
Tugas Matakuliah
EC7010 Keamanan Sistem Lanjut

**Kajian dan Implementasi Sistem Keamanan Data pada
Ponsel Berbasis J2ME Menggunakan Profile MIDP 1.0**

Disusun oleh :
Antonius Aditya Hartanto
23202054

Program Magister Teknologi Informasi
Institut Teknologi Bandung
2003

Daftar Isi

I	Pendahuluan	3
II	Paket Bouncy Castle.....	7
III	Enkripsi Simetrik RC4 untuk Proteksi Data	15
IV	Model Otorisasi	24
IV.1	Transport Layer Security	24
IV.2	Otentifikasi SSL dan TLS.....	24
V	TLS pada MIDP 1.0	24
VI	TLS pada MIDP 2.0	25
VII	Jenis Otentifikasi.....	26
VII.1	Otentifikasi Password pada MIDP	26
VII.2	Otentifikasi Client Menggunakan Sertifikat.....	26
VIII	Menandatangani sebuah MIDlet Suite	27
IX	Membuat Sertifikat Tanda Tangan	27
X	Menambahkan Sertifikat ke dalam descriptor Aplikasi.....	27
XI	Menciptakan tanda tangan digital RSA SHA-1 signature dari file JAR	28
XII	Memverifikasi Sertifikat dari Pemberi Tanda Tangan	28
XIII	Proses verifikasi MIDlet Suite JAR	28
	Daftar Pustaka	29

I Pendahuluan

Pada waktu dahulu, metode pengamanan data dilakukan dengan cara mengenkripsi data sebelum dikirimkan. Kekuatan enkripsi ini ditekankan pada kerahasiaan cara enkripsi data yang dilakukan, dan umum digunakan oleh raja-raja dahulu semacam Romawi. Saat ini kekuatan enkripsi data bukan lagi terletak dari kerahasiaan dari algoritma yang ada, namun terletak pada kemampuan sulit dipecahkannya dari sisi waktu. Semakin lama waktu yang diperlukan untuk memecahkan data yang dienkripsi dengan metode coba-coba, maka semakin bagus algoritma enkripsi yang ada.

Banyak sekali aspek keamanan dalam transaksi di internet, namun dalam bahasan dengan MIDP hanya akan dibicarakan mengenai bagaimana membuat data yang dikirim dan diterima dapat diamankan, atau dengan kata lain dienkripsi sehingga terlindung dari pencuri data. Idealnya setiap data yang ditransmisikan harusnya terjamin :

1. Integritas data
Jaminan integritas data sangat penting, sehingga data yang dikirimkan akan sama persis dengan data yang diterima, tanpa mengalami perubahan apapun selama ditransmisikan.
2. Kerahasiaan data
Jaminan kerahasiaan data juga penting, karena dengan ini maka tidak ada pihak lain yang bisa membaca data yang ada selama data tersebut ditransmisikan.
3. Otentikasi akses data
Mekanisme otentikasi akses data menjamin bahwa data ditransmisikan oleh pihak yang benar dengan tujuan transmisi pihak yang benar pula.

Jika berbicara tentang keamanan, maka mau tak mau kita akan membicarakan tentang Kriptografi. Berbicara tentang Kriptografi, maka kita sedang membicarakan tentang cabang ilmu matematika yang memanfaatkan proses komputasi untuk mengacak data yang akan dikirimkan. Secara umum, Kriptografi didasarkan pada penggunaan kunci. Kunci adalah sebuah bilangan yang membuat sebuah persoalan matematika dapat diselesaikan. Untuk analogi perhatikan contoh berikut :

$$3 + \boxed{2} = 5$$

Dari contoh diatas jika persoalan matematikanya adalah :

$$3 + \dots = 5$$

Maka bilangan yang merupakan kunci untuk menyelesaikan problem matematika tersebut adalah

$\boxed{2}$

Jika kunci yang digunakan untuk menyelesaikan problem tersebut bukan $\boxed{2}$, maka persoalan tersebut tidak dapat diselesaikan dengan tepat.. Dalam kasus keamanan data, sebuah data harus diacak terlebih dahulu dan ditentukan komponen kuncinya sebelum dikirimkan ke tujuan. Setelah terkirim, di tempat tujuan, data yang teracak tersebut tidak mungkin dikonstruksi ke

bentuk aslinya tanpa menggunakan komponen kunci tadi. Untuk itu, Kunci tersebut merupakan komponen yang rahasia.

Sebagian besar kriptografi didasarkan pada pemanfaatan persamaan sederhana menggunakan sejumlah besar bilangan yang lebih besar daripada tipe data integer atau long. Algoritma yang memungkinkan data tersebut tetap rahasia selama pengacakan dinamakan dengan *cipher*. Algoritma ini dapat mentranslasikan data biasa yang dinamakan dengan plaintext dan data terenkripsi yang dinamakan *ciphertext*. Jadi secara sederhana cipher adalah persamaan sederhana yang memungkinkan diperoleh data terenkripsi.

Jika mengambil contoh penjumlahan diatas, maka yang dinamakan dengan plaintext adalah bilangan 3, dan yang dimaksud dengan cipher adalah fungsi + . Agar diperoleh data yang terenkripsi diperlukan key atau kunci seperti yang telah dikemukakan sebelumnya. Dalam contoh diatas, maka yang menjadi kuncinya adalah 2. Sehingga yang dimaksud dengan data terenkripsi adalah bilangan 5. Proses pengembalian hasil enkripsi ini dinamakan dengan dekripsi.

Dunia kriptografi menawarkan beberapa solusi penting yang saat ini sudah sangat umum digunakan dalam jaringan internet, yakni :

1. Chiper

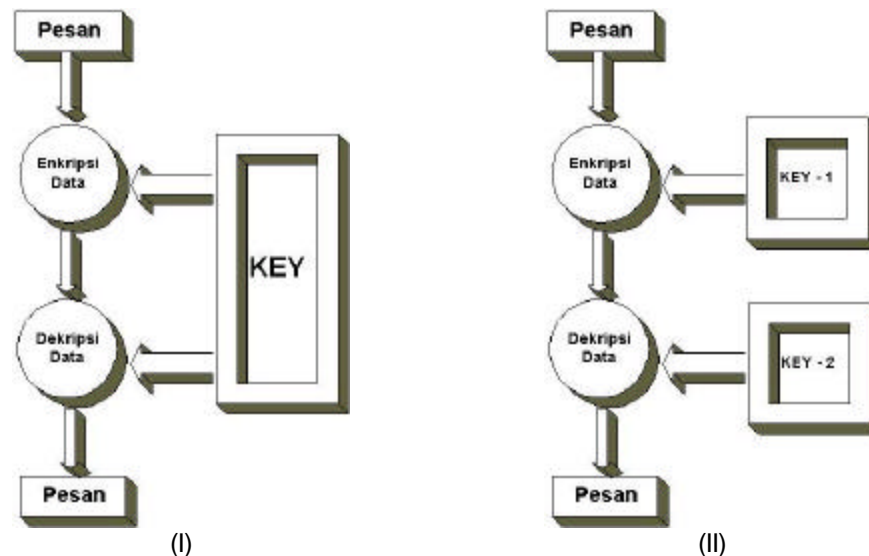
Tak lain adalah teknologi untuk enkripsi dan dekripsi data. Teknik kriptografi data untuk enkripsi ada 2 macam :

a. Kriptografi simetrik

Dengan model kriptografi ini data dienkripsi dan didekripsi dengan kunci rahasia yang sama.

b. Kriptografi asimetrik

Dengan model kriptografi ini data dienkripsi dan didekripsi dengan kunci rahasia yang berbeda. Pasangan kunci untuk enkripsi dan dekripsi dikenal dengan *private key* atau *secret key* dan *public key*. Suatu data yang dienkripsi dengan suatu *private key* hanya bisa didekripsi dengan *public key* yang berkaitan dengan *private key* tersebut, demikian sebaliknya.



Gambar 1. Metode enkripsi simetrik (I) dan asimetrik (II)

Cipher Asymmetric menggunakan sepasang kunci yang berhubungan satu sama lain. Satu merupakan *public key*, sedangkan lainnya adalah *private key*. Data yang terenkripsi menggunakan kunci yang satu, harus didekripsi menggunakan kunci yang lain. Public key dapat didistribusikan secara bebas, akan tetapi private key harus tetap pribadi untuk menjaga keamanan data tersebut.

Cipher Asymmetric berguna untuk tujuan otentifikasi yang berarti pula berfungsi sebagai identitas. Ketika seseorang mengirimkan sebuah pesan yang terenkripsi dengan memanfaatkan public key dimana anda yakin dengan private key yang anda miliki sajalah maka data tersebut dapat didekripsi oleh anda. Hal ini berarti bahwa telah terjadi proses otentifikasi kepada pengirim yaitu bahwa private key yang dikirimkan oleh si pengirim menjadi identitas dari pengirim pribadi.

