Studi dan Analisis Implementasi Algoritma RC4 dengan Modifikasi Kunci Menggunakan Fungsi SHA-1

Fitriana Passa and 13508036[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]if18036@students.if.itb.ac.id

***Abstrak*—RC4 adalah salah satu algoritma kriptografi stream cipher yang masih sering digunakan pada protokol keamanan. Algoritma ini terdiri dari 3 langkah utama (inisialisasi array, KSA, dan PRGA) untuk mengenerate suatu keystream yang akan digunakan dalam proses enkripsi dan dekripsi. Proses enkripsi dan dekripsi pada RC4 dilakukan dengan operasi XOR antara keystream dengan plainteks. Algoritma RC4 menggunakan *padding* pada kunci masukan user sehingga terjadi pengulangan kunci berdasarkan periode tertentu. SHA-1 merupakan salah satu varian fungsi hash yang sering dipakai dalam pemampatan pesan satu arah. SHA-1 digunakan untuk menghasilkan kunci asli dari kunci masukan user yang akan digunakan dalam proses enkripsi/dekripsi pesan pada bagian KSA agar kunci yang ada tidak berulang. Implementasi program dilakukan dalam lingkungan bahasa C# dan menghasilkan 2 program RC4_hash dan program RC4 biasa. Hasil pengujian menunjukkan bahwa implementasi algoritma RC4 berhasil dilakukan dengan baik dan benar dalam program. Sementara itu, penggunaan fungsi SHA-1 dalam program terbukti dapat menghilangkan perulangan kunci masukan user dengan cara mengganti kunci user dengan kunci yang telah dimodifikasi menggunakan SHA-1. Penggunaan SHA-1 dalam RC4 membuat setiap kunci yang berbeda akan menghasilkan keluaran yang berbeda. Hal ini tidak terjadi pada RC4 biasa dimana kunci tertentu yang di*padding* akan menghasilkan hasil enkripsi yang sama dengan kunci dasar yang tidak di*padding*.**

***Index Terms*—RC4, SHA-1, KSA, PRGA.**

I. PENDAHULUAN

Algoritma kriptografi klasik yang dikenal dalam bidang kriptografi sudah jarang digunakan karena tingkat keamanannya yang rendah. Sebagai gantinya, diciptakan berbagai algoritma baru yang lebih mangkus untuk mengatasi kelemahan-kelemahan yang ada. Salah satu algoritma kriptografi yang digunakan sekarang adalah RC4. RC4 banyak digunakan protokol-protokol populer seperti SSL dan WEP.

Algoritma RC4 termasuk ke dalam algoritma stream cipher. Sampai saat ini tidak ada yang dapat memecahkan RC4 sehingga dapat dikatakan sangat kuat. Akan tetapi, *padding* yang digunakan pada algoritma ini dapat menyebabkan kemungkinan nilai-nilai di dalam larik S ada yang sama. RC4 juga mudah diserang dengan known-plaintext attack, dengan cara meng-XOR-kan dua set byte cipher teks.

SHA adalah fungsi hash satu arah yang dibuat oleh NIST. Salah satu varian SHA yang sering dipakai adalah SHA-1. SHA-1 sangat sulit dipecahkan dengan cara sederhana. Secara umum, isi pesan yang berbeda akan menghasilkan hasil SHA-1 yang berbeda. Dengan begitu, kemungkinan yang sangat kecil untuk terjadinya kolisi pada SHA-1 dapat dimanfaatkan untuk menutup kekurangan yang dimiliki pada algoritma RC4 dalam hal *padding* kunci dengan cara biasa.

Kombinasi algoritma RC4 yang terbukti sangat kuat dan SHA-1 yang menghasilkan keluaran berbeda pada setiap input kunci yang dimasukkan diharapkan akan mengurangi kelemahan algoritma RC4 dalam enkripsi data.

II. LANDASAN TEORI

1. RC4

Algoritma RC4 merupakan salah satu algoritma kriptografi yang termasuk ke dalam kategori *cipher* aliran (*stream cipher*). Algoritma ini dibuat oleh Ron Rivest (1987) dari Laboratorium *RSA.* RC4 banyak digunakan pada sistem keamanan seperti protokol SSL, WEP, dan WPA untuk komunikasi nirkabel.

RC4 terdiri dari 3 langkah utama:

a. Inisialisasi array S : $S_0 = 0$, $S_1 = 1$, $S_2 = 2$, ... , $S_{255} = 255$ (inisialisasi array S dengan nilai yang sesuai dengan nilai indeks maing-masing elemen)
b. Permutasi nilai-nilai dalam larik dengan Key Scheduling Algorithm (KSA)
c. Bangkitkan aliran kunci dan lakukan enkripsi menggunakan Pseudo Random Generation Algorithm (PRGA)

Jika panjang kunci U < 256, lakukan *padding* sehingga panjang kunci menjadi 256 byte.

KSA

Key Scheduling Algorithm atau KSA digunakan untuk menginisialisasi permutasi dalam array S. Pertama, array S diinisialisasi sesuai dengan penjelasan pada bagian sebelumnya. S kemudian diproses sebanyak 256 kali

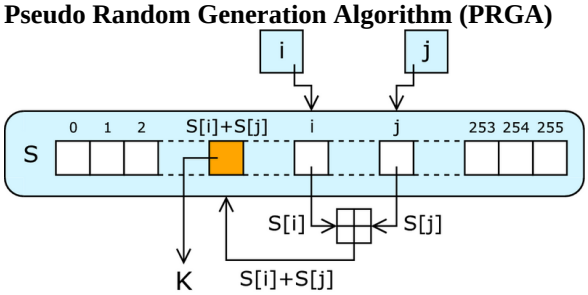
iterasi untuk mendapatkan nilai S akhir pada setiap elemen array.

pseudocodenya adalah sebagai berikut:

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

dalam makalah ini, keylength yang digunakan adalah sepanjang 256 byte.

Pseudo Random Generation Algorithm (PRGA)



Seperti pada gambar di atas, PRGA mengubah state larik yang ada dan menghasilkan sebuah byte pada keystream. Pada setiap iterasi, PRGA akan menambahkan nilai i, menukar nilai S[i] dan S[j], mengeluarkan output elemen array S pada posisi (S[i] + S[j]) mod 256. Keluaran inilah yang disebut keystream dan akan diXORkan dengan plainteks untuk menghasilkan cipherteks.

pseudocodenya adalah sebagai berikut:

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

Faktor utama yang menyebabkan RC4 banyak digunakan pada berbagai aplikasi adalah karena kecepatan dan kesederhanaannya. Algoritma ini tidak menggunakan LFSR (*Linear Feedback Shift Register*) yang tidak efisien digunakan pada software. Dengan kata lain, implementasi yang efisien dalam software dan hardware menggunakan algoritma ini mudah untuk dikembangkan.

Karena RC4 merupakan algoritma simetric key, proses dekripsi yang terjadi sama seperti proses enkripsi. Alasannya sederhananya adalah karena pada sebuah kunci masukan user, kunci yang digenerate akan selalu sama.

(A xor B) xor B = A

yang berarti bahwa jika kita menggunakan kunci yang sama untuk mengenkripsi sebuah file dan memasukkan hasil file terenkripsi dalam proses enkripsi kembali, maka output file yang dihasilkan sama dengan plainteks aslinya.

2. SHA-1

SHA adalah fungsi *hash* satu-arah yang dibuat oleh *NIST* dan digunakan bersama *DSS* (*Digital Signature Standard*). Algoritma *SHA* menerima masukan berupa pesan dengan ukuran maksimum 2^{64} bit (2.147.483.648 *gigabyte*) dan menghasilkan *message digest* yang panjangnya 160 bit, lebih panjang dari *message digest* yang dihasilkan oleh *MD5*. Ada 6 varian dalam SHA, salah satunya adalah SHA1.

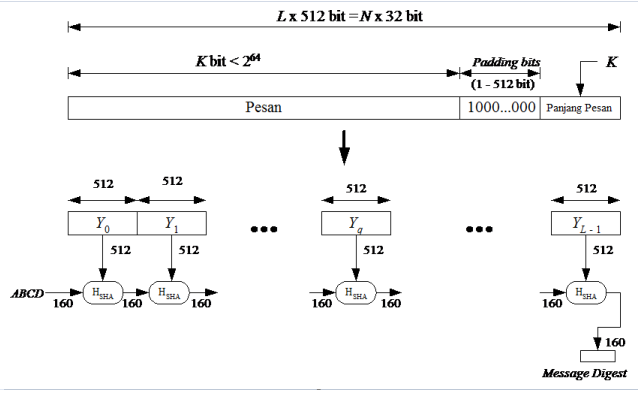
Langkah-langkah pembuatan message digest pada SHA-1:

a. Penambahan bit pengganjal (*padding* bits)
b. Penambahan nilai panjang pesan semula
c. Inisialisasi penyangga (buffer) MD
d. Pengolahan pesan dalam blok berukuran 512 bit.

