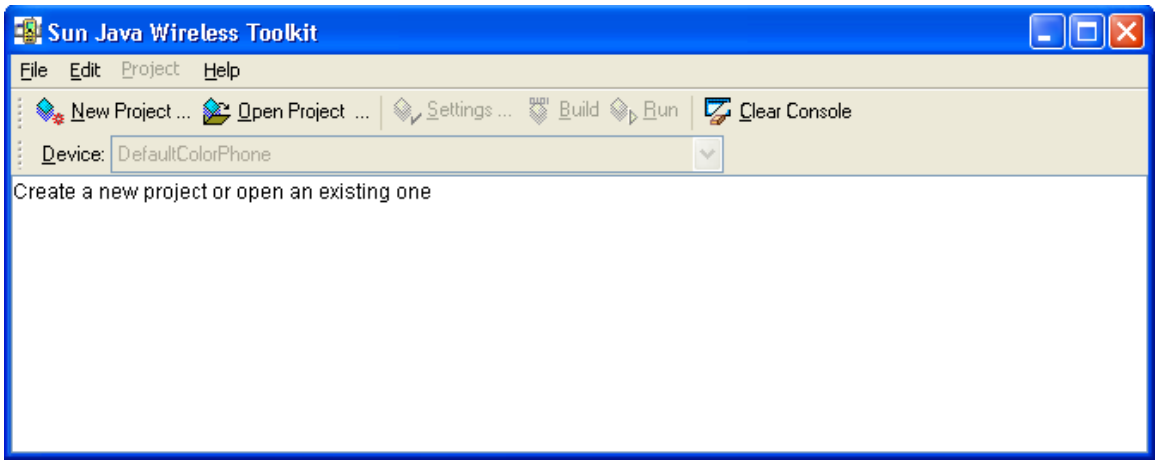
Membuat Message Digest :

```
/*
* DigestMidlet.java
*
*/

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.security.*;

public class DigestMidlet extends MIDlet {
	public void startApp() {
		String message = "I LOVE JENI!";
		System.out.println("Generating digest for message: " +
		message);

		byte[] digest = generateDigest(message.getBytes());

		System.out.println("SHA-1 Digest:");
		for (int i=0; i<digest.length; i++) {
			System.out.print(digest[i] + " ");
		}
		System.out.println();
	}
	public void pauseApp() {
	}
	public void destroyApp(boolean unconditional) {
	}
	public byte[] generateDigest(byte[] message) {
		String algorithm = "SHA-1";
		int digestLength = 20;
		byte[] digest = new byte[digestLength];
		try {
			MessageDigest md;
			md = MessageDigest.getInstance(algorithm);
			md.update(message, 0, message.length);
			md.digest(digest, 0, digestLength);
		} catch (Exception e) {
			System.out.println("Exception: " + e.getMessage());
		}
		return digest;
	}
}
```

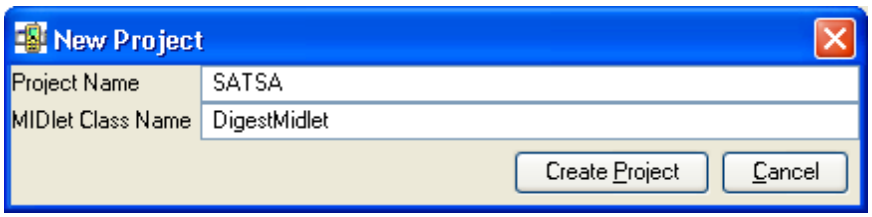
Sun Java Wireless Toolkit 2.3 menyediakan dukungan JSR 177 (atau SATSA) :

Proses Build dan Run file DigestMidlet :