Tentunya cipher asymmetric jauh lebih kompleks dalam perhitungan matematis dibandingkan cipher symmetric. Namun biasanya enkripsi menggunakan Cipher symmetric berlangsung lebih cepat dibandingkan dengan bila menggunakan cipher asymmetric, terlebih lagi jika digunakan untuk mengenkripsi data yang lebih panjang. Selain itu sering pula memanfaatkan kedua jenis cipher tersebut secara bersamaan. Beberapa contoh algoritma cipher yang ada diantaranya adalah Rjndael, DES, Blowfish, ElGamal, dan sebagainya.

Kunci dapat dibangkitkan dari bilangan acak. Public key dan private key merupakan pasangan kunci yang secara matematis berhubungan namun dapat dibangkitkan secara acak menggunakan apa yang dinamakan dengan *pseudo-random number generator* (PRNG), yang mampu menghasilkan sederetan bit yang berulang. bits. Penggunaan dua buah PRNG yang diinisialisasi secara identik akan menghasilkan deretan bit yang sama.

Cara lain untuk membangkitkan kunci adalah penggunaan kunci yang dinamakan *agreement protocol*. Hal ini merupakan sebuah trik matematis yang memungkinkan dua pihak untuk menggunakan kunci tersebut pada sebuah *session key*. Untuk itu, masing-masing pihak harus mengerti pihak yang lain. Setiap perubahan dari pihak yang lain diperoleh melalui *session key* tersebut. Salah satu contohnya adalah Diffie-Hellman.

2. Message Digest/Digital Fingerprint

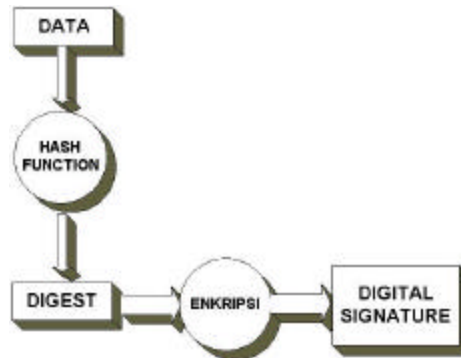
Idenya adalah membuat suatu ringkasan matematik atas sebuah data, dimana perubahan atas data tersebut akan menghasilkan ringkasan matematik yang berbeda. Sebagai contoh ada sebuah data X memiliki nilai digital fingerprint : d07182257bce13d49a5c183864e4a277, maka jika anda merubah barang 1 byte data tersebut, maka nilai digital fingerprint akan berbeda. Teknologi ini menyediakan solusi bahwa data selalu terjamin integritasnya karena perubahan atas data yang ditransmisikan akan terdeteksi dengan mudah. Contoh algoritma message digest adalah : MD2, MD4, MD5, SHA1, RIPEM160, dan RIPEM128.

Sebuah *message digest* berfungsi sebagai sidik jari bagi sebuah data. Dengan Sidik jari yang berbeda, maka data yang diperoleh juga berbeda sekalipun perubahan tersebut hanya 1 bit saja. Jadi secara umum *message digest* digunakan untuk memeriksa integritas sebuah data. Hal tersebut bisa anda lakukan ketika anda mendownload sebuah file dari server, maka anda dapat menghitung nilai digest-nya dan membandingkannya dengan nilai hasil komputasi di server. Jika hasilnya sama, maka anda boleh yakin bahwa selama proses pengiriman, data tidak mengalami modifikasi oleh pihak lain.

3. Digital Signatures

Idenya adalah mengenkripsi message digest dari suatu data dengan kunci rahasia (*private key*) seseorang. Pihak lain kemudian dapat mendapatkan message digest aslinya dengan mendekripsi data dengan kunci publik dari orang tersebut. Pengembangan dari model ini adalah adanya *certificate* yakni setiap orang memiliki semacam *certificate* yang disahkan

oleh pihak ketiga yang ditunjuk, misalnya pengadilan, yang bisa digunakan untuk identitas digital seseorang. Jadi fungsi *certificate* ini seperti kartu tanda penduduk atau kartu SIM. Contoh algoritma signature adalah RSA dan DSA.



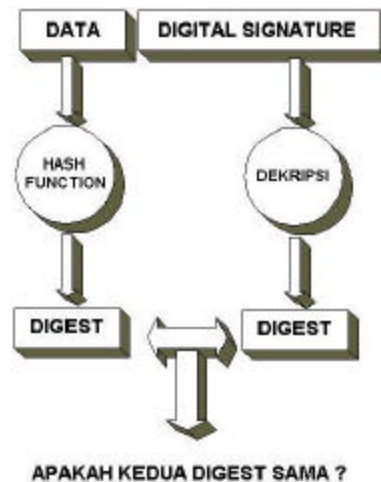
Gambar 2. Metode pembuatan digital signature

Pada algoritma tersebut, data yang dikirimkan ke pihak penerima nantinya tidak sekedar data saja, tapi juga data ditambah dengan *digital signature*-nya.



Gambar 3. Data yang dikirim ditambahi dengan digital signature

Berbicara mengenai digital signature, mungkin anda menjadi bertanya-tanya bagaimana cara penerima data mengetahui bahwa data yang diterima benar-benar terjamin keamanannya dan benar-benar dikirimkan oleh pihak yang benar? Penjelasan nya adalah sebagai berikut, sebagaimana diketahui bahwa data dikirim bersama dengan *digital signature*-nya. Dari digital signature dilakukan dekripsi dengan menggunakan kunci publik dari pihak yang bersangkutan. Jika gagal didekripsi, maka kunci publik tentunya tidak bersesuaian dengan kunci privat (*private key*) dari pihak yang benar, sehingga dari sini kita sudah bisa menentukan bahwa pengirim data adalah pihak yang sah. Dari dekripsi yang dilakukan dihasilkan digest dari data, jika nilai digest ini sama dengan hasil nilai digest yang kita lakukan terhadap data maka data terjamin keamanannya.



Gambar 4. Proses pemeriksaan message digest

II Paket Bouncy Castle

Dalam ruang lingkup J2SE (*Java 2 Standar Edition*) yang didedikasikan pada perangkat dengan memori yang besar semacam komputer PC, Sun Microsystems telah mengembangkan dukungan Java API untuk urusan kriptografi yakni berupa JCA (*Java Cryptography Architecture*) dan JCE (*Java Cryptography Extension*). Namun JCA dan JCE tidak cocok untuk digunakan dalam J2ME karena terlalu besar memori yang dibutuhkannya, padahal perangkat J2ME misal ponsel hanya memiliki kapasitas memori yang terbatas. Sebagai alternatif, untuk J2ME anda bisa menggunakan paket kriptografi dari *Bouncy Castle*. *Bouncy Castle Cryptography* (<http://www.bouncycastle.org>) merupakan proyek open source yang berpusat di Australia. Selain menyediakan paket-paket API java kriptografi untuk J2SE, juga disediakan paket-paket kriptografi untuk J2ME. Dari paket *Bouncy Castle*, akan coba dijelaskan beberapa penggunaan kelas kriptografi dengan Java yang penting dalam bab berikutnya. Penggunaan lebih lanjut dari paket *BouncyCastle* ini dapat anda lihat di situsnya di <http://www.bouncycastle.org>.

Dalam kasus keamanan data pada Java, solusi menggunakan message digest juga dapat dilakukan. Idenya adalah kita membuat message digest dari nilai password yang ada untuk dikirimkan ke server. Di sisi server, cukup dilakukan penyamaan message digest yang dikirim dengan message digest dari password yang tersimpan di server, jika sama maka login sah, jika tidak maka sesi login tidak sah. Salah satu contoh algoritma Message Digest yang sering digunakan adalah MD5.

Jika anda pernah menggunakannya pada pemrograman PHP, maka syntak-nya kurang lebih adalah :

```
$mdigest = md5("aditya");
```