Proses H_{SHA} terdiri dari 80 putaran. masing-masing putaran menggunakan bilangan penambah K_p :

Putaran $0 \le t \le 19$ K_t = 5A827999
Putaran $20 \le t \le 39$ K_t = 6ED9EBA1
Putaran $40 \le t \le 59$ K_t = 8F1BBCDC
Putaran $60 \le t \le 79$ K_t = CA62C1D6

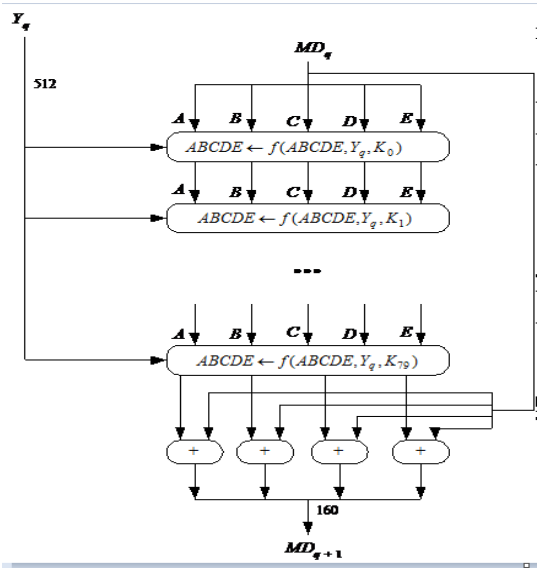
Pembuatan message digest dengan SHA-1 dapat ditunjukkan dengan skema sebagai berikut:



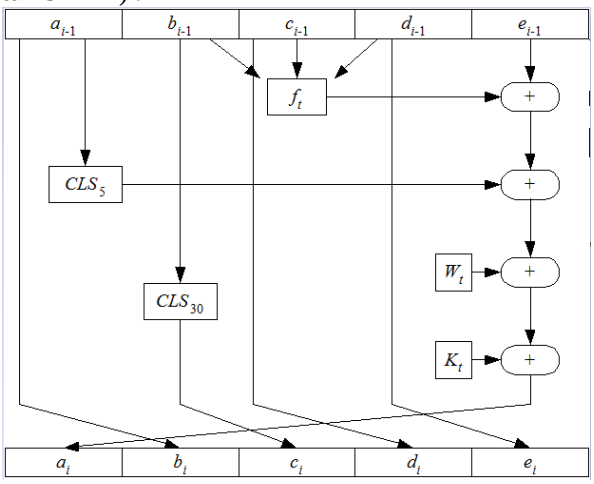
SHA membutuhkan 5 buah penyangga yang masing-masing mempunyai panjang 32 bit (total : 160bit). Setiap penyangga diinisialisasi dengan nilai sebagai berikut (HEX):

A = 67452301
B = EFCDAB89
C = 98BADCFE
D = 10325476
E = C3D2E1F0

Pengolahan blok 512-bit:



Operasi dasar pada setiap putaran SHA-1 (80x putaran untuk SHA-1) :



Nilai W_1 sampai W_{16} berasal dari 16 *word* pada blok yang sedang diproses, sedangkan nilai W_t berikutnya didapatkan dari persamaan

$$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$$

Meskipun telah ditemukan *collision* pada hasil message digestnya dengan teoretical attack, secara umum SHA-1 masih sering digunakan karena sifatnya yang relatif aman untuk penggunaan biasa. Penggunaannya biasanya dikombinasikan dengan digital signature untuk memverifikasi kepemilikan berkas.

III. RANCANGAN ALGORITMA

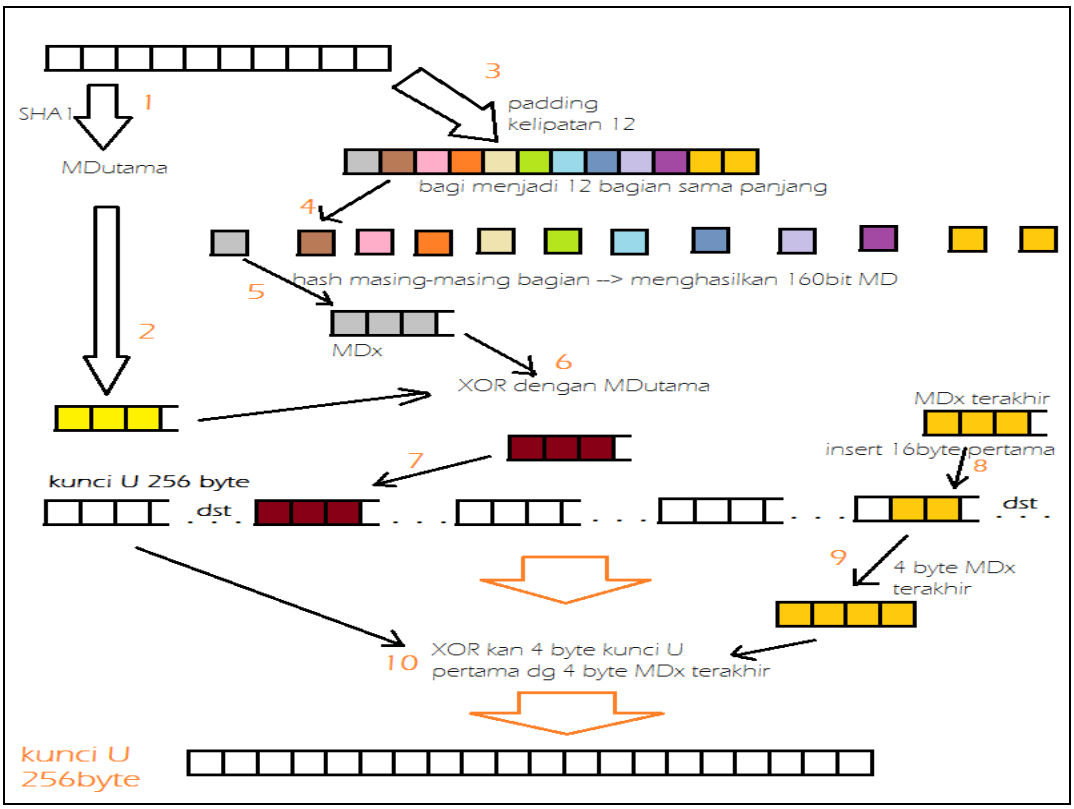
RC4 yang diimplementasikan pada makalah ini menggunakan SHA-1 untuk modifikasi kunci semula. Pada struktur kunci sebelumnya, kunci yang memiliki panjang kurang dari panjang plainteks akan di*padding* dengan kunci yang sama sehingga kunci yang ada menjadi berulang. Ada beberapa alternatif panjang kunci yang dapat digunakan, biasanya diantara 40 – 256bit. Pada makalah ini, panjang kunci yang digunakan adalah 256 byte, seperti yang dijelaskan pada slide kuliah Kriptografi.

1. Pembangkitan kunci

Kunci masukan user bukan merupakan kunci sebenarnya yang akan digunakan dalam algoritma RC4. Kunci yang ada dibangkitkan oleh program berdasarkan kunci masukan user. Algoritma pembangkitan kunci RC4-hash ini adalah sebagai berikut:

1. Kunci user di hash dengan SHA1 menghasilkan message digest utama (MD_{utama}). Salin isi MD_{utama} ke dalam *byte array* kunci akhir U dari indeks ke-0 sampai indeks ke-19 (160bit pertama).
2. Jika kunci masukan user bukan merupakan kelipatan 12, *padding* kunci user sebanyak N-byte kekurangannya dengan N-byte pertama kunci user (kunci user menjadi berulang). *Padding* ini menghasilkan KunciUserBaru
3. Bagi KunciUserBaru menjadi 12 bagian sama panjang. Untuk 11 bagian pertama:
   - Hash bagian kunci yang ada, menghasilkan MD_x.
   - MD_x diXORkan dengan message digest yang diperoleh sebelumnya, yaitu $MD_{utama..}$ Hasil message digest ini ditambahkan ke dalam kunci akhir U mulai dari indeks terakhir yang telah diisi pada tahap sebelumnya sebanyak 20 byte (160bit, sesuai dengan hasil SHA-1). Hasil message digest ini menjadi MD_{utama}.
   - Lakukan hal di atas hingga sebanyak 11 kali dan kunci U yang telah diisi sebanyak 20 x 12 byte = 240 byte.
   - Untuk pengisian kunci U sebanyak 16 byte terakhir, dilakukan dengan menggunakan hasil proses seperti pada tahap sebelumnya. Bedanya, bukan 20 byte hasil message digest yang disalin, tetapi hanya 16 byte pertama message digest.
   - 4 Byte sisa dari message digest yang terakhir dihasilkan di XOR kan dengan 4 byte pertama pada kunci U.
   - Kunci U sepanjang 256 byte telah dapat digunakan untuk proses selanjutnya.

ilustrasi:



2. Enkripsi / dekripsi

Proses enkripsi dan dekripsi RC4 dilakukan sama seperti yang telah dijelaskan dikelas, yaitu dengan menggunakan Pseudo Random Generation Algorithm atau yang dikenal dengan PRGA untuk menghasilkan cipherteks yang memiliki panjang yang sama dengan panjang plainteks. Sebelum melakukan enkripsi, dilakukan terlebih dahulu inisialisasi array S yang akan digunakan dalam key scheduling algorithm. Setelah inisialisasi, dilakukan Key Scheduling (KSA) untuk mempermutasikan nilai-nilai pada array S yang telah diinisialisasi. Setelah itu, dilakukan proses utama dalam PRGA untuk menghasilkan cipher teks dengan langkah yang sama yang telah tertulis pada bagian dasar teori.

IV. Implementasi dan pengujian

Implementasi RC4 yang dibahas pada makalah ini dilakukan menggunakan kakas Microsoft Visual Studio 2010 dengan bahasa pemrograman C#. Untuk dapat membandingkan hasil pengujian program serta menganalisis pengaruh penggunaan fungsi SHA terhadap algoritma RC4, dibuat dua program dalam bahasa C#. Program pertama diberi nama RC4hash yang mengimplementasikan algoritma RC4 dengan kombinasi fungsi SHA-1. Sedangkan program kedua diberi nama RC4 yang mengimplementasikan algoritma RC4 tanpa fungsi SHA-1.

Implementasi mencakup fitur enkripsi dan dekripsi pada file dengan berbagai ekstensi maupun dari inputan user secara manual. Kode program dan tampilan ada pada bagian lampiran (untuk RC4hash saja, karena perbedaannya hanya pada bagian KSA).