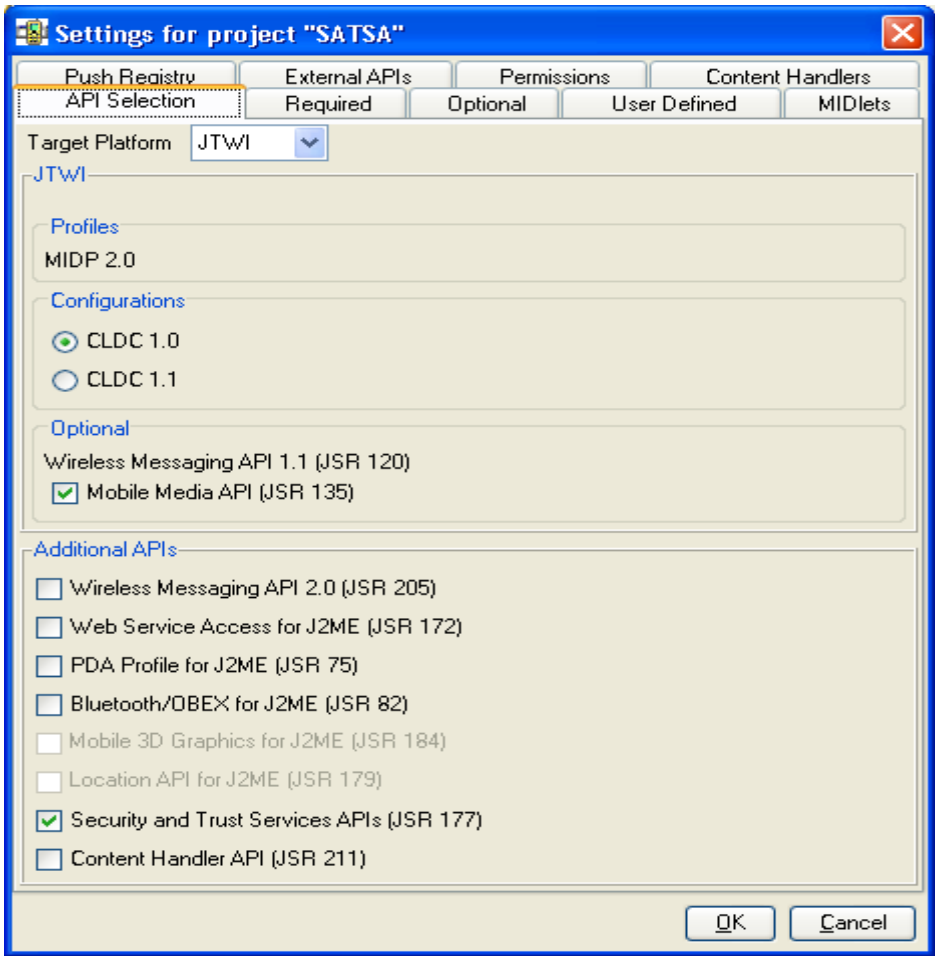
1. Buka aplikasi Ktoolbar dari Wireless Toolkit :