Jika algoritma MD5 diterapkan pada pemrograman Java, maka implementasinya salah satunya dapat dilihat dari listing program berikut ini :

```
/*
 * MD5 Message Digest Implementation
 * From Bouncycastle.org, rewritten by Antonius Aditya Hartanto
 */
```

```
class MD5Digest {

    private byte[] xBuf;
    private long   byteCount;
    private static final int    DIGEST_LENGTH = 16;
    private int    xBufOff, xOff, H1, H2, H3, H4;
    private int[]  X = new int[16];
    private static final char[] kDigits = {
        '0','1','2','3','4','5','6','7','8','9',
        'a','b','c','d','e','f'    };

    public MD5Digest(){
        xBuf = new byte[4];
        xBufOff = 0;
        reset();
    }

    public String calculate(String m){
        update(m.getBytes(), 0, m.getBytes().length);
        byte[] digestValue = new byte[DIGEST_LENGTH];
        doFinal(digestValue, 0);
        return bytesToHex(digestValue);
    }

    public String bytesToHex(byte[] raw) {
        int length = raw.length;
        char[] hex = new char[length * 2];
        for (int i = 0; i < length; i++) {
            int value = (raw[i] + 256) % 256;
            int highIndex = value >> 4;
            int lowIndex = value & 0x0f;
            hex[i * 2 + 0] = kDigits[highIndex];
            hex[i * 2 + 1] = kDigits[lowIndex];
        }
        return (new String(hex)).toString();
    }

    public void reset(){
        byteCount = 0;
        xBufOff = 0;
        for(int i = 0; i < xBuf.length; i++) { xBuf[i] = 0; }
        H1 = 0x67452301; H2 = 0xefcdab89;
        H3 = 0x98badcfe;
        H4 = 0x10325476;
        xOff = 0;
        for (int i = 0; i != X.length; i++){ X[i] = 0; }
    }

    public void finish(){
        long bitLength = (byteCount << 3);
        update((byte)128);
        while (xBufOff != 0){
            update((byte)0);
        }
        processLength(bitLength);processBlock();
    }

    public void update(byte in){
```

```
xBuf[xBufOff++] = in;

if (xBufOff == xBuf.length){
    processWord(xBuf, 0);
    xBufOff = 0;
}
byteCount++;
}

public void update(byte[] in, int inOff, int len){
    while ((xBufOff != 0) && (len > 0)){
        update(in[inOff]);
        inOff++;len--;
    }
    while (len > xBuf.length){
        processWord(in, inOff);
        inOff += xBuf.length; len -= xBuf.length;
        byteCount += xBuf.length;
    }
    while (len > 0){
        update(in[inOff]);
        inOff++;len--;
    }
}

protected void processWord(byte[] in, int inOff){
    X[xOff++] = (in[inOff] & 0xff) | ((in[inOff + 1] & 0xff) << 8)
        | ((in[inOff + 2] & 0xff) << 16)
        | ((in[inOff + 3] & 0xff) << 24);
    if (xOff == 16){
        processBlock();
    }
}

protected void processLength(long bitLength){
    if (xOff > 14){ processBlock(); }
    X[14] = (int)(bitLength & 0xffffffff);
    X[15] = (int)(bitLength >>> 32);
}

private void unpackWord(int word, byte[] out, int outOff){
    out[outOff] = (byte)word;
    out[outOff + 1] = (byte)(word >>> 8);
    out[outOff + 2] = (byte)(word >>> 16);
    out[outOff + 3] = (byte)(word >>> 24);
}

public int doFinal(byte[] out, int outOff){
    finish();
    unpackWord(H1, out, outOff);
    unpackWord(H2, out, outOff + 4);
    unpackWord(H3, out, outOff + 8);
    unpackWord(H4, out, outOff + 12);
    reset();
    return DIGEST_LENGTH;
}
// round 1 left rotates
private static final int S11 = 7;
```

```
private static final int S12 = 12;
private static final int S13 = 17;
private static final int S14 = 22;

// round 2 left rotates
private static final int S21 = 5;
private static final int S22 = 9;
private static final int S23 = 14;
private static final int S24 = 20;

// round 3 left rotates
private static final int S31 = 4;
private static final int S32 = 11;
private static final int S33 = 16;
private static final int S34 = 23;

// round 4 left rotates
private static final int S41 = 6;
private static final int S42 = 10;
private static final int S43 = 15;
private static final int S44 = 21;

// rotate int x left n bits.
private int rotateLeft(int x,int n){
    return (x << n) | (x >>> (32 - n)); }
    // F, G, H and I are the basic MD5 functions.
    private int F(int u, int v, int w){ return (u & v) | (~u &
w); }
    private int G(int u, int v, int w){ return (u & w) | (v &
~w); }
    private int H(int u, int v, int w){ return u ^ v ^ w; }
    private int K(int u, int v, int w){ return v ^ (u | ~w); }

protected void processBlock(){
    int a = H1; int b = H2;
    int c = H3; int d = H4;
    // Round 1 - F cycle, 16 times.
    a = rotateLeft((a + F(b, c, d) + X[ 0] + 0xd76aa478), S11) + b;
    d = rotateLeft((d + F(a, b, c) + X[ 1] + 0xe8c7b756), S12) + a;
    c = rotateLeft((c + F(d, a, b) + X[ 2] + 0x242070db), S13) + d;
    b = rotateLeft((b + F(c, d, a) + X[ 3] + 0x1bdceee), S14) + c;
    a = rotateLeft((a + F(b, c, d) + X[ 4] + 0xf57c0faf), S11) + b;
    d = rotateLeft((d + F(a, b, c) + X[ 5] + 0x4787c62a), S12) + a;
    c = rotateLeft((c + F(d, a, b) + X[ 6] + 0xa8304613), S13) + d;
    b = rotateLeft((b + F(c, d, a) + X[ 7] + 0xfd469501), S14) + c;
    a = rotateLeft((a + F(b, c, d) + X[ 8] + 0x698098d8), S11) + b;
    d = rotateLeft((d + F(a, b, c) + X[ 9] + 0x8b44f7af), S12) + a;
    c = rotateLeft((c + F(d, a, b) + X[10] + 0xffff5bb1), S13) + d;
    b = rotateLeft((b + F(c, d, a) + X[11] + 0x895cd7be), S14) + c;
    a = rotateLeft((a + F(b, c, d) + X[12] + 0x6b901122), S11) + b;
    d = rotateLeft((d + F(a, b, c) + X[13] + 0xfd987193), S12) + a;
    c = rotateLeft((c + F(d, a, b) + X[14] + 0xa679438e), S13) + d;
    b = rotateLeft((b + F(c, d, a) + X[15] + 0x49b40821), S14) + c;
    // Round 2 - G cycle, 16 times.
    a = rotateLeft((a + G(b, c, d) + X[ 1] + 0xf61e2562), S21) + b;
    d = rotateLeft((d + G(a, b, c) + X[ 6] + 0xc040b340), S22) + a;
    c = rotateLeft((c + G(d, a, b) + X[11] + 0x265e5a51), S23) + d;
    b = rotateLeft((b + G(c, d, a) + X[ 0] + 0xe9b6c7aa), S24) + c;
```

```

a = rotateLeft((a + G(b, c, d) + X[ 5] + 0xd62f105d), S21) + b;
d = rotateLeft((d + G(a, b, c) + X[10] + 0x02441453), S22) + a;
c = rotateLeft((c + G(d, a, b) + X[15] + 0xd8a1e681), S23) + d;
b = rotateLeft((b + G(c, d, a) + X[ 4] + 0xe7d3fbc8), S24) + c;
a = rotateLeft((a + G(b, c, d) + X[ 9] + 0x21e1cde6), S21) + b;
d = rotateLeft((d + G(a, b, c) + X[14] + 0xc33707d6), S22) + a;
c = rotateLeft((c + G(d, a, b) + X[ 3] + 0xf4d50d87), S23) + d;
b = rotateLeft((b + G(c, d, a) + X[ 8] + 0x455a14ed), S24) + c;
a = rotateLeft((a + G(b, c, d) + X[13] + 0xa9e3e905), S21) + b;
d = rotateLeft((d + G(a, b, c) + X[ 2] + 0xfcefa3f8), S22) + a;
c = rotateLeft((c + G(d, a, b) + X[ 7] + 0x676f02d9), S23) + d;
b = rotateLeft((b + G(c, d, a) + X[12] + 0x8d2a4c8a), S24) + c;
// Round 3 - H cycle, 16 times.
a = rotateLeft((a + H(b, c, d) + X[ 5] + 0xfffa3942), S31) + b;
d = rotateLeft((d + H(a, b, c) + X[ 8] + 0x8771f681), S32) + a;
c = rotateLeft((c + H(d, a, b) + X[11] + 0x6d9d6122), S33) + d;
b = rotateLeft((b + H(c, d, a) + X[14] + 0xfde5380c), S34) + c;
a = rotateLeft((a + H(b, c, d) + X[ 1] + 0xa4beea44), S31) + b;
d = rotateLeft((d + H(a, b, c) + X[ 4] + 0x4bdecfa9), S32) + a;
c = rotateLeft((c + H(d, a, b) + X[ 7] + 0xf6bb4b60), S33) + d;
b = rotateLeft((b + H(c, d, a) + X[10] + 0xbebfbc70), S34) + c;
a = rotateLeft((a + H(b, c, d) + X[13] + 0x289b7ec6), S31) + b;
d = rotateLeft((d + H(a, b, c) + X[ 0] + 0xeaad127fa), S32) + a;
c = rotateLeft((c + H(d, a, b) + X[ 3] + 0xd4ef3085), S33) + d;
b = rotateLeft((b + H(c, d, a) + X[ 6] + 0x04881d05), S34) + c;
a = rotateLeft((a + H(b, c, d) + X[ 9] + 0xd9d4d039), S31) + b;
d = rotateLeft((d + H(a, b, c) + X[12] + 0xe6db99e5), S32) + a;
c = rotateLeft((c + H(d, a, b) + X[15] + 0x1fa27cf8), S33) + d;
b = rotateLeft((b + H(c, d, a) + X[ 2] + 0xc4ac5665), S34) + c;
// Round 4 - K cycle, 16 times.
a = rotateLeft((a + K(b, c, d) + X[ 0] + 0xf4292244), S41) + b;
d = rotateLeft((d + K(a, b, c) + X[ 7] + 0x432aff97), S42) + a;
c = rotateLeft((c + K(d, a, b) + X[14] + 0xab9423a7), S43) + d;
b = rotateLeft((b + K(c, d, a) + X[ 5] + 0xfc93a039), S44) + c;
a = rotateLeft((a + K(b, c, d) + X[12] + 0x655b59c3), S41) + b;
d = rotateLeft((d + K(a, b, c) + X[ 3] + 0x8f0ccc92), S42) + a;
c = rotateLeft((c + K(d, a, b) + X[10] + 0xffeff47d), S43) + d;
b = rotateLeft((b + K(c, d, a) + X[ 1] + 0x85845dd1), S44) + c;
a = rotateLeft((a + K(b, c, d) + X[ 8] + 0x6fa87e4f), S41) + b;
d = rotateLeft((d + K(a, b, c) + X[15] + 0xfe2ce6e0), S42) + a;
c = rotateLeft((c + K(d, a, b) + X[ 6] + 0xa3014314), S43) + d;
b = rotateLeft((b + K(c, d, a) + X[13] + 0x4e0811a1), S44) + c;
a = rotateLeft((a + K(b, c, d) + X[ 4] + 0xf7537e82), S41) + b;
d = rotateLeft((d + K(a, b, c) + X[11] + 0xbd3af235), S42) + a;
c = rotateLeft((c + K(d, a, b) + X[ 2] + 0x2ad7d2bb), S43) + d;
b = rotateLeft((b + K(c, d, a) + X[ 9] + 0xeb86d391), S44) + c;
H1 += a; H2 += b; H3 += c; H4 += d;
xOff = 0;
for (int i = 0; i != X.length; i++) { X[i] = 0; }
}
}