Pengujian program dilakukan pada berbagai tipe file dalam berbagai kasus uji. Dalam makalah ini, pengujian dilakukan pada 5 tipe file yang dipilih secara acak: pdf, doc, xls, jpg, dan mp3. Proses pengujian pada masing-masing tipe fiel terdiri dari langkah-langkah sebagai berikut:

1. Dekripsi menggunakan kunci asli
2. Dekripsi menggunakan kunci palsu / kunci yang telah dimodifikasi
3. Pengubahan beberapa byte cipherteks
4. Penghapusan sebagian isi cipherteks.

Hasil pengujian menunjukkan bahwa implementasi algoritma RC4 yang dibuat dapat berjalan dengan baik. Enkripsi dan dekripsi dapat dilakukan dengan benar. Tampilan antarmuka dan hasil enkripsi dekripsi setiap file uji dapat dilihat pada bagian lampiran.

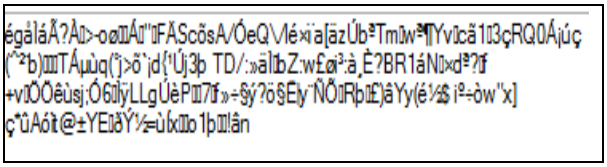
Berdasarkan pengujian yang telah dilakukan, hasil pembukaan dokumen setelah dilakukan proses dekripsi dapat dilihat dalam tabel berikut:

tipe file	kunci asli	kunci palsu	Pengubahan cipherteks	Penghapusan cipherteks
pdf	berhasil	file corrupt	file corrupt	file corrupt
jpg	berhasil	gambar tidak valid	gambar tidak valid	gambar tidak valid
xls	berhasil	file corrupt	file corrupt	file corrupt

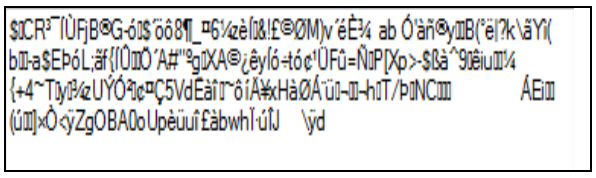
doc	berhasil	file corrupt	file corrupt	file corrupt
mp3	berhasil	tidak dapat dibuka	tidak dapat dibuka	tidak dapat dibuka

Untuk membandingkan hasil yang didapatkan pada algoritma RC4 biasa dan RC4_hash, digunakan file bertipe .txt agar hasil lebih mudah terlihat dari plainteks ke cipherteks hasil. Pada pengujian ini, digunakan beberapa kasus uji, yaitu sebagai berikut:

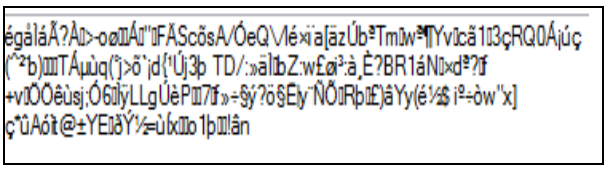
1. file teks : hahahahahahaha .... dst (256 karakter)
   kunci : popopo
   hasil:
   - RC4 biasa



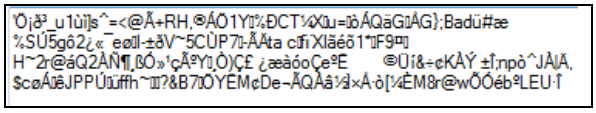
   - RC4 hash



2. file teks : hahahahahahaha .... dst (256 karakter)
   kunci : popopopopo
   hasil:
   - RC4 biasa

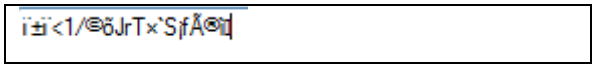


   - RC4 hash



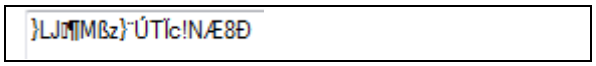
3. file teks : aaaaaaaaaaaaaaaaaaa
   kunci : aaaaa
   hasil:
   - RC4 biasa

   - RC4 hash

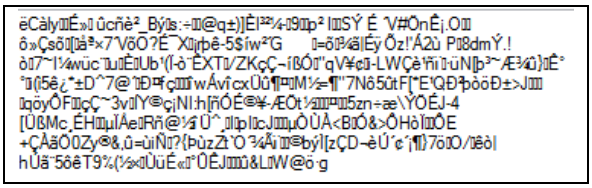


4. file teks : aaaaaaaaaaaaaaaaaaa
   kunci : aaaaaaaaaa
   hasil:
   - RC4 biasa

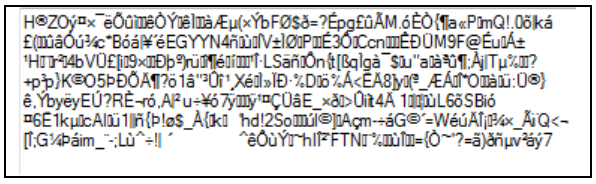
   - RC4 hash



5. file teks : RC4 is a stream cipher (another example is A5 GSM cell phones), which means that data usually is encrypted and decrypted one byte at a time. Other ciphers, such as RC2, 3DES, DES, and AES, encrypt and decrypt data in blocks, often 64-bit or 128-bit blocks.RC4 is a stream cipher (another example is A5 GSM cell phones), which means that data usually is encrypted and decrypted one byte at a time. Other ciphers, such as RC2, 3DES, DES, and AES, encrypt and decrypt data in blocks, often 64-bit or 128-bit blocks.
   kunci : RC4 is a stream cipher (another example is A5 GSM cell phones), which means that data usually is encrypted and decrypted one byte at a time. Other ciphers, such as RC2, 3DES, DES, and AES, encrypt and decrypt data in blocks, often 64-bit or 128-bit blocks.
   hasil:
   - RC4 biasa



   - RC4 hash



Pada pengujian di atas, hal yang diperhatikan mencoba membandingkan penggunaan fungsi hash dalam RC4 untuk mencegah pengulangan kunci masukan user menjadi bentuk yang tidak berulang dibandingkan dengan algoritma RC4 biasa yang menggunakan *padding* kunci.

V. ANALISIS

Berdasarkan implementasi program RC4_SHA, algoritma yang ada telah teruji dapat digunakan untuk melakukan proses enkripsi/dekripsi pesan. Pesan yang didekripsi menggunakan kunci yang asli dapat dibuka kembali oleh kakas yang sesuai. Sedangkan cipherteks yang ada secara total merusak struktur file sehingga tidak dapat dibuka kembali.

Pada pengujian algoritma RC4 menggunakan fungsi hash di atas, penggunaan SHA-1 dalam enkripsi pesan pada RC4 dapat membantu mencegah *padding* yang terjadi pada kunci masukan user. *Padding* kunci ini berpengaruh terhadap hasil enkripsi yang dilakukan oleh RC4. Contoh kasus 1-2 dan contoh kasus 3-4 pada bagian

pengujian menunjukkan bahwa untuk file teks yang sama dan kunci yang berbeda akan menghasilkan hasil enkripsi pesan yang sama. Sementara itu, penggunaan SHA-1 pada RC4 menyebabkan pada contoh kasus 1-2 dan 3-4 cipher teks yang dihasilkan tidak sama. Hal ini menunjukkan bahwa penggunaan message digest dari SHA-1 untuk mencegah berulangnya kunci (misal : popo dan popopopo yang dianggap sama) dapat digunakan.

Secara umum, cipherteks yang dihasilkan oleh RC4 biasa maupun RC4 hash sulit dibaca dengan mata biasa. Penggunaan SHA dalam RC4 hanya untuk mengurangi kemungkinan terjadinya kesamaan cipherteks karena duplikasi yang ada pada kunci. Sedangkan tingkat keamanan kunci itu sendiri pada dasarnya tidak berubah karena kunci masukan userlah yang dijadikan awal dari proses modifikasi kunci.

Meskipun sering digunakan dalam protokol keamanan, algoritma enkripsi yang digunakan pada RC4 tetap mempunyai 2 kelemahan:

a. RC4 menghasilkan keystream yang sama untuk setiap key yang sama yang dimasukkan oleh user.
   Hal ini diperburuk dengan kondisi bahwa 2 key yang menghasilkan nilai U yang sama akan menghasilkan nilai keystream yang sama. Dengan demikian, penggunaan SHA sangat terasa bermanfaat dalam menangani kasus ini. Pada dasarnya keystream hanya berubah bentuk menjadi karakter-karakter lain hasil dari message digest oleh fungsi SHA.
b. RC4 mengenkripsi data 1 byte dalam 1 waktu
   Hal ini membuat RC4 lebih mudah dipecahkan daripada block cipher yang notabene menggunakan satuan bit dalam operasinya, sehingga lebih sukar untuk dipecahkan. Salah satu cara pemecahan RC4 adalah menggunakan dengan serangan known plaintext.

VI. KESIMPULAN

Kombinasi algoritma RC4 yang terkenal ampuh dan fungsi hash SHA-1 yang secara teori selalu menghasilkan keluaran yang berbeda untuk setiap masukan yang berbeda memberikan alternatif pengamanan yang lebih dalam mengatasi kekurangan yang ditemukan dalam RC4 biasa, khususnya masalah redundansi kunci yang terjadi akibat adanya *padding*.

