IMPLEMENTASI APLIKASI SAFC(SAFE AUTHENTICATED FILE COPIER) DENGAN PEMANFAATAN MAC (MESSAGE AUTHENTICATION CODE) UNTUK DUPLIKASI DATA.

Jonathan Ery Pradana / 13508007
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18007@students.if.itb.ac.id

Penggunaan dari fungsi duplikasi data saat ini sudah menjadi hal yang mutlak ada dalam kehidupan sehari-hari. Dalam proses duplikasi tersebut, keamanan dari file baru hasil duplikasi itu menjadi tumpuan agar data-data dari file yang baru itu dapat diketahui keasliannya. Namun, seiring dengan majunya teknologi dan perubahan jaman, serta semakin dibutuhkannya fitur duplikasi data dalam kehidupan sehari-hari ini, semakin banyak juga peluang kejahatan yang mungkin terjadi. Kejahatan yang ada memiliki banyak tipe yang beragam, dimulai dari pemalsuan data, duplikasi data tanpa seijin pemilik, serta pemalsuan hak milik. Oleh karena masalah-masalah tersebut, dibutuhkan suatu inovasi dan teknik untuk memberi peringatan atau tanda, apabila terjadi suatu perubahan pada file-file baru hasil duplikasi dari file tersebut. Saat konten dari file baru itu diterima oleh seseorang, maka perlu sistem pencocokan untuk menentukan apakah file baru hasil duplikasi itu layak untuk digunakan sesuai dengan file aslinya. Mekanisme pencocokan ini akan dapat memberikan notifikasi agar penggunanya makin waspada dengan kejahatan-kejahatan yang mungkin terjadi. Saat ini sudah terdapat suatu mekanisme untuk memeriksa integritas terhadap sebuah file atau data, yaitu sebuah teknik yang disebut dengan MAC (Message Authentication Code). SAFC (Safe Authenticated File Copier) merupakan sebuah aplikasi yang memanfaatkan mekanisme MAC untuk menghasilkan solusi dari permasalahan diatas. Dalam implementasinya SAFC menggunakan SHA-1 dan RSA untuk menggenerate kode sisipannya.

Index: SAFC, MAC, File, Duplikasi, SHA-1, RSA

I. PENDAHULUAN

Duplikasi berkas merupakan sesuatu yang sudah lumrah dilakukan setiap hari oleh pengguna komputer. Dalam proses duplikasi tersebut, keamanan dari file baru hasil duplikasi itu menjadi tumpuan agar data-data dari file yang baru itu dapat diketahui keasliannya. Namun, seiring dengan majunya teknologi dan perubahan jaman, serta semakin dibutuhkannya fitur duplikasi data dalam kehidupan sehari-hari ini, semakin banyak juga peluang kejahatan yang mungkin terjadi. Kejahatan yang ada memiliki banyak tipe yang beragam, dimulai dari pemalsuan data, duplikasi data tanpa seijin pemilik, serta pemalsuan hak milik.

Kriptografi, secara umum adalah ilmu atau seni untuk menjaga kerahasiaan suatu berita atau data. Kriptografi juga dapat berarti teknik matematika yang berhubungan dengan aspek keamanan informasi, namun tidak semua aspek keamanan informasi ditangani oleh kriptografi. Saat ini ilmu kriptografi masih terus berkembang. Hanya saja, penggunaan ilmu ini sudah makin luas dan jumlah penggunaannya oleh orang-orang di seluruh dunia juga makin banyak seiring berkembangnya ilmu dari kriptografi ini. Pada awalnya ilmu ini hanya digunakan pada bidang militer, diplomatic, atau pemerintahan secara umum saja. Namun kemudian ilmu ini sekarang digunakan oleh pihak swasta sebagai alat untuk melindungi informasi-informasi yang sifatnya vital agar kerahasiaannya dapat dijaga.