```

Simpan kode implementasi MD5 diatas file bernama MD5Digest.java, dan anda sekarang sudah punya pustaka tambahan untuk MD5 ini. Penggunaan dari kelas MD5 diatas sangat sederhana, semacam ini :

```
M5Digest s = new MD5Digest();
String val = s.calculate("TES DATA");
```

Jadi fungsi yang dipanggil dari objek MD5Digest yang dibuat adalah calculate() saja. Contoh penggunaannya adalah :

```
import java.io.*;

class sampleMD5Usage {
    public static void main(String args[]){
        try {
            String s;
            BufferedReader b;
            MD5Digest m = new MD5Digest();
            b = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Masukkan data : ");
            s = b.readLine();
            System.out.println("Digest of \"" + s + "\"" is " +
                m.calculate(s));
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

Jika anda eksekusi aplikasi Java di atas pada *console*, maka akan diperoleh hasil semacam ini :

```
C:\javaprogram>javac -target 1.1 MD5Digest.java
C:\javaprogram>javac -target 1.1 sampleMD5Usage.java

C:\javaprogram>java sampleMD5Usage
Masukkan data : data1
Digest of "data1" is 89d903bc35dede724fd52c51437ff5fd
```

Jika anda akan memakainya untuk aplikasi MIDlet simpan file MD5Digest.java pada direktori src dari proyek J2ME anda. Kembali ke aplikasi MIDlet, coba buat proyek "proteksi" dengan J2ME Wireless Toolkit dan buatlah kode MIDlet berikut :

```
/*
 * Nama File:c:\J2mewtk\apps\proteksi\src\formMD5.java(J2ME Toolkit 1.0.3)
 * c:\WTK104\apps\proteksi\src\formMD5.java (J2ME Toolkit 1.0.4)
 * Deskripsi: MIDP client untuk akses form login dengan memberikan password
 * dengan message digest MD5
 */

import javax.microedition.io.*;
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;

public class formMD5 extends MIDlet
implements CommandListener {

    private Display display;
    private Command openCmd =
```

```
        new Command("Login",Command.OK,1);
private Command exitCmd =
        new Command("Keluar",Command.EXIT,2);
private Command backCmd =
        new Command("Kembali",Command.EXIT,2);
private Form f1,f2;
private TextField t1,t2,t3;
private String URL =
        new String("http://127.0.0.1/someaction.php");

public formMD5(){
public void startApp(){
    display      = Display.getDisplay(this);
    f1           = new Form("Form Login");
    t1           = new TextField("Username","",256,TextField.ANY);
    t2           =
        new TextField("Password","",256,TextField.PASSWORD);
    f1.append(t1); f1.append(t2);
    f1.addCommand(openCmd); f1.addCommand(exitCmd);
    f1.setCommandListener(this); display.setCurrent(f1);
}
public void formHasilUrl(){
    f2 = new Form("Status Login");
    t3 = new TextField("",null,1024,TextField.ANY);
    bukaUrl(); f2.append(t3); f2.addCommand(backCmd);
    f2.setCommandListener(this);
    display.setCurrent(f2);
}
public void bukaUrl(){
    HttpURLConnection c = null;
    InputStream is = null;
    byte[] data;

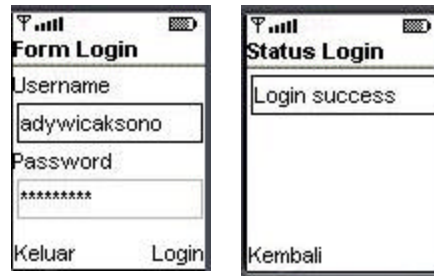
    try{
        // Create the message digest.
        MD5Digest digest = new MD5Digest();
        // Calculate the digest value & send to web server
        c = (HttpURLConnection)Connector.open(URL
            +"?login=" + t1.getString()
            + "&p=" + digest.calculate(t2.getString()));

        is = c.openInputStream();
        f2.setTitle("Status Login");

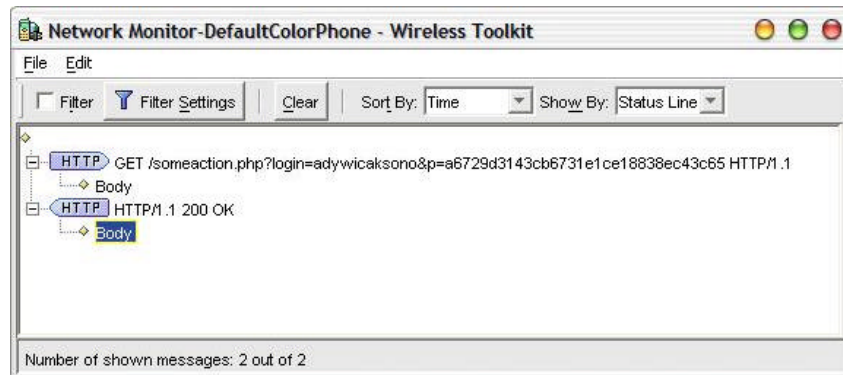
        if(c.getResponseCode() == HttpURLConnection.HTTP_OK){
            data = new byte[(int)c.getLength()];
            int actual = is.read(data);
            t3.setString((new String(data)).toString());
        }
        if (is != null){ is.close();}
        if (c != null){ c.close(); }
    }catch(IOException e){
        t3.setString("Error I/O : " + e.toString());
    }
}
public void keluar(){ destroyApp(true); }
```