REFERENSI

[1] Munir, Rinaldi. RC4 dan A5. *Slide Kuliah IF3058 Kriptografi*.
[2] Munir, Rinaldi. SHA. *Slide Kuliah IF3058 Kriptografi*.
[3] Michael Howard. http://archive.devx.com/security/bestdefense/2001/mh0201/mh0201-1.asp. Tanggal akses: 8 Mei 2011 pukul 17.00 WIB.
[4] ------ . http://www.symatech.net/rc4. Tanggal akses: 8 Mei 2011 pukul 17.00 WIB.
[5] ------ . http://www.fyneworks.com/encryption/RC4-Encryption/. Tanggal akses: 8 Mei 2011 pukul 17.00 WIB.
[6] ------ . http://netlab18.cis.nctu.edu.tw/html/wlan_course/powerpoint/RC4.pdf . Tanggal akses: 8 Mei 2011 pukul 17.00 WIB.
[7] ------ . http://security-freak.net/encryption/encryption-rc4.html. Tanggal akses: 8 Mei 2011 pukul 17.00 WIB.

PERNYATAAN

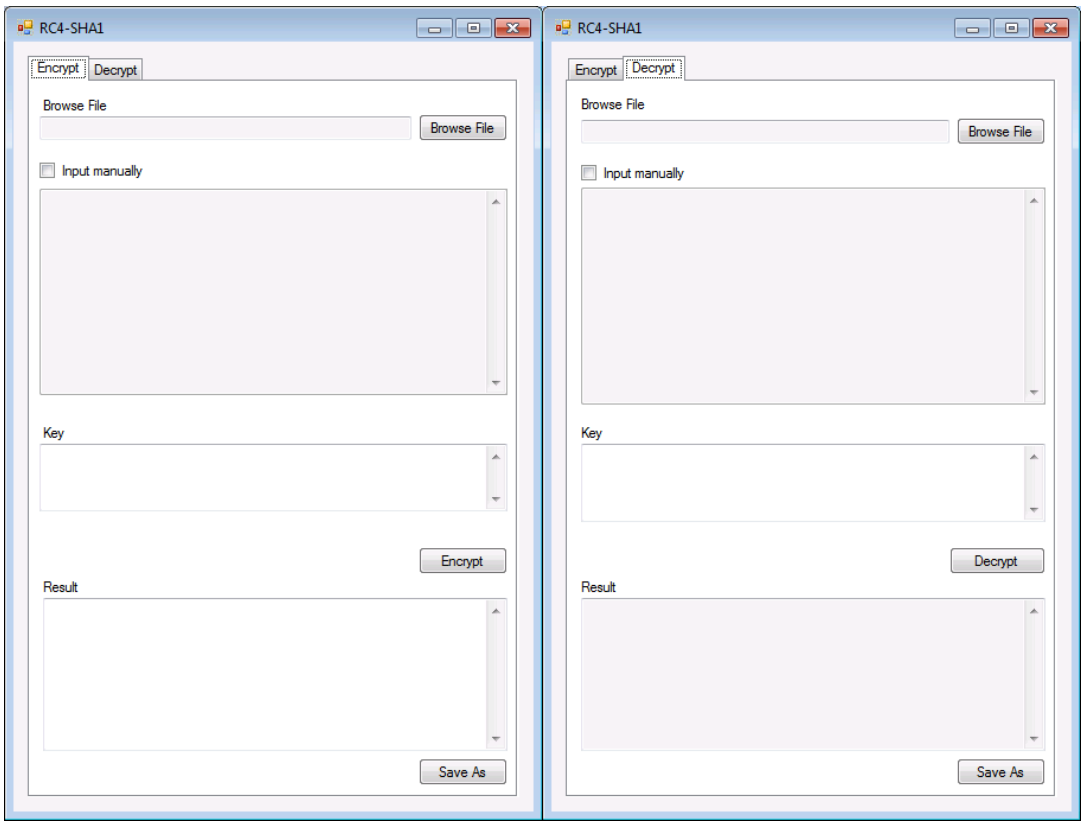
Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Mei 2011

Fitriana Passa
NIM 13508036

LAMPIRAN

A. Tampilan Antarmuka



B. Kode Program

1. Kelas RC4.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RC4Hash.algorithm
{
    class RC4
    {
        static byte[] S;
        static byte[] U; // this is the key


        /// <summary>
        /// initialize var S for permutation and U for key
        /// </summary>
        public static void initVar(byte[] inputkey)
        {
            //init S
            S = new byte[256];
            for (int i = 0; i < 256; ++i)
            {
                S[i] = (byte)i;
            }

            //process U for generating true key
            U = new byte[256];
            U = generateTrueKey(inputkey);
        }

        /// <summary>
        /// get N byte from source array
        /// </summary>
        /// <param name="source"></param>
```

```
        /// <param name="start"></param>
        /// <param name="length"></param>
        /// <returns></returns>
        public static byte[] getNByte(byte[] source, int start, int length)
        {
            byte[] retval = new byte[length];

            for (int i = 0; i < length; ++i)
                retval[i] = source[i + start];

            return retval;
        }


        /// <summary>
        /// XOR prosedur untuk xor 2 array of N byte
        /// </summary>
        /// <param name="key1">key1 untuk xor</param>
        /// <param name="key2">key2 untuk xor</param>
        /// <param name="keylength">keylength menyatakan panjang kunci dalam byte</param>
        /// <returns>array of 8 byte from xor-ing key1 and key2</returns>
        public static byte[] XorNBitKey(byte[] key1, byte[] key2, int keylength)
        {
            byte[] result = new byte[keylength];
            for (int i = 0; i < keylength; ++i)
            {
                result[i] = (byte)((int)key1[i] ^ (int)key2[i]);
            }
            return result;
        }


        /// <summary>
        /// padding key into key that can be divided by 12
        /// </summary>
        /// <param name="key"></param>
        /// <returns></returns>
        public static byte[] paddingKey(byte[] key)
        {
            if (key.Length % 12 != 0)
            {
                int padding_length = 12 - (key.Length % 12);

                byte[] temp = new byte[key.Length + padding_length];
                Array.Copy(key, temp, key.Length);

                for (int i = 0; i < padding_length; ++i)
                {
                    temp[i + key.Length] = temp[i];
                }

                return temp;
            }
            else
            {
                return key;
            }
        }


        /// <summary>
        /// generate true key from user's key
        /// </summary>
        /// <param name="inputkey"></param>
        /// <returns></returns>
        public static byte[] generateTrueKey(byte[] inputkey)
        {
            byte[] retval = new byte[256];

            //get 160bits message digest from input message (20bytes);

            SHAalgorithm.initMyHash();
            byte[] dum = SHAalgorithm.insertPaddingAndMsgLength(inputkey);
            byte[] firstMD = SHAalgorithm.getMessageDigest(dum);
            Array.Copy(firstMD, retval, firstMD.Length);
```

```
            byte[] temp_key = paddingKey(inputkey);
            int len_eachstep = temp_key.Length / 12;

            //11 MD pertama
            for (int i = 0; i < 11; ++i)
            {
                byte[] keyForThisStep = getNByte(temp_key, i * len_eachstep, len_eachstep);

                SHAalgorithm.initMyHash();
                dum = SHAalgorithm.insertPaddingAndMsgLength(keyForThisStep);
                byte[] tempMD = SHAalgorithm.getMessageDigest(dum);

                tempMD = XorNBitKey(firstMD, tempMD, 20);
                Array.Copy(tempMD, 0, retval, (i + 1) * 20, 20);
                firstMD = tempMD;

            }

            byte[] lastKey = getNByte(temp_key, 11 * len_eachstep, len_eachstep);

            SHAalgorithm.initMyHash();
            dum = SHAalgorithm.insertPaddingAndMsgLength(lastKey);
            byte[] lastMD = SHAalgorithm.getMessageDigest(dum);

            lastMD = XorNBitKey(firstMD, lastMD, 20);
            Array.Copy(lastMD, 0, retval, 12 * 20, 16);

            //assign 4byte- rest of lastMD into 4-first retval --> XOR
            retval[0] = (byte)(retval[0] ^ lastMD[16]);
            retval[1] = (byte)(retval[1] ^ lastMD[17]);
            retval[2] = (byte)(retval[2] ^ lastMD[18]);
            retval[3] = (byte)(retval[3] ^ lastMD[19]);


            return retval;
        }


        /// <summary>
        /// key scheduling algorithm (KSA)
        /// </summary>
        /// <returns></returns>
        public static void keyScheduling(byte[] key)
        {
            int j = 0;
            for (int i = 0; i < 256; ++i)
            {
                j = (j + S[i] + U[i]) % 256;
                //j = (j + S[i] + key[i % key.Length]) % 256;

                //swap
                byte temp = S[i];
                S[i] = S[j];
                S[j] = temp;
            }
        }

        /// <summary>
        /// pseudo random generation algorithm
        /// </summary>
        /// <param name="plaintext"></param>
        /// <returns></returns>
        public static byte[] PRGA(byte[] plaintext)
        {
                int i = 0;
                int j = 0;
            byte[] cipher = new byte[plaintext.Length];

                for (int idx = 0; idx < plaintext.Length; ++idx)
            {
                i = (i + 1) % 256;
                j = (j + S[i]) % 256;

                //swap value of S[i] and S[j]
                byte temp = S[i];
                S[i] = S[j];
```

```
                S[j] = temp;

                int t = (S[i] + S[j]) % 256;
                byte K = S[t];  /* keystream */
                cipher[idx] = (byte)(K ^ plaintext[idx]);
            }

            return cipher;
        }

        /// <summary>
        /// main procedure for decryption
        /// </summary>
        /// <param name="message"></param>
        /// <param name="inputkey"></param>
        /// <returns></returns>
        public static byte[] decrypt_RC4(byte[] message, byte[] inputkey)
        {
            byte[] retval;

            initVar(inputkey);
            keyScheduling(inputkey);
            retval = PRGA(message);
            return retval;
        }


        /// <summary>
        /// main procedure for encryption
        /// </summary>
        /// <param name="message"></param>
        /// <param name="inputkey"></param>
        /// <returns></returns>
        public static byte[] encrypt_RC4(byte[] message, byte[] inputkey)
        {
            byte[] retval;

            initVar(inputkey);
            keyScheduling(inputkey);
            retval = PRGA(message);
            return retval;
        }