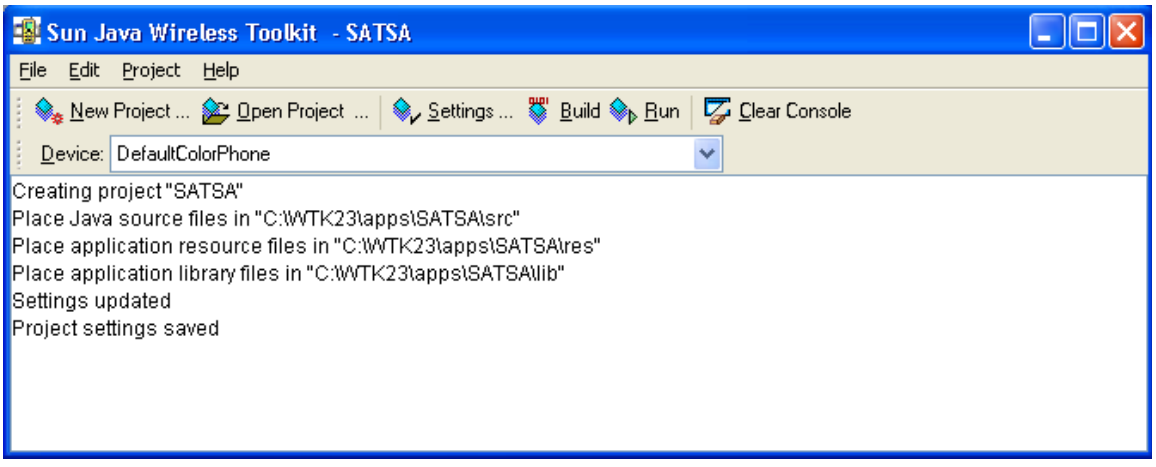


2. Pilih "New Project" kemudian tentukan nama project dan class MIDlet :



3. Pilih "JWTI" sebagai Target Platform. Tandai pilihan "Security and Trust Services APIs (JSR 177) dan kilik "OK".



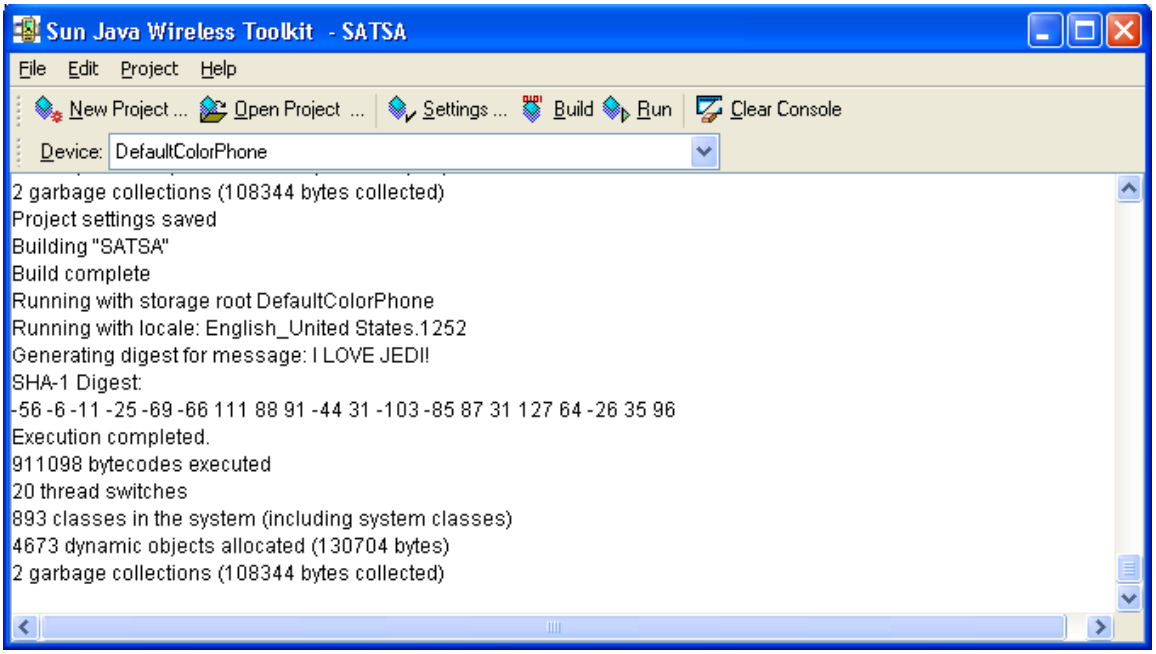


4. Buat file dengan nama *DigestMidlet.java* pada direktori : WTK/apps/SATSA/src

(WTK adalah direktori instalasi dari Sun Java Wireless Toolkit, secara default adalah C:\WTK23, dan SATSA adalah nama project).

5. Klik ”Build”

6. Klik ”Run” untuk menjalankan MIDlet pada emulator. Jika anda menjalankan MIDlet pada emulator, anda tidak akan mendapatkan output grafis apapun. Output yang akan dihasilkan adalah berupa console pada WTK Tollbar.