Seiring perkembangan ilmu kriptografi, jenis-jenis teknik kriptografi pun bermunculan pula. Pada umumnya, kebanyakan ilmu kriptografi di dunia diklasifikasikan sebagai berikut:

- Algoritma sandi
  - Simetris
    - Block-Cipher
    - Stream-Cipher
  - Asimetris
    - Fungsi Enkripsi dan Dekripsi Algoritma - Sandi Kunci-Asimetris
- Algoritma untuk Hash

Fungsi sandi itu bertujuan untuk mengamankan informasi. Hal-hal yang harus dimiliki oleh fungsi sandi adalah kekuatan untuk mengacak dan membingungkan orang yang melihatnya, selain itu difusi atau peleburan juga menambah kekuatan dari fungsi sandi itu. Dengan menggunakan algoritma yang kuat dalam mengacak dan meleburkan data, diharapkan suatu informasi akan aman dari orang-orang yang tidak berhak.

Fungsi hash berbeda dengan fungsi sandi yang bertujuan untuk mengamankan data, fungsi hash ini pada umumnya digunakan untuk keperluan autentikasi dan integritas data. Dengan menggunakan fungsi hash, maka manipulasi terhadap informasi akan dapat diketahui. Berdasarkan masalah keamanan yang ada pada duplikasi data, penulis akan menggunakan fungsi hash untuk memberikan kode integritas yang disisipkan pada file di akhir data. Teknik khusus untuk melakukan hal ini disebut MAC (Message Authentication Code).

II. DASAR TEORI

Berikut akan dipaparkan beberapa dasar-dasar teori terkait yang membantu penulis dalam mengimplementasi SAFC ini.

1. MAC (Message Authentication Code)

MAC adalah fungsi satu arah yang menggunakan kunci rahasia dalam pembangkitan nilai hash. MAC merupakan salah satu teknik kriptografi dalam klasifikasi yang menggunakan fungsi hash. Berbeda dengan fungsi MD5 atau SHA-1 yang tidak memerlukan kunci untuk membangkitkan nilai hash, MAC memerlukan kunci dalam membangkitkan nilai hash. Seperti kebanyakan teknik hash yang lain, MAC akan menghasilkan nilai hash yang panjangnya tetap (fixed). Biasanya MAC dilekatkan dengan pesan yang dikirim yang selanjutnya akan digunakan untuk autentikasi tanpa merahasiakan pesan. MAC bukanlah tanda tangan digital, namun hanya digunakan untuk menyediakan autentikasi pengirim.

MAC = C[k](M)

MAC = nilai hash

C = fungsi hash (atau algoritma MAC)

k = kunci rahasia

Dengan menggunakan sebuah fungsi hash, dan mengenkripsi hasil fungsi hash itu dengan algoritma enkripsi beserta sebuah kunci rahasia, maka akan didapat nilai hash MAC.

2. Fungsi Hash SHA-1

SHA-1 merupakan algoritma hash yang banyak diaplikasikan dalam keamanan protokol menggunakan SSL (Secure Sockets Layer), PGP (Pretty Good Privacy), XML Signature, dan beberapa aplikasi lainnya. SHA 1 digunakan karena memiliki beberapa kelebihan antara lain:

- **Validate password (memvalidasi password);** nilai hash dari password akan disimpan, kemudian ketika password diotentikasi, maka password yang dimasukkan oleh user akan dihitung hashnya dan jika hashnya sesuai maka password dinyatakan valid. Namun untuk mendapatkan password yang asli tidak dapat diperoleh dari hash yang telah disimpan.
- **Challenge handshake authentication;** untuk menghindari kesalahan pengiriman password dalam kondisi "clear", client dapat mengirim nilai hash sebuah password melalui internet untuk divalidasi oleh server tanpa beresiko disadapnya password yang asli.
- **Anti-tamper ;** untuk memastikan data tidak berubah selama ditransmisikan. Penerima akan menghitung nilai hash dan mencocokkan dengan hash yang dikirimkan, apabila nilainya sama berarti data yang dikirimkan tidak berubah.
- **Digital signatures;** dilakukan dengan cara mengenkrip nilai hash sebuah dokumen dengan menggunakan private key, sehingga menghasilkan tanda tangan digital untuk dokumen tersebut. Orang lain dapat mengecek otentikasi dokumen tersebut dengan cara mendekrip tanda tangan tersebut menggunakan public key untuk mendapatkan nilai hash yang asli dan membandingkannya dengan nilai hash dari teks.