    }
}
```

2. Kelas SHA_algorithm.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RC4Hash.algorithm
{
    class SHAalgorithm
    {
        //atribute
        private static int blocksize = 64; // in byte
        private static int sizemsgLength = 8; // 8 byte = 64 bit
        public static UInt32[] H;
        public static UInt32[] K;
        public static byte[] ABCDE;
        public static byte[] H1234;

        public static void initMyHash()
        {
            //init vars
            H = new UInt32[5];
            K = new UInt32[4];
            H[0] = 1732584193;
```

```
            H[1] = 4023233417;
            H[2] = 2562383102;
            H[3] = 271733878;
            H[4] = 3285377520;
            K[0] = 1518500249;
            K[1] = 1859775393;
            K[2] = 2400959708;
            K[3] = 3395469782;

            //create array of byte from ABCDE
            ABCDE = new byte[20]; // 20 byte = 20 * 8 = 160bit
            H1234 = new byte[16]; // 16byte = 16 * 8 = 128bit
            byte[] dump;

            //create array from element 0-3 from buffer and H
            for (int i = 0; i < 4; ++i)
            {
                dump = breakLongIntoArrByte(H[i], 4);
                byte[] dump2 = breakLongIntoArrByte(K[i], 4);
                for (int j = 0; j < 4; ++j)
                {
                    ABCDE[i * 4 + j] = dump[j];
                    H1234[i * 4 + j] = dump2[j];
                }
            }

            //create last 4 array from buffer
            dump = breakLongIntoArrByte(H[4], 4);
            for (int j = 0; j < 4; ++j)
            {
                ABCDE[16 + j] = dump[j];
            }
        }


        //METHODS
        /// <summary>
        /// prosedur untuk mendapatkan N byte message dari parameter message
        /// </summary>
        /// <param name="bytes">array of byte yang mau diambil</param>
        /// <param name="offset">indeks mulai message yang mau diambil</param>
        /// <param name="length">length menyatakan panjang dalam byte</param>
        /// <returns>N byte message dari parameter dimulai dari posisi ke offset</returns>
        public static byte[] getNbyteKey(byte[] bytes, long offset, long length)
        {
            byte[] result = new byte[length];
            for (long i = 0; i < length; ++i)
            {
                result[i] = bytes[offset + i];
            }
            return result;
        }

        //METHODS
        /// <summary>
        /// prosedur untuk mendapatkan N byte message dari parameter message
        /// </summary>
        /// <param name="bytes">array of byte yang mau diambil</param>
        /// <param name="offset">indeks mulai message yang mau diambil</param>
        /// <param name="length">length menyatakan panjang dalam byte</param>
        /// <returns>N byte message dari parameter dimulai dari posisi ke offset</returns>
        public static UInt32 getUInt32FromNByte(byte[] bytes, long offset, long length)
        {
            UInt32 result = bytes[offset];
            for (long i = 1; i < length; ++i)
            {
                result = (result << 8) + bytes[offset + i];
            }
            return result;
        }

        /// <summary>
        /// XOR prosedur untuk xor 2 array of 8 byte
        /// </summary>
        /// <param name="key1">key1 untuk xor</param>
        /// <param name="key2">key2 untuk xor</param>
```

```
        /// <param name="keylength">keylength menyatakan panjang kunci dalam byte</param>
        /// <returns>array of 8 byte from xor-ing key1 and key2</returns>
        public static byte[] XorNBitKey(byte[] key1, byte[] key2, long keylength)
        {
            byte[] result = new byte[keylength];
            for (long i = 0; i < keylength; ++i)
            {
                result[i] = (byte)((int)key1[i] ^ (int)key2[i]);
            }
            return result;
        }


        /// <summary>
        /// breaking a long (64bit) var into 8 byte array (@8bit)
        /// </summary>
        /// <param name="source"></param>
        /// <returns></returns>
        public static byte[] breakLongIntoArrByte(long source, int CounterArray)
        {
            long filesize = source;
            byte[] retval = new byte[CounterArray];

            for (int i = 0; i < CounterArray; ++i)
            {
                retval[CounterArray - i - 1] = (byte)(filesize & 255);
                filesize = filesize >> 8;
            }
            return retval;
        }


        /// <summary>
        /// rotate bits left
        /// </summary>
        /// <param name="source"></param>
        /// <param name="shifting"></param>
        /// <returns></returns>
        public static UInt32 ROTL(UInt32 source, byte shifting)
        {
            return (UInt32)(((source) << (shifting)) | ((source) >> (32 - (shifting))));
        }


        /// <summary>
        /// insert padding and message length into real_message
        /// </summary>
        /// <param name="message">true message</param>
        /// <returns></returns>
        public static byte[] insertPaddingAndMsgLength(byte[] message)
        {
            long len = message.LongLength;
            byte[] message_result;
            byte[] filelen = new byte[sizemsgLength];                           //normal file size
            filelen = breakLongIntoArrByte(message.LongLength * 8, sizemsgLength);

            int paddinglength = blocksize - (int)(len % blocksize);

            if (paddinglength < 9)
            {
                paddinglength += blocksize; // padding lengthnya harus nyisain paling gak 8 byte
buat message length + 1 byte buat padding asal yg 128 --> 1xxx
            }

            message_result = new byte[len + paddinglength];
            Array.Copy(message, message_result, len);

            // insert padding max 512bit
            message_result[len] = 128;
            for (long i = 1; i < paddinglength - sizemsgLength; ++i)
            {
                message_result[len + i] = 0;
            }

            // insert msg length 64bit = 8byte
            for (long i = (paddinglength - sizemsgLength); i < paddinglength; ++i)
```

```
            {
                message_result[len + i] = filelen[i - (paddinglength - sizemsgLength)];
            }

            return message_result;
        }

        /// <summary>
        /// get 160bit (20byte) of msg digest from Nx512bit msg data
        /// </summary>
        /// <param name="message">msg to find its msg digest</param>
        /// <returns></returns>
        public static byte[] getMessageDigest(byte[] message)
        {
            long step = 0;
            byte[] result = new byte[20];
            UInt32[] working_abcde = new UInt32[5];

            //for i = 1 to N-512 bit msg
            while (step < message.LongLength)
            {
                UInt32[] Wt = new UInt32[80]; // message schedule
                UInt32 T = new UInt32();

                //initialize Wt
                for (int t = 0; t < 16; ++t)
                {
                    Wt[t] = getUInt32FromNByte(message, step + t * 4, 4);
                }
                for (int t = 16; t < 80; ++t)
                {
                    Wt[t] = ROTL((Wt[t - 3] ^ Wt[t - 8] ^ Wt[t - 14] ^ Wt[t - 16]), 1);
                }


                //initialize 5 working variable a, b, c, d, e with hash value from previous step
                for (int i = 0; i < 5; ++i)
                {
                    working_abcde[i] = H[i];
                }

                /*------------------------
                MAIN. 80loops
                ------------------------*/
                int indexKForUse = 0;
                // loop 0 .. 19

                // fungsi logika pada setiap putaran
                UInt32[] function = new UInt32[4];

                for (int t = 0; t < 80; ++t)
                {
                    function[0] = (working_abcde[1] & working_abcde[2]) ^ (~working_abcde[1] &
working_abcde[3]);
                    function[1] = (working_abcde[1] ^ working_abcde[2] ^ working_abcde[3]);
                    function[2] = (working_abcde[1] & working_abcde[2]) ^ (working_abcde[1] &
working_abcde[3]) ^ (working_abcde[2] & working_abcde[3]);
                    function[3] = (working_abcde[1] ^ working_abcde[2] ^ working_abcde[3]);

                    indexKForUse = t / 20; //putaran 1 0..19 = function[0], dst
                    T = ROTL(working_abcde[0], 5) + function[indexKForUse] + working_abcde[4] +
K[indexKForUse] + Wt[t];
                    //setup new working memory
                    working_abcde[4] = working_abcde[3];
                    working_abcde[3] = working_abcde[2];
                    working_abcde[2] = ROTL(working_abcde[1], 30);
                    working_abcde[1] = working_abcde[0];
                    working_abcde[0] = T;
                }

                //compute the i-th intermediate hash value Hi
                for (int i = 0; i < 5; ++i)
                {
                    H[i] = working_abcde[i] + H[i];
                }
```

```
                step += blocksize;
            }

            //resulting 160-bit message digest of 'message'
            for (int i = 0; i < 5; ++i)
            {
                byte[] tempresult = breakLongIntoArrByte(H[i], 4);
                for (int j = 0; j < 4; ++j)
                {
                    result[i * 4 + j] = tempresult[j];
                }
            }

            return result;
        }
    }
}
```

3. Kelas Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace RC4Hash
{
    public partial class Form1 : Form
    {
        String filename = "";
        byte[] isifile;
        Boolean enkrip_manual = false;
        Boolean dekrip_manual = false;

        public Form1()
        {
            InitializeComponent();
        }



        /***
         *
         * ENCRYPTION TAB
         *
         ***/

        /// <summary>
        /// byte to string converter
        /// </summary>
        /// <param name="b">array of byte b</param>
        /// <returns> a string converter </returns>
        string bytetostring(byte[] b)
        {
            if (b == null)  //kasus kosong
                return "";
            else
            {
                StringBuilder s = new StringBuilder();
                for (int i = 0; i < b.Length; i++)
                {
                    //if (b[i] != 0)
                        s.Append((char)b[i]);
                }

                return s.ToString();
            }
        }
```

```
        /// <summary>
        /// byte to string converter. erase null character
        /// </summary>
        /// <param name="b">array of byte b</param>
        /// <returns> a string converter </returns>
        string bytetostring2(byte[] b)
        {
            if (b == null)  //kasus kosong
                return "";
            else
            {
                StringBuilder s = new StringBuilder();
                for (int i = 0; i < b.Length; i++)
                {
                    if (b[i] != 0)
                    s.Append((char)b[i]);
                }

                return s.ToString();
            }
        }

        /// <summary>
        /// string to byte array converter
        /// </summary>
        /// <param name="Str"></param>
        /// <returns></returns>
        public static byte[] StrToByteArray(string Str)
        {
            System.Text.UTF8Encoding encoding = new System.Text.UTF8Encoding();
            return encoding.GetBytes(Str);
        }