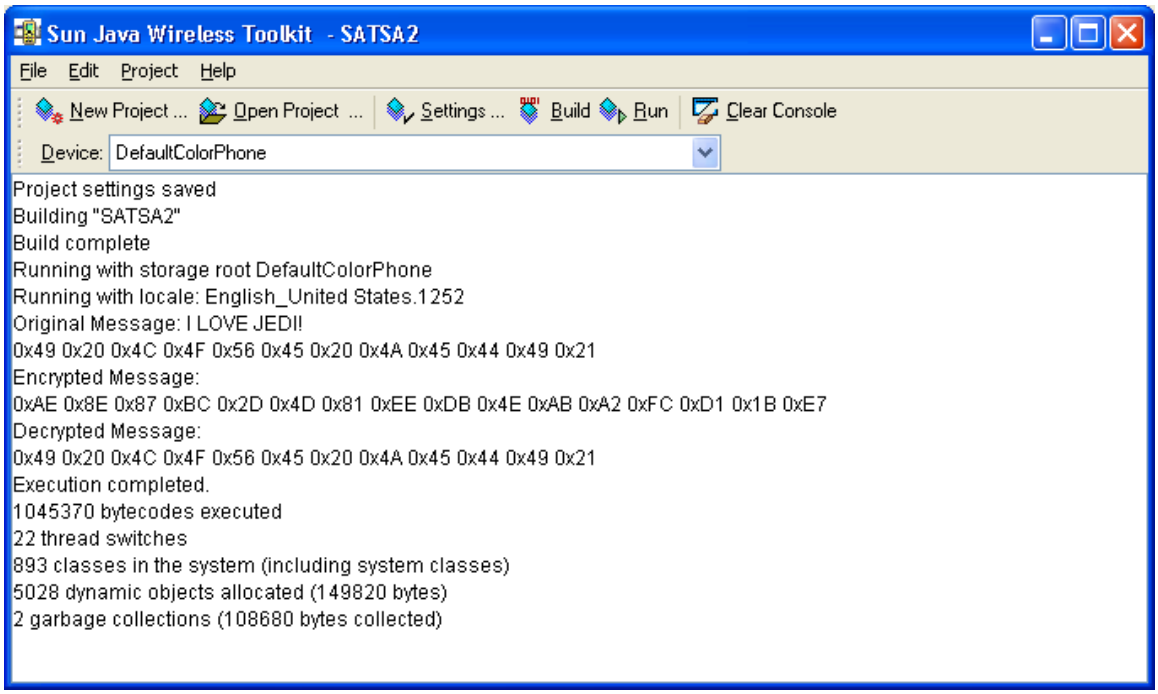


Enkripsi dan Dekripsi Messages menggunakan symmetric keys :

```
/*
* SymmetricCipherMidlet.java
*
*/

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.microedition.securityservice.*;


public class SymmetricCipherMidlet extends MIDlet {
       private static final byte[] key = {
              (byte) 0xab, (byte) 0xcd, (byte) 0xef, (byte) 0x88,
              (byte) 0x12, (byte) 0x34, (byte) 0x56, (byte) 0x78
```

```
        };
        String message = "I LOVE JEDI!";

        public SymmetricCipherMidlet() {
        try {
               System.out.println("Original Message: " + message);
               printBytes(message.getBytes());
               byte[] encryptedMessage =
               encrypt("DES/ECB/PKCS5Padding",
               message.getBytes(), key, "DES");
               System.out.println("Encrypted Message:");
               printBytes(encryptedMessage);
               byte[] decryptedMessage =
               decrypt("DES/ECB/PKCS5Padding",
               encryptedMessage, key, "DES");
               System.out.println("Decrypted Message:");
               printBytes(decryptedMessage);
        } catch (Exception e) {
               System.out.println("Exception: " + e.getMessage());
               e.printStackTrace();
        }
}
private void printBytes(byte[] bytes) {
        for (int i = 0; i<bytes.length; i++){
               System.out.print(toHex(bytes[i]) + " ");
        }
        System.out.println();
}
private String toHex(byte b) {
        char d1 = toHexDigit((byte) ((b>>4) & 0x0F));
        char d2 = toHexDigit((byte) (b & 0x0F));
        return ("0x"+d1+d2);
}
private char toHexDigit(byte x) {
        if (x > 9) {
               return ((char) ('A' + (x-10)));
        } else {
               return ((char) ('0' + x));
        }
}
private byte[] encrypt(String algorithm,
        byte[] message, byte[] keybytes, String keyAlgo)
        throws NoSuchAlgorithmException, NoSuchPaddingException,
        InvalidKeyException, ShortBufferException,
        IllegalBlockSizeException, BadPaddingException {
               Cipher cipher = Cipher.getInstance(algorithm);
               Key key = new SecretKeySpec(keybytes, 0, keybytes.length,
               keyAlgo);
               int blockSize = 16;
               int cipherLength =
                      ((message.length / blockSize)
                      + ((message.length % blockSize) > 0 ? 1 : 0))
                      * blockSize;
               cipher.init(Cipher.ENCRYPT_MODE, key);
               byte[] cipherText = new byte[cipherLength];
               cipher.doFinal(message, 0, message.length, cipherText, 0);
               return(cipherText);
        }
public byte[] decrypt(String algorithm,
        byte[] cipherText, byte[] keybytes, String keyAlgo)
        throws NoSuchAlgorithmException, NoSuchPaddingException,
        InvalidKeyException, ShortBufferException,
        IllegalBlockSizeException, BadPaddingException {
```

```
                    Cipher cipher = Cipher.getInstance(algorithm);
                    Key key = new SecretKeySpec(keybytes, 0, keybytes.length,
                    keyAlgo);
                    cipher.init(Cipher.DECRYPT_MODE, key);
                    byte[] decrypted = new byte[message.length()];
                    cipher.doFinal(cipherText, 0, cipherText.length, decrypted,
                    0);
                    return(decrypted);
        }

        public void startApp() {
        }
        public void pauseApp() {
        }
        public void destroyApp(boolean unconditional) {
        }
}
```



7.5 Latihan

Buat sebuah method yang akan melakukan verifikasi atas integritas dari sebuah message dengan cara mengolah dan membandingkan antaradigest dari message tersebut dengan message digest asli.

public boolean verifyDigest(byte[] message, byte[] originalDigest)