```
public void kembali(){ display.setCurrent(f1); }
public void commandAction(Command c, Displayable d){
    String lbl = c.getLabel();
    if(lbl.equals("Keluar")){
        keluar();
    }else if(lbl.equals("Login")){
        formHasilUrl();
    }else if(lbl.equals("Kembali")){
        kembali();
    }
}
public void pauseApp() {}
public void destroyApp(boolean unconditional){notifyDestroyed();}
}
```

MIDlet sederhana diatas akan mengakses aplikasi PHP sederhana untuk pengecekan password. Perhatikan hasil eksekusi berikut dengan proses koneksi HTTP via MIDlet yang terjadi, dimana tampak password tidak dikirimkan apa adanya lagi tapi sudah berupa *message digest* dari password



Gambar 5. Hasil eksekusi MIDlet dengan password dikirim via MD5



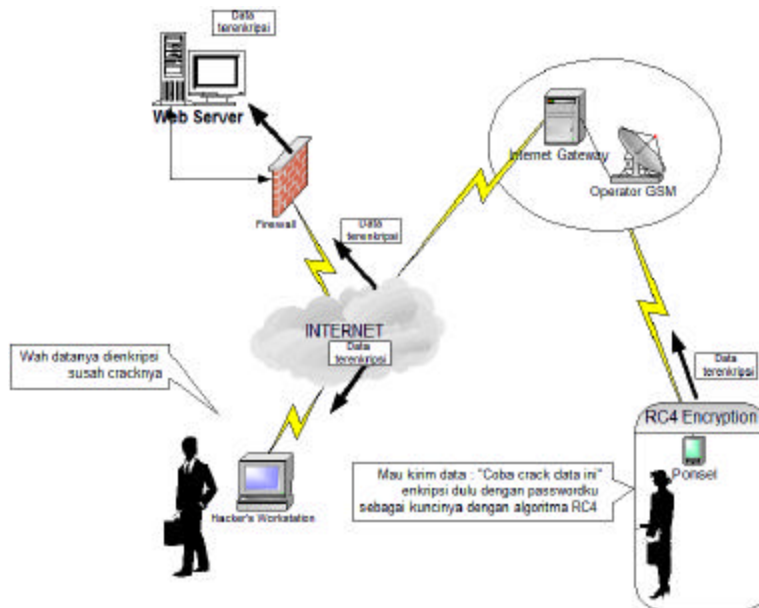
Gambar 6. Perhatikan request HTTP yang dikirim MIDlet

Pendekatan diatas tentulah tidak sempurna, jadi jika anda ingin transaksi internet yang aman gunakanlah SSL. Sayang sekali spesifikasi MIDP 1.0 yang ada saat ini belum mendukung penggunaan SSL, karena implementasi untuk SSL relatif rumit. Dukungan SSL ini akan ada pada MIDP 2.0.

III Enkripsi Simetrik RC4 untuk Proteksi Data

Pendekatan lain untuk mengamankan data anda selama transaksi network selama MIDP 1.0 belum mendukung SSL adalah dengan menggunakan metode enkripsi baik simetrik atau asimetrik. Penulis akan mencoba mengulas penggunaan algoritma RC4 yang merupakan algoritma enkripsi simetrik untuk pengamanan data ini. Algoritma RC4 ini diciptakan oleh ahli matematik Ron Rivest penemu algoritma RSA yang kemudian dipatenkan dan menjadi nama perusahaan keamanan data terkemuka, yakni RSA (www.rsa.com). RC4 digunakan oleh banyak sekali aplikasi besar semacam Oracle dan Microsoft SQL Server. Kembali ke aplikasi yang akan mencoba RC4, skenario yang terjadi antara aplikasi client dan server digambarkan sebagai berikut:

1. Client MIDlet pada ponsel ingin mengirimkan data rahasia ke web server
2. Web server telah menyimpan informasi password pengguna, misalnya "rahasia". Informasi password ini hanya dimiliki oleh web server dan pengguna ponsel.
3. Client MIDlet mengirimkan data yang dienkripsi dengan RC4, dimana kunci enkripsi yang digunakan adalah passwordnya. Dalam hal ini hasil enkripsi hanya akan bisa dibuka dengan algoritma RC4 dan kunci dekripsi password yang diketahui oleh pengguna ponsel dan web server saja.



Gambar 7. Data sebelum dikirim dienkripsi dahulu, sehingga mempersulit hacker mengenalinya

Implementasi dari algoritma RC4 ini ditulis ulang oleh penulis dari kode asli di www.bouncycastle.org sebagai berikut :

```
/* Implementasi enkripsi simetrik
RC4, kode diambil dari www.bouncycastle.org
Nama file : RC4Engine.java
*/

class KeyParameter{
    private byte[] key;
    public KeyParameter(byte[] key){
```

```
        this(key, 0, key.length);
    }

    public KeyParameter(byte[] key, int keyOff,
                        int keyLen){
        this.key = new byte[keyLen];
        System.arraycopy(key, keyOff, this.key, 0, keyLen);
    }

    public byte[] getKey(){
        return key;
    }
}

class EncRC4Engine{
    private final static int STATE_LENGTH = 256;
    private byte[] engineState = null, workingKey = null;
    private int x = 0, y = 0;
    private static final char[] kDigits = {
        '0','1','2','3','4','5','6','7','8','9',
        'a','b','c','d','e','f' };
    EncRC4Engine(){ // Konstruktor

    /**
     * Inisialisasi chiper RC4
     * Parameter "forEncryption" harus diset
     * - true jika akan dilakukan proses enkripsi
     * - false jika akan dilakukan proses dekripsi
     * Parameter "params" merupakan kunci untuk proses enkripsi/dekripsi
     */
    public void init(boolean forEncryption,
                    KeyParameter params){
        if (params instanceof KeyParameter){
            workingKey = ((KeyParameter)params).getKey();
            setKey(workingKey);
            return;
        }
        throw new
IllegalArgumentException("invalid parameter passed to RC4 init - "
                            + params.getClass().getName());
    }

    public void processBytes(byte[] in, int inOff,
                             int len, byte[] out, int outOff){
        if ((inOff + len) > in.length){
            throw new RuntimeException("input buffer too short");
        }
        if ((outOff + len) > out.length){
            throw new RuntimeException("output buffer too short");
        }

        for (int i = 0; i < len ; i++){
            x = (x + 1) & 0xff;
            y = (engineState[x] + y) & 0xff;
            // swap
            byte tmp = engineState[x];
            engineState[x] = engineState[y];
            engineState[y] = tmp;
        }
    }
}
```

```
        // xor
        out[i+outOff] = (byte)(in[i + inOff]
            ^ engineState[(engineState[x] + engineState[y])
                & 0xff]);
    }
}
public String bytesToHex(byte[] raw) {
    int length = raw.length;
    char[] hex = new char[length * 2];
    for (int i = 0; i < length; i++) {
        int value = (raw[i] + 256) % 256;
        int highIndex = value >> 4;
        int lowIndex = value & 0x0f;
        hex[i * 2 + 0] = kDigits[highIndex];
        hex[i * 2 + 1] = kDigits[lowIndex];
    }
    return (new String(hex)).toString();
}
public void reset(){
    setKey(workingKey);
}

// Private implementation
private void setKey(byte[] keyBytes){
    workingKey = keyBytes;
    x = 0; y = 0;
    if (engineState == null){
        engineState = new byte[STATE_LENGTH];
    }
    // reset the state of the engine
    for (int i=0; i < STATE_LENGTH; i++){
        engineState[i] = (byte)i;
    }
    int i1 = 0, i2 = 0;
    for (int i=0; i < STATE_LENGTH; i++){
        i2 = ((keyBytes[i1] & 0xff) + engineState[i] + i2) & 0xff;
        // do the byte-swap inline
        byte tmp = engineState[i];
        engineState[i] = engineState[i2];
        engineState[i2] = tmp;
        i1 = (i1+1) % keyBytes.length;
    }
}
}
```

Jika anda ingin menggunakan kelas tersebut dalam sebuah program, silakan coba listing berikut ini:

```
import java.io.*;

class tesRC4Engine {
    public static void main(String args[]){
        try {
            String myKey1, myKey2, text2Encrypt, text2Decrypt;
            BufferedReader b;
            byte[] ciphertext1, ciphertext2;
            RC4Engine s1 = new RC4Engine();
            RC4Engine s2 = new RC4Engine();
        }
    }
}
```

```
b = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Masukkan data untuk dienkrpsi : ");
text2Encrypt = b.readLine();
System.out.print("Masukkan kunci enkripsi : ");
myKey1 = b.readLine();
s1.init(true,new KeyParameter(myKey1.getBytes()));
ciphertext1 = new byte[text2Encrypt.length()];
s1.processBytes(text2Encrypt.getBytes(),
    0, text2Encrypt.length(), ciphertext1, 0);

System.out.print("Hasil Enkripsi dari \" + text2Encrypt );
System.out.println("\n" adalah " + s1.bytesToHex(ciphertext1));

text2Decrypt = s1.bytesToHex(ciphertext1);
System.out.print("Masukkan kunci untuk dekripsi : ");
myKey2 = b.readLine();
s2.init(false,new KeyParameter(myKey2.getBytes()));
ciphertext2 = new byte[ciphertext1.length];
s2.processBytes(ciphertext1, 0, ciphertext1.length,
    ciphertext2, 0);
System.out.print("Hasil Dekripsi dari \" + text2Decrypt);
System.out.println("\n" adalah "
    + (new String(ciphertext2)).toString());
}catch(IOException e){
    e.printStackTrace();
}
}
}
```

Kompilasi dan hasil eksekusinya adalah sebagai berikut :

```
C:\>javac RC4Engine.java
C:\>javac tesRC4Engine.java

C:\>java tesRC4Engine
Masukkan data untuk dienkrpsi : coba crack data ini
Masukkan kunci enkripsi : rahasia
Hasil Enkripsi dari "coba crack data ini"
    adalah c983fc14279dbe0a657cfd0e6fe95819382008
Masukkan kunci untuk dekripsi : rahasia
Hasil Dekripsi dari "c983fc14279dbe0a657cfd0e6fe95819382008"
    adalah coba crack data ini

C:\>java tesRC4Engine
Masukkan data untuk dienkrpsi : tes2
Masukkan kunci enkripsi : rahasia
Hasil Enkripsi dari "tes2" adalah de89ed47
Masukkan kunci untuk dekripsi : tidak tahu
Hasil Dekripsi dari "de89ed47" adalah ]â é
```

Perhatikan bahwa pada contoh eksekusi kedua hasil dekripsi salah, karena kunci dekripsi tidak sama dengan kunci enkripsi. Penggunaan kelas RC4Engine di atas sederhana, yakni sebagai berikut :

1. Buatlah sebuah objek RC4Engine kemudian inialisasi

```
RC4Engine enkripsi = new RC4Engine()  
RC4Engine dekripsi = new RC4Engine()
```

2. Inialisasi mesin enkripsi/dekripsi berikut kunci yang akan digunakan untuk enkripsi/dekripsi. Untuk proses enkripsi :

```
String kunciEnkripsi = "akey";  
enkripsi.init(true,new KeyParameter(kunciEnkripsi.getBytes()));
```

Untuk proses dekripsi