Langkah-langkah pada SHA-1 adalah sebagai berikut:

1. Melakukan padding terhadap pesan sehingga panjangnya adalah 448 modulus 512.
2. 64 bit sisanya adalah representasi biner dari panjang pesan.
3. Melakukan inisialisasi5 word buffer (160 bit)A, B, C, D, dan E dengan nilai A=67452301, B=efcdab89, C=98badcfe, D= 10325476, dan E=c3d2e1f0.
4. Memproses pesan dalam blok-blok 16 word (512 bit) dengan ketentuan:
5. Ekspansi 16 words menjadi 80 words dengan teknik mixing dan shifting.
6. Menggunakan 4 round dari 20 operasi bit pada blok pesan dan buffer.
7. Menambahkan output dengan input untuk memperoleh nilai buffer yang baru. Output nilai hash adalah nilai terakhir dari buffer.

3. Fungsi Enkripsi dan Dekripsi RSA

RSA adalah salah satu contoh kriptografi yang menerapkan konsep public key. Algoritma ini pertama kali dipublikasikan di tahun 1977 oleh Ron Rivest, Adi Shamir, dan Leonard Adleman dari Massachusetts Institute of Technology (MIT). Nama RSA sendiri adalah singkatan dari nama belakang mereka bertiga.

Clifford Cocks, seorang matematikawan Inggris sebenarnya juga telah mengembangkan algoritma yang hampir sama dengan RSA ini pada tahun 1973. Namun algoritma buatannya tidak begitu dikenal oleh publik, dan baru dipublikasi pada tahun 1997 karena merupakan proyek rahasia. Walau begitu algoritma yang dikembangkan Rivest, Shamir, dan Adleman tidak berhubungan dengan pekerjaan Cocks.

Pada algoritma RSA terdapat 3 langkah utama yaitu key generation (pembangkitan kunci), enkripsi, dan dekripsi.

Kunci pada RSA mencakup dua buah kunci, yaitu public key dan private key. Public key digunakan untuk melakukan enkripsi, dan dapat diketahui oleh orang lain. Sedangkan private key tetap dirahasiakan dan digunakan untuk melakukan dekripsi.

Pembangkitan kunci atau key generation dari RSA adalah sebagai berikut :

1. Pilih dua buah bilangan prima sembarang a dan b. Jaga kerahasiaan a dan b ini.
2. Hitung n = a * b. Besaran n ini tidak perlu dirahasiakan.

3. Hitung m = (a-1) * (b-1). Sekali m telah dihitung, a dan b dapat dihapus untuk mencegah diketahuinya oleh pihak lain.
4. Pilih sebuah bilangan bulat untuk kunci publik, sebut namanya e, yang relatif prima terhadap m (relatif prima berarti GCD(e, m) = 1) dengan syarat e ≠ (p-1), e ≠ (q-1), dan e < n.
5. Hitung kunci dekripsi, d, dengan kekongruenan ed ≡1 (mod m).Perhatikan bahwa nilai yang didapat sama dengan nilai awal, yaitu 48.

Proses enkripsi dapat dilakukan dengan:

Ci =Pie modn

Sedangkan proses dekripsi dilakukan dengan:

Pi =Cid modn

Blok-blok plainteks dinyatakan dengan p1, p2, p3, … (harus dipenuhi persyaratan bahwa nilai pi harus terletak dalam himpunan nilai 0, 1, 2, ... , n-1 untuk menjamin hasil perhitungan tidak berada di luar himpunan).

Pada langkah kelima pembangkitan kunci atau key generation, kekongruenan ed ≡ 1 (mod m) sama dengan ed mod m = 1. Sehingga dapat pula dikatakan bahwa ed ≡1 (mod m) ekivalen dengan ed = 1 + km.