        /// <summary>
        /// browse file in encrypt tab
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void enkrip_browse_Click(object sender, EventArgs e)
        {
            OpenFileDialog o = new OpenFileDialog();
            o.Title = "Open File";
            DialogResult d = o.ShowDialog();

            if (d == DialogResult.OK)
            {
                filename = o.FileName;
                enkrip_filename.Text = filename;

                isifile = File.ReadAllBytes(filename);

                enkrip_filetext.Text = bytetostring2(isifile);
            }
        }

        private void enkrip_inputmanual_CheckedChanged(object sender, EventArgs e)
        {
            enkrip_manual = !enkrip_manual;
            enkrip_filetext.Enabled = enkrip_manual;
        }


        /// <summary>
        /// main process
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void encrypt_Click(object sender, EventArgs e)
        {
            if (enkrip_manual)
            {
                isifile = StrToByteArray(enkrip_filetext.Text.ToString());
            }
```

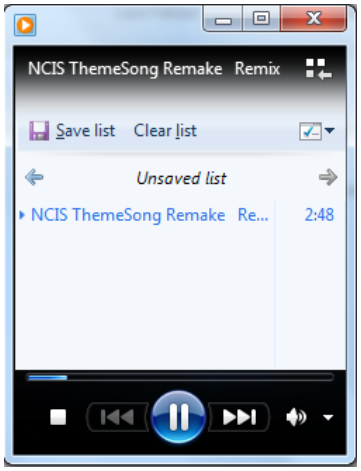
```
            isifile = algorithm.RC4.encrypt_RC4(isifile,
StrToByteArray(enkrip_keyarea.Text.ToString()));
            enkrip_result.Text = bytetostring2(isifile);
        }


        /***
         *
         * DECRYPTION TAB
         *
         ***/
        /// <summary>
        /// browse file in the decrypt tab
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void dekrip_browse_Click(object sender, EventArgs e)
        {
            OpenFileDialog o = new OpenFileDialog();
            o.Title = "Open File";
            DialogResult d = o.ShowDialog();

            if (d == DialogResult.OK)
            {
                filename = o.FileName;
                dekrip_filename.Text = filename;

                isifile = File.ReadAllBytes(filename);

                dekrip_fileteks.Text = bytetostring2(isifile);
            }
        }

        private void dekrip_inputmanual_CheckedChanged(object sender, EventArgs e)
        {
            dekrip_manual = !dekrip_manual;
            dekrip_fileteks.Enabled = dekrip_manual;
        }


        /// <summary>
        /// main decryption process
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void decrypt_Click(object sender, EventArgs e)
        {
            if (dekrip_manual)
                isifile = StrToByteArray(enkrip_filetext.Text.ToString());

            isifile = algorithm.RC4.decrypt_RC4(isifile, StrToByteArray(textBox5.Text.ToString()));
            dekrip_result.Text = bytetostring2(isifile);
        }

        private void dekrip_saveas_Click(object sender, EventArgs e)
        {
            //save sugnature in same document
            SaveFileDialog s = new SaveFileDialog();
            s.Title = "Save File with Signature";
            DialogResult d = s.ShowDialog();

            if (d == DialogResult.OK)
            {
                BinaryWriter bw = new BinaryWriter(File.OpenWrite(s.FileName));
                bw.Write(isifile);
                bw.Close();
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //save sugnature in same document
            SaveFileDialog s = new SaveFileDialog();
            s.Title = "Save File with Signature";
            DialogResult d = s.ShowDialog();
```