```
String kunciDekripsi = "suatukunci";  
dekripsi.init(false,new KeyParameter(kunciDekripsi.getBytes()));
```

3. Buat sebuah variabel array of bytes yang besar buffernya sepanjang panjang string data yang akan dienkripsi/dekripsi

```
String data1 = "data rahasia"; // Data untuk dienkripsi  
byte[] hasilenkripsi = new byte[data1.length();  
  
// Data untuk didekripsi  
String data2 = "c983fc14279dbe0a657cfd0e6fe95819382008";  
byte[] hasildekripsi = new byte[data2.length();
```

4. Lakukan proses enkripsi/dekripsi dengan memanggil fungsi *processByte()*. Untuk enkripsi :

```
enkripsi.processBytes(data1.getBytes(), 0, data1.length(), hasilenkripsi,  
0);
```

Untuk dekripsi :

```
dekripsi.processBytes(data2.getBytes(), 0,data2.length(), hasildekripsi,  
0);
```

5. Hasil enkripsi dan dekripsi ada pada buffer pada poin 3

Sekarang mari kita coba penggunaannya dalam lingkungan J2ME, dalam kasus dimana kita ingin mengirim sebuah data yang sangat rahasia ke sebuah web server untuk disimpan. Kunci yang dipakai untuk enkripsi dan dekripsi telah dishare secara offline atau manual dan hanya diketahui oleh web server dan pengguna ponsel yang bersangkutan. Aplikasi pada sisi server akan menggunakan servlet Java, dimana panduan instalasinya bisa anda peroleh pada bagian lampiran. Jangan lupa, anda copy file RC4Engine.java ke direktori src/ dari proyek J2ME wireless toolkit anda sebelum mengkompilasi kode MIDP ini. Dan kode MIDlet untuk simpan data rahasia tersebut adalah :

```
/* NamaFile:c:\J2mewtk\apps\proteksi\src\secretData.java(J2METoolkit 1.0.3)  
   c:\WTK104\apps\proteksi\src\secretData.java (J2ME Toolkit 1.0.4)  
   Deskripsi: MIDP client untuk akses form penyimpanan data rahasia dengan  
   memberikan data yang dienkripsi dengan RC4  
   */  
  
import javax.microedition.io.*;
```

```
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;

public class secretData extends MIDlet
implements CommandListener {

    private Display display;
    private Command openCmd =
        new Command("Simpan",Command.OK,1);
    private Command exitCmd =
        new Command("Keluar",Command.EXIT,2);
    private Command backCmd =
        new Command("Kembali",Command.EXIT,2);
    private Form f1,f2;
    private TextField t1,t2;
    private String URL =
        new String("http://127.0.0.1:8080/examples/servlet/saveData");
    // Ini kunci untuk enkripsi dan dekripsi pada RC4
    private String KUNCI_RAHASIA = "secretWord2003";

    public secretData(){}
    public void startApp(){
        display = Display.getDisplay(this);
        f1 = new Form("Data Rahasia");
        t1 = new TextField("Masukkan Data","",1024,TextField.ANY);
        f1.append(t1);
        f1.addCommand(openCmd); f1.addCommand(exitCmd);
        f1.setCommandListener(this); display.setCurrent(f1);
    }
    public void formHasilUrl(){
        f2 = new Form("Status Penyimpanan");
        t2 = new TextField("",null,1024,TextField.ANY);
        bukaUrl(); f2.append(t2); f2.addCommand(backCmd);
        f2.setCommandListener(this);
        display.setCurrent(f2);
    }
    public void bukaUrl(){
        HttpURLConnection c = null;
        InputStream is = null;
        byte[] data;
        String text2Encrypt;
        byte[] ciphertext1,ciphertext2;
        RC4Engine s1;

        try{
            // --- Enkripsi password RC4 dengan KUNCI_RAHASIA
            s1 = new RC4Engine();
            text2Encrypt = t1.getString();
            s1.init(true,new KeyParameter(KUNCI_RAHASIA.getBytes()));
            ciphertext1 = new byte[text2Encrypt.length()];
            s1.processBytes(text2Encrypt.getBytes(), 0,
                text2Encrypt.length(), ciphertext1, 0);
            // Kirim password yang terenkripsi ke server
            System.out.println("Accessing " + URL + "?dt="
                + s1.bytesToHex(ciphertext1));
        }
    }
}
```

```
        c = (HttpConnection)Connector.open(URL + "?dt="
            + sl.bytesToHex(ciphertext1));
        is = c.openInputStream();
        f2.setTitle("Respon Web Server");
        if(c.getResponseCode() == HttpURLConnection.HTTP_OK){
            data = new byte[(int)c.getLength()];
            int actual = is.read(data);
            t2.setString((new String(data)).toString());
        }else{
            t2.setString("HTTP " + c.getResponseCode()
                + " " + c.getResponseMessage());
        }
        if (is != null){ is.close();}
        if (c != null){ c.close(); }
    }catch(IOException e){
        t2.setString("Error I/O : " + e.toString());
        e.printStackTrace();
    }
}

public void keluar(){ destroyApp(true); }
public void kembali(){ display.setCurrent(f1); }

public void commandAction(Command c, Displayable d){
    String lbl = c.getLabel();
    if(lbl.equals("Keluar")){
        keluar();
    }else if(lbl.equals("Simpan")){
        formHasilUrl();
    }else if(lbl.equals("Kembali")){
        kembali();
    }
}
public void pauseApp() {}
public void destroyApp(boolean unconditional){notifyDestroyed();}
}
```

Adapun kode Servlet Java untuk menyimpan data rahasia, jangan lupa copy file RC4Engine.java ke direktori c:\jakarta-tomcat\webapps\examples\WEB-INF\classes dan kompilasi RC4Engine.java tersebut.

```
/*Namafile:C:\jakarta-tomcat\webapps\examples\WEB-INF\classes\saveData.java
   Data hanya diterima dan didekripsi saja tidak disimpan sebagai file
*/
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

class HexCodec {
    private static final char[] kDigits = {
        '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
        'a', 'b', 'c', 'd', 'e', 'f'
    };

    public static char[] bytesToHex(byte[] raw) {
```

```
        int length = raw.length;
        char[] hex = new char[length * 2];
        for (int i = 0; i < length; i++) {
            int value = (raw[i] + 256) % 256;
            int highIndex = value >> 4;
            int lowIndex = value & 0x0f;
            hex[i * 2 + 0] = kDigits[highIndex];
            hex[i * 2 + 1] = kDigits[lowIndex];
        }
        return hex;
    }
    public static byte[] hexToBytes(char[] hex) {
        int length = hex.length / 2;
        byte[] raw = new byte[length];
        for (int i = 0; i < length; i++) {
            int high = Character.digit(hex[i * 2], 16);
            int low = Character.digit(hex[i * 2 + 1], 16);
            int value = (high << 4) | low;
            if (value > 127) value -= 256;
            raw[i] = (byte)value;
        }
        return raw;
    }
    public static byte[] hexToBytes(String hex) {
        return hexToBytes(hex.toCharArray());
    }
}

public class saveData extends HttpServlet {
    // Entri poin request GET dari klien akan dihandle oleh
    // Fungsi ini
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException{

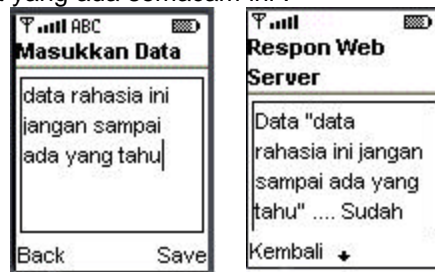
        // Ini kunci untuk enkripsi dan dekripsi pada RC4
        String KUNCI_RAHASIA = "secretWord2003";
        RC4Engine s = new RC4Engine();
        byte[] text;
        byte[] Bytes_ciphertext;

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String ciphertext = request.getParameter("dt");
        if(ciphertext == null){
            out.println("Data variabel dt tidak ada");
        }else{
            Bytes_ciphertext = HexCodec.hexToBytes(ciphertext);
            s.init(false,new KeyParameter(KUNCI_RAHASIA.getBytes()));

            text = new byte[Bytes_ciphertext.length];
            s.processBytes(Bytes_ciphertext, 0,
                Bytes_ciphertext.length, text, 0);
            System.out.println("Ciphertext = " + ciphertext);
            out.println("Data \" + (new String(text)).toString()
                + "\" .... Sudah diterima");
        }
    }
}
```

}

Hasil eksekusi dari MIDlet yang ada semacam ini :

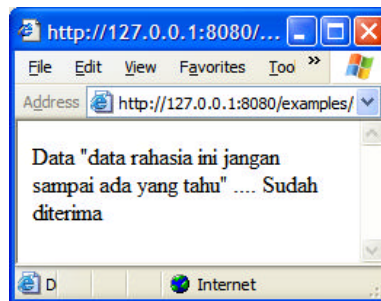


Gambar 8. Pengiriman data dengan enkripsi RC4

Jika anda perhatikan dari network monitoring pada J2ME toolkit versi 1.0.4, maka akan nampak MIDlet mengirimkan request ke Servlet Java dengan URL semacam ini :