Dengan mencoba nilai k = 1, 2, 3, ..., diperoleh nilai d yang bulat. Nilai itu yang akan dipakai sebagai kunci pribadi untuk dekripsi pesan.

Dalam implementasi sebenarnya, nilai a dan b diharapkan sangat besar sekali (misalnya 100 digit) agar pekerjaan memfaktorkan n menjadi faktor primanya menjadi sangat sukar, sehingga lebih susah untuk ditembus.

Kekuatan algoritma RSA terletak pada tingkat kesulitan dalam memfaktorkan bilangan menjadi faktor primanya, dalam hal ini memfaktorkan n menjadi a dan b. Karena sekali n berhasil difaktorkan, maka menghitung nilai m adalah perkara mudah. Selanjutnya, walau nilai e diumumkan, perhitungan kunci d tidaklah mudah pula karena nilai m yang tidak diketahui.

4. Kejahatan pada File

- VIRUS

  Virus merupakan salah satu jenis kejahatan file yang berupa penambahan suatu program jahat ke dalam folder. Nantinya virus ini dapat mengakibatkan suatu hal yang tidak diinginkan.
- PENAMBAHAN FILE

  Penambahan file dapat berupa suatu kegiatan memasukkan file tambahan yang bukan dibuat oleh pengirim. Dengan penambahan file ini, penerima pesan akan dibuat bingung dengan tidak mengetahui file mana yang dikirim oleh pengirim asli, dan mana yang bukan.
- PENGURANGAN FILE

  Pengurangan file dapat berupa manipulasi isi folder kiriman dengan menghilangkan semua atau sebagian file yang terdapat dalam folder.
- PERUBAHAN NAMA FILE

  Dengan perubahan nama file ini, maka informasi yang tadinya tersusun dengan rapih, dapat menjadi tidak teratur. Nama file ini memang tidak merubah secara keseluruhan isi dari file, namun hal ini tetap dapat berakibat fatal. Sebagai contoh, pada file-file notulensi rapat yang tergabung pada satu folder. Apabila nama-nama file notulensi ini ditukar, maka akan terjadi suatu mis-komunikasi mengenai pencatatan kejadian pada rapat.
- PERUBAHAN ISI FILE

  Perubahan pada isi file dapat berpengaruh besar, mulai dari perubahan informasi yang ingin dikirimkan, sampai hilangnya informasi.
- PERUBAHAN BYTE FILE

  Perubahan satu byte pada sebuah file mungkin terdengar tidak signifikan, namun dapat berakibat sebuah file tidak dapat dibuka, atau rusak sama sekali.

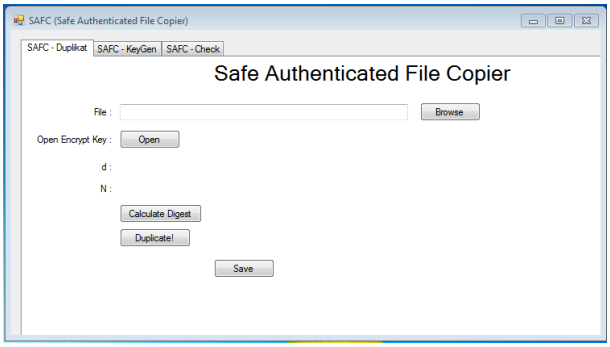
Kejahatan file yang sudah dipaparkan di atas akan menjadi sorotan penulis apakah file dengan menggunakan teknik FAC dapat dianalisa autentikasinya dengan akurat.