```
            if (d == DialogResult.OK)
            {
                BinaryWriter bw = new BinaryWriter(File.OpenWrite(s.FileName));
                bw.Write(isifile);
                bw.Close();
            }
        }
    }
}
```

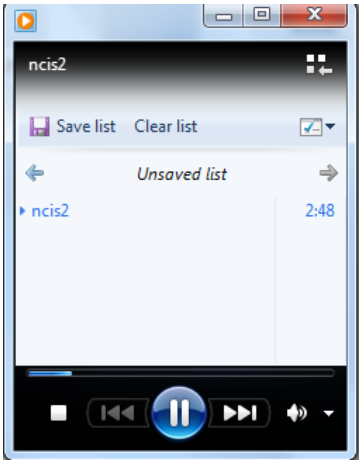
C. Screenshot hasil pengujian

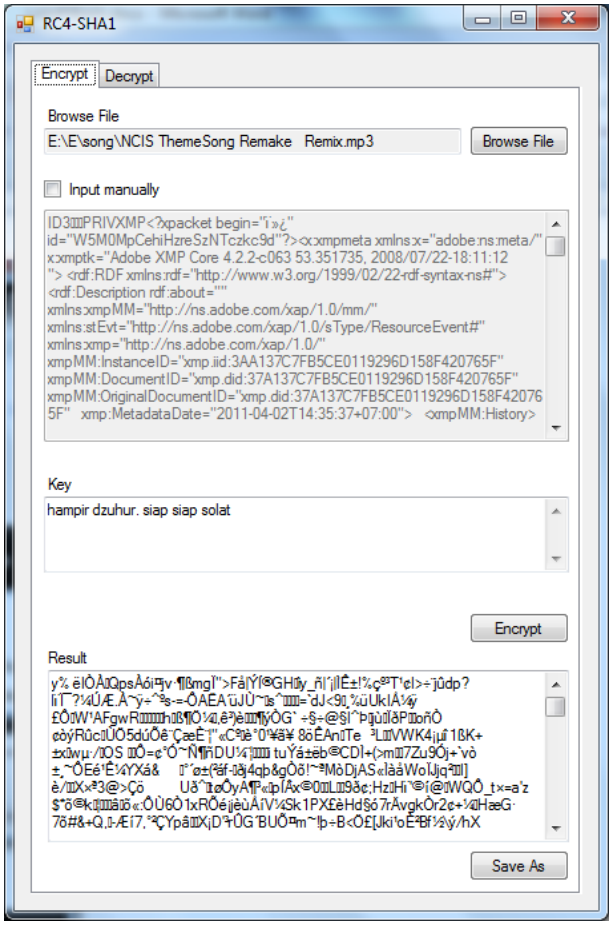
1. Pengujian terhadap file berekstensi .mp3

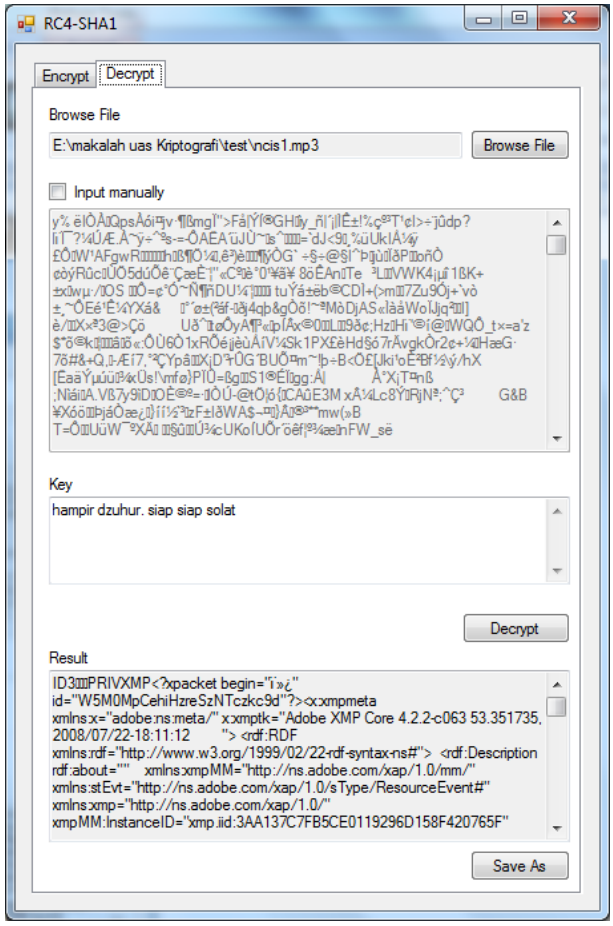
before :



after :

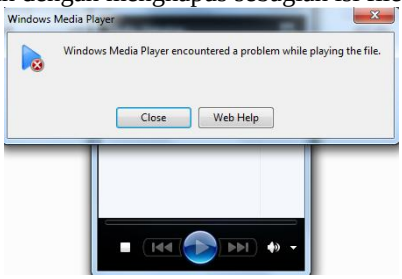






Pengujian dengan menggunakan kunci yang salah menyebabkan file tidak dapat dibuka.

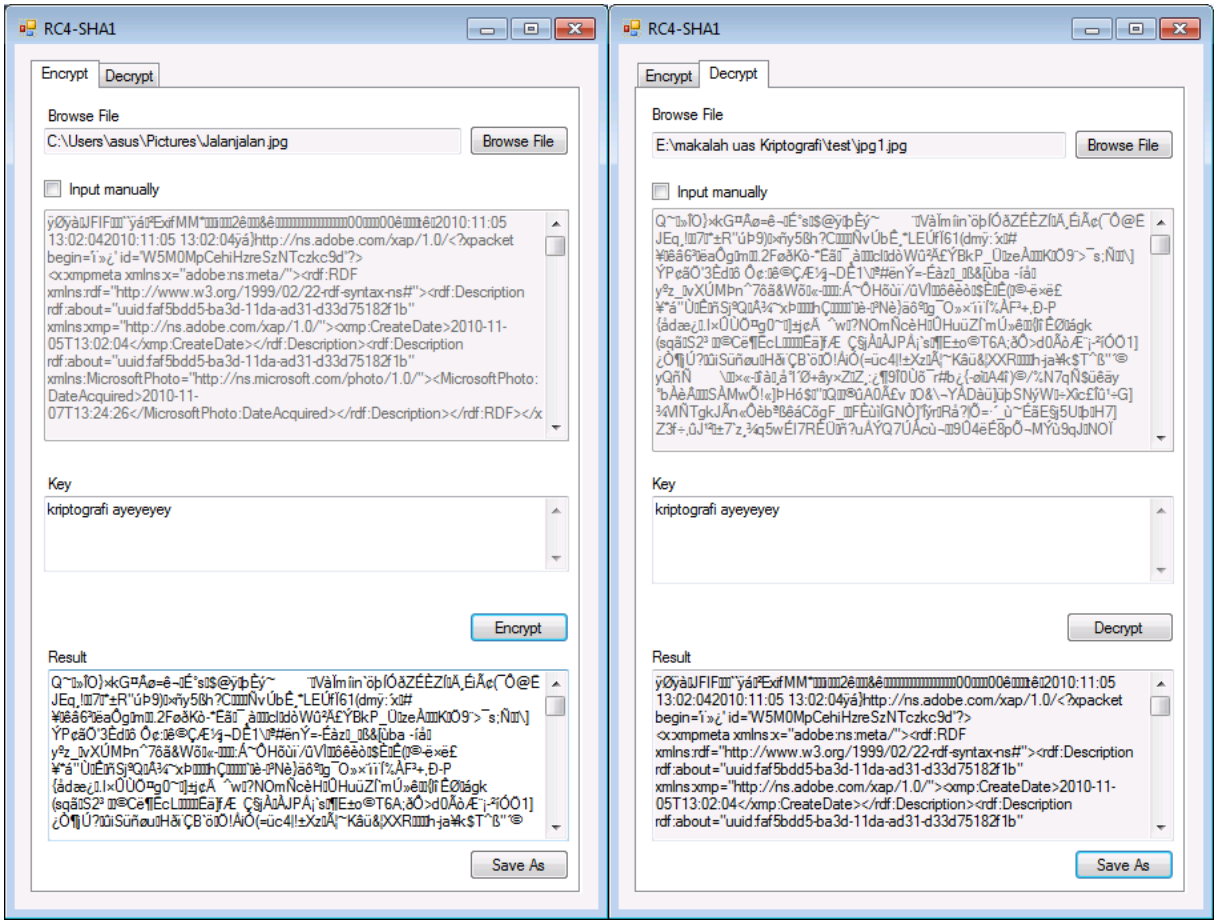
Pengujian dengan menghapus sebagian isi file menyebabkan file tidak dapat dibuka.



2. Pengujian terhadap file berekstensi .jpg

before :

after :

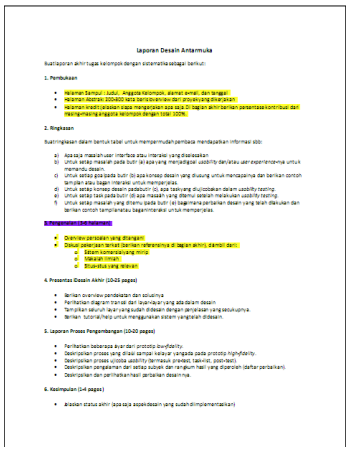


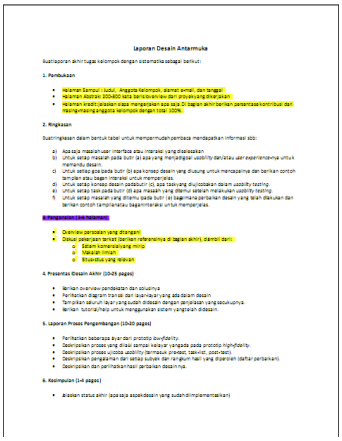
Pengujian dengan menggunakan kunci yang salah menyebabkan file tidak dapat dibuka.

Pengujian dengan menghapus sebagian isi file menyebabkan file tidak dapat dibuka.

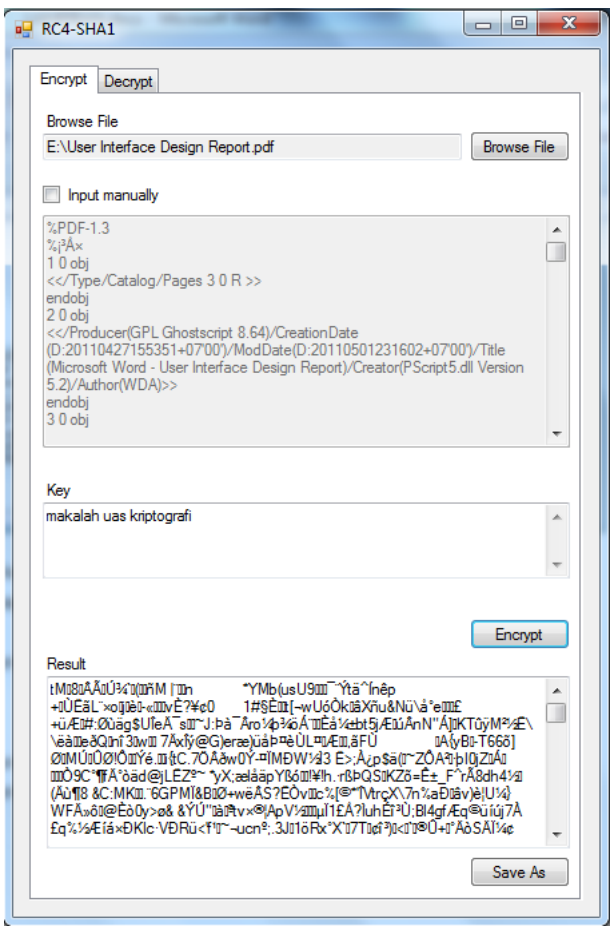