```
http://127.0.0.1:8080/examples/servlet/saveData?dt=8c8857f0de18fb255baf3000c8dfe647c951a1cfe36b16fb34ecbe4d37b05660213580c1ad1f8d4ad3ed6877
```

Yang jika anda akses menggunakan Internet Explorer tampak hasil yang sama dengan tampilan di MIDlet ponsel anda.



Gambar 9. Akses URL yang sama dengan MIDlet dengan IE

Seperti telah dikemukakan sebelumnya, pada MIDP versi 1.0, fasilitas untuk membuat aplikasi keamanan belumlah disiapkan secara khusus oleh sang pembuat J2ME. Pada MIDP 2.0 mulai diperkenalkan konsep yang dinamakan dengan *trusted application* yang memungkinkan untuk membatasi penggunaan sebuah aplikasi atau bahkan menolaknya jika MIDlet suite tidak dilengkapi dengan semacam kartu pas atau sertifikat.

Sistem keamanan pada Trusted MIDlet Suite didasarkan pada protection domain. Protection domain ini berisi sertifikat yang dimaksudkan agar mengizinkan MIDlet yang bersangkutan bekerja dalam domain tersebut. Pemilik Protection Domain ini memberikan gambaran terlebih dahulu tentang bagaimana peralatan harus mengidentifikasi sebuah MIDlet suite. Dengan identifikasi ini, maka pemilik protection domain berarti telah mengikat sebuah aplikasi dengan sebuah protection domain dalam bentuk aturan permisi yang diwakili oleh sertifikat yang memungkinkan akses ke API atau fungsi yang dilindungi. Jadi Sertifikat ini diciptakan dahulu oleh pemilik protection domain sebelum aplikasi dikirimkan. Mekanisme pengenalan oleh peralatan ini dikirimkan secara terpisah bersamaan dengan dikirimkannya aplikasi yang dimaksud. Di tempat yang dituju, aplikasi akan dibuka dengan memperhatikan sertifikat tersebut.

Trusted MIDlet Suite pada MIDP 2.0 menggunakan X.509 PKI yang berisi mekanisme untuk mengidentifikasi *trusted MIDlet suite* melalui mekanisme pengenalan dan verifikasi *signature* atau tanda tangan digital.

IV Model Otorisasi

Model otorisasi dari sebuah MIDlet suite diperoleh dengan cara menghubungkan beberapa elemen berikut :

- Sebuah *protection domain* yang berisi sekumpulan aturan perijinan yang diperbolehkan termasuk perijinan terhadap user
- Sekumpulan perijinan yang diminta oleh MIDlet suite dalam bentuk atribut *MIDlet-Permissions* dan *MIDlet-Permissions-Opt*
- Sekumpulan perijinan untuk masing-masing API atau fungsi pada peralatan yang merupakan gabungan dari semua perijinan yang didefinisikan oleh setiap API pada peralatan untuk melindungi fungsi tertentu.
- User yang menginginkan bantuan perijinan

Dalam MIDP, digunakan apa yang dinamakan dengan TLS dan SSL. TLS atau *Transport Layer Security* adalah protokol yang memungkinkan otentifikasi dan enkripsi data melalui kondisi jaringan yang tidak aman. Sedangkan SSL atau Secure Socket Layer adalah layer yang berada antara TCP/IP dan protokol jaringan di tingkat yang lebih tinggi seperti HTTP, SMTP, dan NNTP. Implementasi dari SSL dalam web browser bagi user bertujuan untuk menyediakan otentifikasi kriptografi, dan enkripsi berbasis session yang mudah digunakan.

IV.1 Transport Layer Security

Protokol TLS merupakan versi update dari protokol SSLv3 yang dibuat oleh Netscape. Salah satu kelebihan TLS adalah bahwa TLS beroperasi di atas socket TCP/IP, dan memiliki perilaku seperti halnya socket TCP/IP. Oleh karena itulah maka relatif lebih mudah membuat aplikasi jaringan menggunakan socket TLS. Salah satunya adalah aplikasi yang menggunakan HTTPS, yang merupakan protokol HTTP yang berjalan pada sebuah socket TLS atau SSL.

Protokol TLS diawali dengan *handshake*, yaitu kondisi dimana client dan server mencoba untuk menyetujui sebuah *cipher suite*, yaitu sekelompok algoritma kriptografi yang akan digunakannya untuk otentifikasi dan enkripsi session. Ketika client dan server telah berhasil melakukan negosiasi dalam bentuk *cipher suite*, maka mereka akan dapat mengotentifikasi satu sama lain serta membangkitkan *premaster secret*, yang merupakan dasar dari *session key*. *session key* ini digunakan untuk enkripsi dan dekripsi semua trafik data yang dikirim antara client dan server.

IV.2 Otentifikasi SSL dan TLS

Server akan melakukan proses otentifikasi dengan cara mengirimkan sertifikat yang berisi private key. Untuk memverifikasi sertifikat tersebut, client harus memiliki sertifikat root sebagai kunci yang digunakan untuk menandatangani sertifikat dari server.

V TLS pada MIDP 1.0

Dalam MIDP 1.0, tipe koneksi jaringan yang dikenal hanyalah koneksi HTTP. Berdasarkan Generic Connection Framework (Silakan baca buku Tip dan Trik J2ME Tingkat Lanjutan), para pembuat aplikasi MIDP bebas untuk menggunakan tipe koneksi tambahan termasuk diantaranya adalah penggunaan socket TLS maupun koneksi HTTPS. Untuk mendapatkan koneksi HTTPS dapat dilakukan seperti contoh berikut :

```
HttpConnection hc = (HttpConnection)
Connector.open("https://www.cert.org/");
```

Apabila anda mendapati pesan *ConnectionNotFoundException* ketika mencoba terhubung ke sebuah server, berarti dapat disimpulkan bahwa implementasi tidak mendukung HTTPS.

VI TLS pada MIDP 2.0

Dalam aplikasi MIDP 2.0 yang mampu mendukung HTTPS, maka data dapat dikirimkan melalui beberapa protokol berikut :

- TLS 1.0
- SSLv3
- Wireless Transport Layer Security (WTLS)
- WAP TLS Profile dan Tunneling Specification

Dalam MIDP 2.0 terdapat API baru yang menyediakan bagi MIDlet informasi tentang koneksi yang aman. Spesifikasi MIDP 2.0 menggunakan antarmuka *javax.microedition.io.HttpsConnection* yang merupakan ekstensi untuk *HttpConnection*. Antarmuka yang baru ini memiliki metode *getSecurityInfo()* yang akan mengembalikan info dari antarmuka baru yang lain yaitu *SecurityInfo*. Antarmuka *SecurityInfo* merupakan metode yang akan mengembalikan informasi tentang sebuah koneksi yang aman misalnya :

```
public String getProtocolName()
public String getProtocolVersion()
public String getCipherSuite()
public Certificate getServerCertificate()
```

Melalui metode-metode diatas, anda dapat mencari tahun nama dan versi dari protokol yang sedang digunakan apakah TLS, SSL, atau WTLS. Sedangkan metode *getCipherSuite()* akan mengembalikan nama dari cipher suite, Sebagai contoh, apabila koneksi referensi MIDP 2.0 dari emulator adalah <https://www.cert.org/> maka nama cipher suite yang akan diperoleh adalah:

```
TLS_RSA_WITH_RC4_128_SHA
```

Dari format diatas, maka algoritma yang digunakan untuk pertukaran kuncinya adalah *RSA*. Selain itu cipher stream yang digunakan adalah *RC4* dengan session key 128-bit yang digunakan untuk enkripsi data. Dan yang terakhir, fungsi *message digest* yang digunakan adalah *SHA-1*.

Sedangkan metode *getServerCertificate()* akan mengembalikan contoh dari *javax.microedition.pki.Certificate*, yang merupakan representasi dari sertifikat kriptografi. Antarmuka sertifikat memiliki metode yang dapat digunakan untuk menentukan subyek dari sertifikat, pihak yang mengeluarkannya, tipenya, dan berbagai informasi lainnya. Sebagai contoh, mendapatkan nama pada sertifikat server dapat dilakukan sebagai berikut :