III. IMPLEMENTASI SAFC (SAFE AUTHENTICATED FILE COPIER)

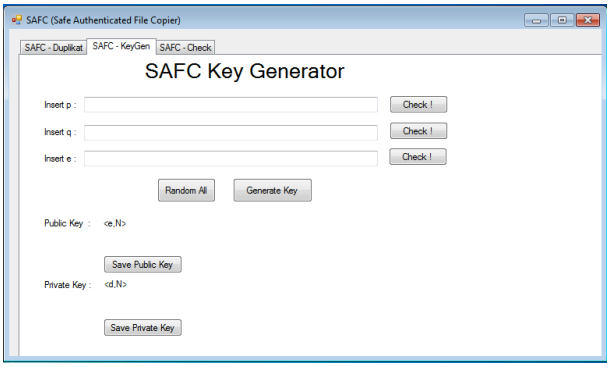
Safe Authenticated File Copier (SAFC) adalah suatu solusi untuk permasalahan autentikasi pada duplikasi data. SAFC menggunakan teknik Message Authentication Code (MAC) untuk disisipkan di belakang data dalam file. Penyisipan data ini berguna untuk memberikan data autentikasi yang nantinya akan digunakan oleh aplikasi ini untuk mencocokan apakah file duplikasi tersebut sesuai dengan file yang asli dengan menggunakan kunci yang diberikan oleh pemberi file.

Pada implementasinya, proses penyisipan autentikasi pada SAFC ini disusun oleh dua algoritma penting yaitu SHA-1 dan RSA. SHA-1 berguna untuk menemukan nilai hash dari data yang ada untuk dipublikasi, sedangkan RSA berfungsi untuk mengenkripsi nilai hash yang sudah ditemukan oleh file. Saat file yang sudah diduplikasi diberikan kepada penerima, penerima diharapkan melakukan pengecekan lewat aplikasi ini juga untuk mengetahui autentikasi/integritas dari file.

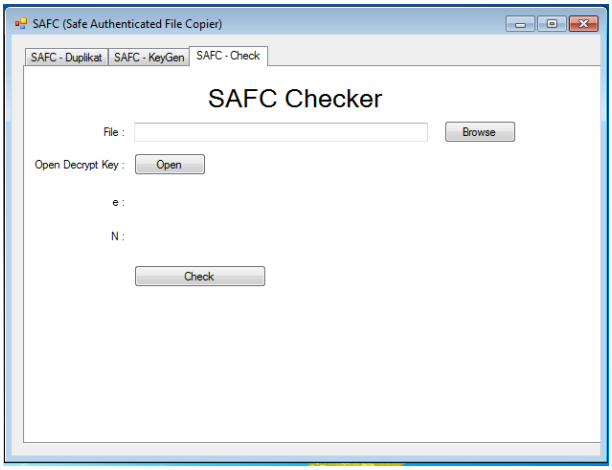
Berikut penulis sertakan beberapa gambar aplikasi hasil implementasi SAFC ini.



Gambar 1. Antarmuka SAFC-Duplicate



Gambar 2. Antarmuka SAFC-KeyGen



Gambar 3. Antarmuka SAFC-Checker

IV. PENGUJIAN

Kasus Normal

Sebuah file gambar Koala.jpg akan diduplikasi dengan key:

D:67732650472103592826455950086657135397234157692964572360792731460772511228480159372271147778018748156691055412549222325292035423591893931315989077076698947

N:78873271659332026927143051843716887621310470674908520134373757738090449196602708468718289396457966859505022308731735312890597454636308498144647160940764 89

Didapat pada akhir file Koala.jpg akan disisipkan kode MAC yang telah digenerate aplikasi:

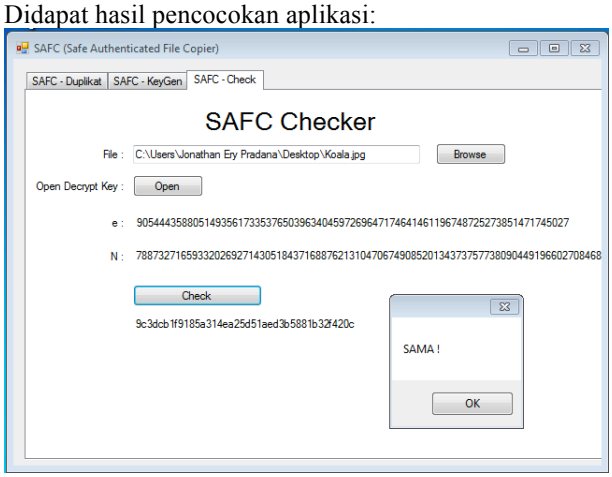
<mac>4c26c57674d71de16372474ab514e252b3deaca0bfa16973e613872d755ad29cc97d3e3b9b08555a66d16a6e3616753e2d04f504401f0fad3e160fe9ee8453bb</mac>