3. Pengujian terhadap file berekstensi .pdf

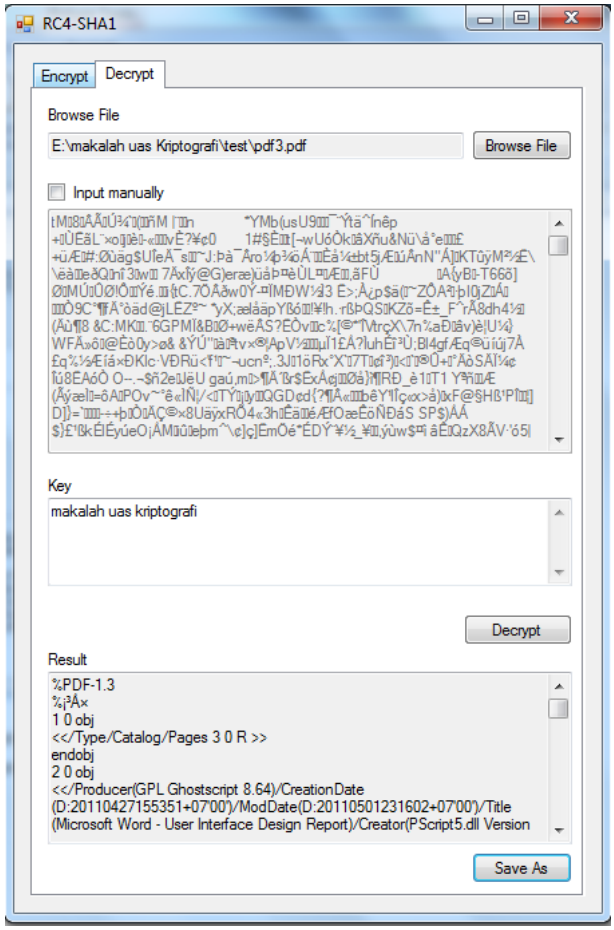
before :



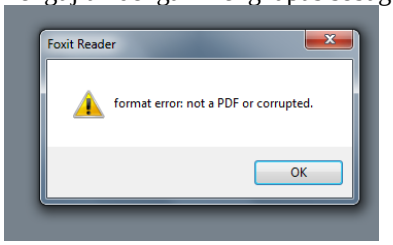


after :



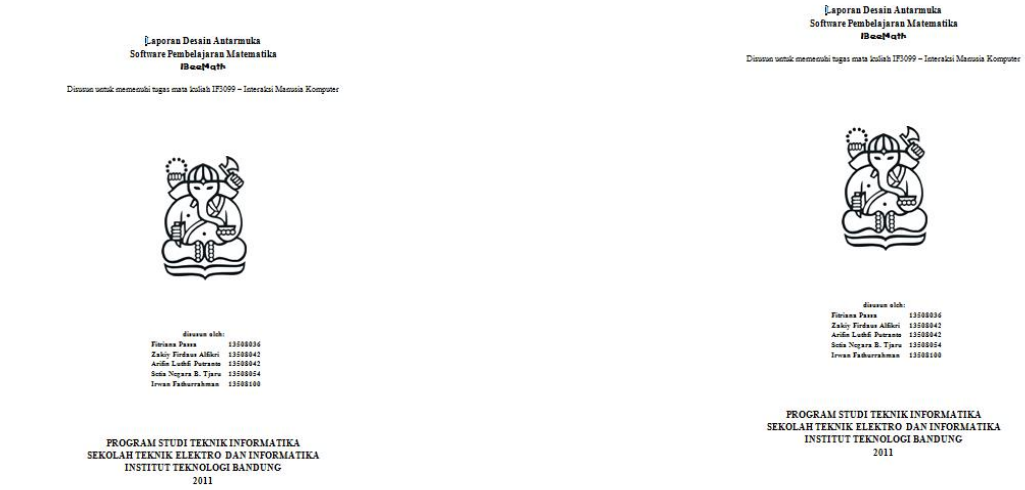


Pengujian dengan menggunakan kunci yang salah menyebabkan file tidak dapat dibuka.
Pengujian dengan menghapus sebagian isi file menyebabkan file tidak dapat dibuka.

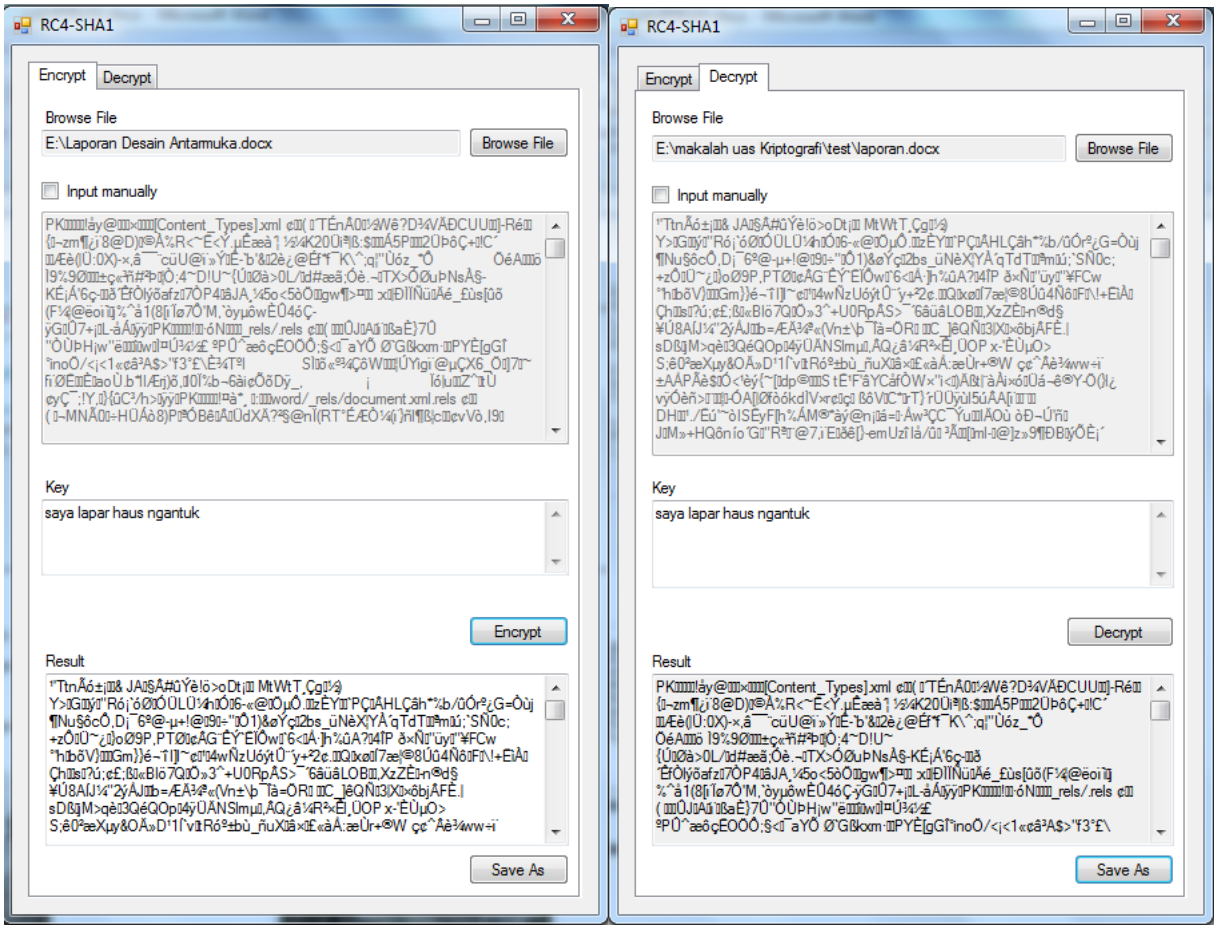


4. Pengujian terhadap file berekstensi .doc

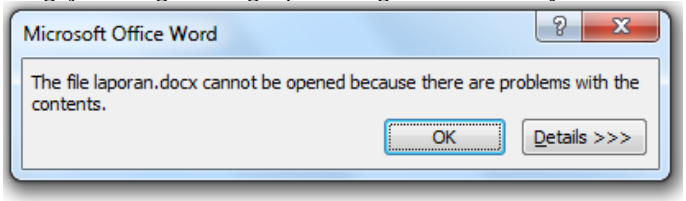
before :



after :

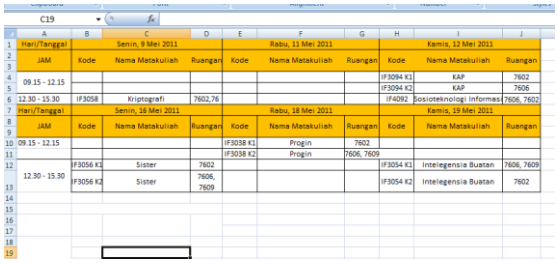


Pengujian dengan menggunakan kunci yang salah menyebabkan file tidak dapat dibuka.
Pengujian dengan menghapus sebagian isi file menyebabkan file tidak dapat dibuka.

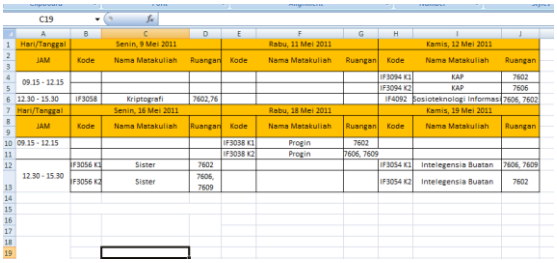


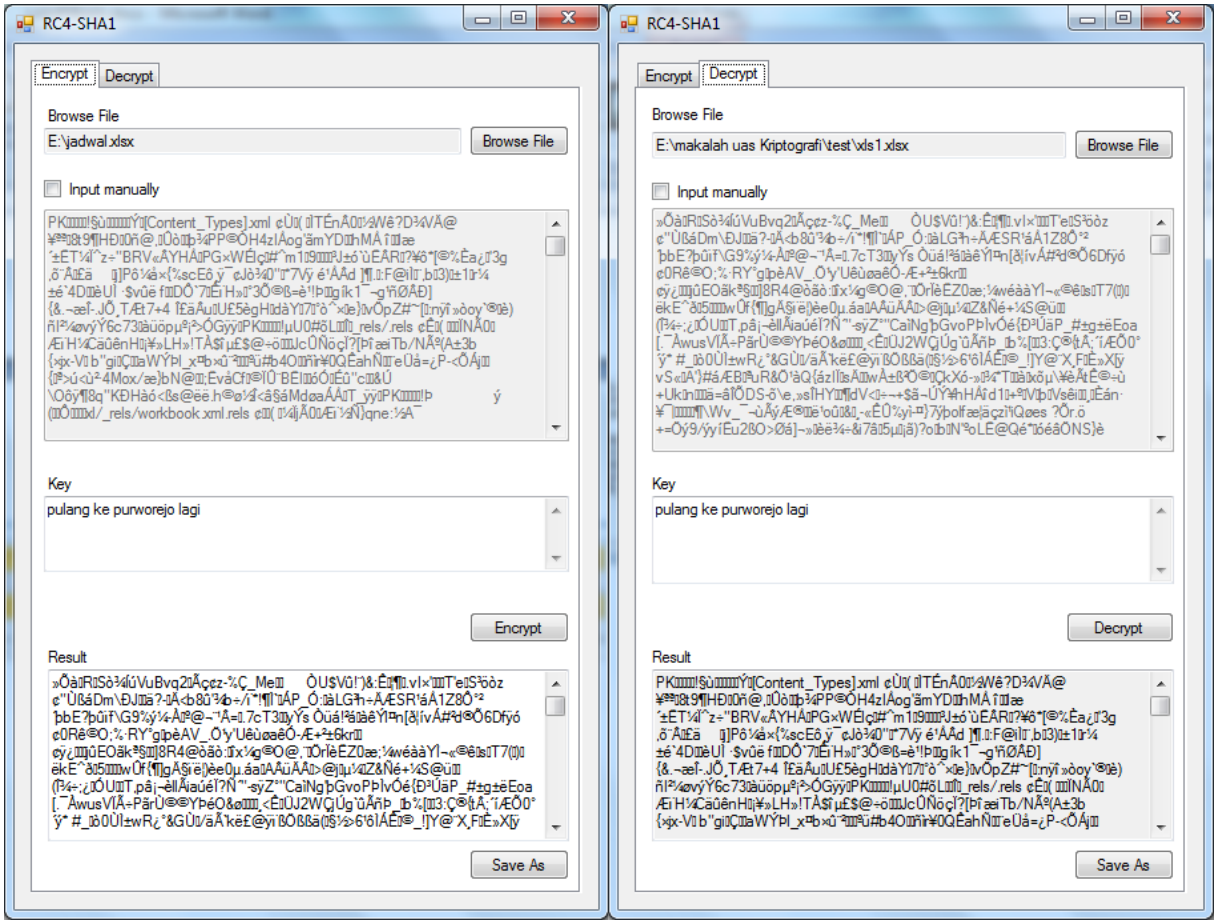
5. Pengujian terhadap file berekstensi .xls

before:



after:





Pengujian dengan menggunakan kunci yang salah menyebabkan file tidak dapat dibuka.

Pengujian dengan menghapus sebagian isi file menyebabkan file tidak dapat dibuka.