```
String url = "https://www.cert.org/";
HttpsConnection hc = null;
hc = (HttpsConnection)Connector.open(url);
SecurityInfo info = hc.getSecurityInfo();
Certificate c = info.getServerCertificate();
String name = c.getIssuer();
```

VII Jenis Otentifikasi

Dalam komunikasi jaringan yang melibatkan dua buah peralatan, client akan mencari layanan untuk melakukan inisialisasi koneksi dan server akan menerimanya. Otentifikasi antara dua peralatan tersebut akan mengalami dua proses :

- *Otentifikasi Server* yaitu proses dimana server membuktikan identitasnya ke client. Implementasi HTTPS pada MIDP 2.0 mendukung secara langsung otentifikasi server menggunakan sertifikat X.509.
- *Otentifikasi Client* yaitu proses dimana client membuktikan identitasnya ke server.

Otentifikasi Server diperlukan karena user perlu diyakinkan bahwa transaksi adalah sah. Otentifikasi Server biasanya ditangani dengan HTTPS. Otentifikasi Client diperlukan karena transaksi memerlukan jaminan bahwa user melakukan transaksi dengan tepat.

VII.1 Otentifikasi Password pada MIDP

Sebuah teknik otentifikasi yang banyak menggunakan aplikasi web adalah otentifikasi password. Seorang user mengirimkan login dan password rahasia, kemudian server memeriksanya di database. Jika password benar, maka user diberi otentifikasi untuk memasuki server.

Skema password sederhana memiliki dua persoalan mendasar. Pertama adalah seringkali user tidak mampu menangani password ini dengan baik misalnya password mudah ditebak dan sebagainya. Persoalan kedua adalah proses otentifikasi password yang sederhana dimana password berbentuk teks sederhana dikirimkan antara client dan server. Ada dua solusi untuk masalah ini. Salah satunya adalah dengan mengirimkan informasi otentifikasi yang telah terenkripsi melalui jaringan.

Pada MIDP, sebuah skema otentifikasi password sederhana dapat dihasilkan dengan menambahkan nama dan password ke sebuah URL HTTP atau HTTPS seperti contoh berikut :

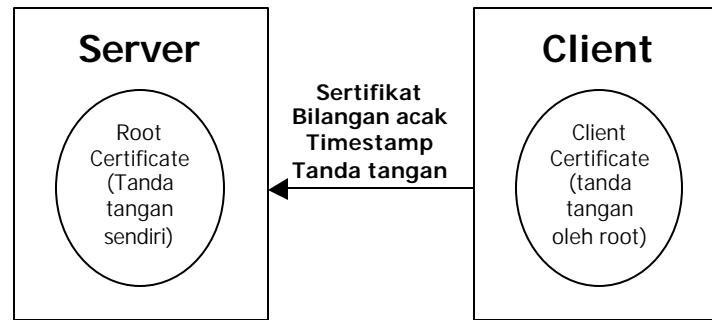
```
String user = "aditya";
String password = "hartanto76";
String base = "https://somehost.com/someservlet";

String url = base + "?user=" + user + "&password=" + password;
HttpsConnection hc = (HttpsConnection)Connector.open(url);
```

Karena masing-masing peralatan MIDP biasanya adalah milik pribadi seseorang, nama user dan password dapat disimpan didalam peralatan sehingga user tak perlu memasukkannya berulang-ulang setiap kali ingin menggunakan aplikasi.

VII.2 Otentifikasi Client Menggunakan Sertifikat

Jika anda ingin menambahkan informasi otentifikasi pada sebuah aplikasi client, maka anda dapat menggunakan *sertifikat X.509*. Prosesnya adalah dengan membuat sepasang *root key* dan sebuah *root certificate* yang merepresentasikan aplikasi tersebut. Selanjutnya, buatlah pasangan client key dan sertifikat yang ditandatangani oleh *root key*. Masing-masing copy dari software client dapat dipaket dengan sebuah *client key* dan sertifikat yang unik. Dan ketika client melakukan koneksi ke server, maka client dapat menggunakan *private key* yang dimilikinya untuk menandatangani atau mengenkripsi beberapa data, dan selanjutnya mengirimkan tanda tangan dan sertifikatnya ke server, seperti Gambar 10 berikut. Selanjutnya server akan menggunakan *root certificate* dari aplikasi untuk memverifikasi identitas dari client serta menggunakan *client certificate* untuk memverifikasi tanda tangannya.



Gambar 10. Proses pengiriman data dari Client ke Server

VIII Menandatangani sebuah MIDlet Suite

Dalam sebuah model keamanan midlet, maka didalamnya akan melibatkan MIDlet suite, tanda tangan, dan sertifikat public key. Sebagai tambahan, sertifikat root dapat dihadirkan dalam bentuk removable media seperti *SIM(WIM) card/modul USIM*. Implementasi harus mendukung sertifikat X.509 beserta algoritma yang bersesuaian.

Pemberi tanda tangan pada MIDlet suite bisa jadi adalah seorang developer atau beberapa entitas yang bertanggung jawab terhadap proses distribusi, supporti dan mungkin juga billing atas penggunaan MIDlet tersebut. Pemberi tanda tangan akan memerlukan sebuah sertifikat public key yang dapat divalidasi ke sebuah sertifikat root untuk perlindungan domain pada peralatan. Public key digunakan untuk memverifikasi tanda tangan pada MIDlet suite. *Public key* seperti telah disebutkan sebelumnya disediakan dalam bentuk sertifikat *RSA X.509* yang terdapat dalam descriptor dari aplikasi.

IX Membuat Sertifikat Tanda Tangan

Untuk membuat sebuah sertifikat tanda tangan digital, diperlukan beberapa proses sebagai berikut:

1. Pemberi tanda tangan perlu mengetahui aturan otorisasi untuk peralatan menghubungi certificate authority (CA) yang berwenang. Sebagai contoh, pemberi tanda tangan perlu mengirimkan distinguished name (DN)-nya beserta public key (yang umumnya, disatukan dengan paket di sebuah permintaan sertifikatt) ke sebuah certificate authority.
2. CA menciptakan sebuah sertifikat RSA X.509 (versi 3) dan mengembalikannya ke pemberi tanda tangan.
3. Jika digunakan lebih dari satu CA maka sebuah pemberi sertifikat tanda tangan dalam descriptor aplikasi harus berisi public key yang sama.

X Menambahkan Sertifikat ke dalam descriptor Aplikasi

Path sertifikat akan menyertakan sertifikat dari pemberi tanda tangan dan beberapa sertifikat yang diperlukan, namun akan mengabaikan sertifikat root. Sertifikat root akan ditemukan dalam peralatan yang digunakan. Masing-masing sertifikat pada path dikodekan dengan menggunakan base64 dan ditambahkan ke dalam descriptor aplikasi dengan format :

MIDlet-Certificate-<n>-<m>: <base64 pengkodean sebuah sertifikat>

Dimana :

<n>:= sebuah bilangan yang sama dengan 1 untuk path sertifikat yang pertama yang berada dalam path sertifikat dalam deskriptor atau 1 lebih besar daripada bilangan sebelumnya untuk path sertifikat tambahan.

<m>:= bilangan yang sama dengan 1 untuk semua sertifikat pemberi tanda tangan yang berada dalam path sertifikat atau 1 lebih besa daripada bilangan sebelumnya untuk bagian sertifikat lanjutannya.

XI Menciptakan tanda tangan digital RSA SHA-1 signature dari file JAR

Tanda tangan digital dari sebuah JAR diciptakan dengan menggunakan private key yang dimiliki pemberi tanda tangan sesuai dengan metode pengkodean EMSA-PKCS1-v1_5 dari standar PKCS #1 versi 2.0. Tanda tangan didasarkan pada metode pengkodean base64 yang diformat sebagai sebuah atribut MIDlet-Jar-RSA-SHA1 tunggal tanpa line break dan ditambahkan ke dalam descriptor aplikasi.

MIDlet-Jar-RSA-SHA1: <base64 encoding of Jar signature>

XII Memverifikasi Sertifikat dari Pemberi Tanda Tangan

Path Sertifikat berisi sertifikat dari pemberi tanda tangan dari descriptor aplikasi dan sertifikat lain yang diperlukan namun tidak termasuk didalamnya sertifikat root. Proses untuk memverifikasinya adalah :

1. Mendapatkan path sertifikat untuk pemberi tanda tangan sertifikat yang dapat diperoleh dari atribut descriptor aplikasi yaitu

MIDlet-Certificate-1-<m>

Dimana <m> dimulai dari 1 dan akan terus bertambah 1 sampai tidak ada atribut dengan nama yang diberikan. Nilai dari masing-masing atribut didasarkan pada sertifikat pengkodean base64 yang nantinya juga diperlukan untuk proses dekode dan parsing.

2. Memvalidasi path sertifikat menggunakan proses validasi path dasar yang dideskripsikan dalam RFC2459 menggunakan perlindungan domain sebagai sumber otoritatif dari sertifikat root sebuah perlindungan domain

XIII Proses verifikasi MIDlet Suite JAR

1. Dapatkan public key dari verifikasi sertifikat pemberi tanda tangan.
2. Dapatkan atribut MIDlet-Jar-RSA-SHA1 dari descriptor aplikasi.
3. Dekodekan nilai atribut dari hasil tanda tangan PKCS #1 menggunakan metode *base64*.
4. Gunakan public key dari pemberi tanda tangan, tanda tangan, dan SHA-1 digest dari JAR, untuk memverifikasi tanda tangan tersebut. Jika verifikasi tanda tangan gagal, maka akan terjadi penolakan terhadap descriptor aplikasi dan MIDlet suite.

Jika proses verifikasi sertifikat , tanda tangan, dan JAR berhasil maka isi dari MIDlet suite diketahui masih utuh dan identitas dari pemberi tanda tangan akan diketahui. Proses ini harus dilakukan selama instalasi.

Daftar Pustaka

1. Ady Wicaksono, Pemrograman Internet dan XML pada Ponsel dengan Midlet JAVA, Elex Media komputindo
2. wireless.java.sun.com
3. www.bouncycastle.org