Ketika dilakukan pencocokan file dengan key:

E:90544435880514935617335376503963404597269647174641461196748725273851471745027

N:78873271659332026927143051843716887621310470674908520134373757738090449196602708468718289396457966859505022308731735312890597454636308498144647160940764 89

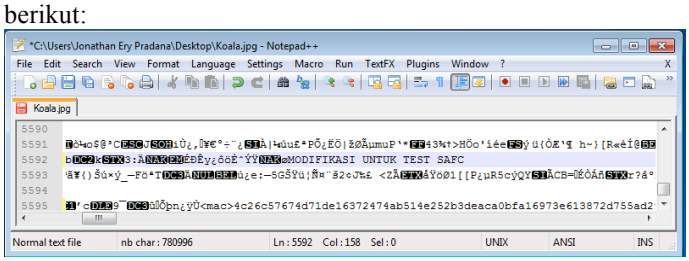
Didapat hasil pencocokan aplikasi:



Gambar 4. Antarmuka hasil pencocokan aplikasi

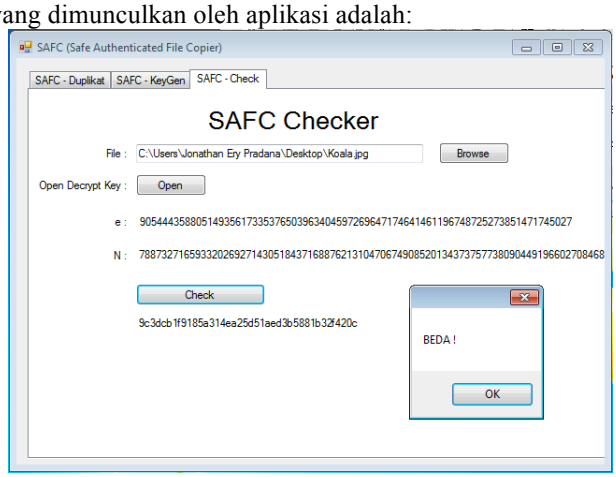
Kasus adanya modifikasi pada byte file

Misalkan Koala.jpg yang sudah diduplikasi tadi kita modifikasi filenya dengan aplikasi Notepad++ sebagai berikut:



Gambar 5. Antarmuka modifikasi byte file Koala.jpg

Lalu kita lakukan pencocokan dengan key yang sama saat mencocokan dengan scenario pertama tadi, hasil yang dimunculkan oleh aplikasi adalah:



Gambar 6. Antarmuka SAFC-Checker pada Koala.jpg yang termodifikasi

V. SOURCE CODE

Berikut akan penulis lampirkan beberapa source code penting yang akan penulis bagi sesuai fungsinya

Algoritma SHA-1 buatan penulis

```
class algorithm
    {
        //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;

        //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>
        /// 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>
        /// 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>
        /// 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;
        }
    }
```

Inisiasi Fungsi Hash:

```
private void initMyHash()
        {
            //init vars
            algorithm.H = new UInt32[5];
            algorithm.K = new UInt32[4];
            algorithm.H[0] = 1732584193;
            algorithm.H[1] = 4023233417;
            algorithm.H[2] = 2562383102;
            algorithm.H[3] = 271733878;
            algorithm.H[4] = 3285377520;
            algorithm.K[0] = 1518500249;
            algorithm.K[1] = 1859775393;
            algorithm.K[2] = 2400959708;
            algorithm.K[3] = 3395469782;
```

```
            //create array of byte from ABCDE
            algorithm.ABCDE = new byte[20]; // 20
byte = 20 * 8 = 160bit
            algorithm.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 =
algorithm.breakLongIntoArrByte(algorithm.H[i], 4);
                byte[] dump2 =
algorithm.breakLongIntoArrByte(algorithm.K[i], 4);
                for (int j = 0; j < 4; ++j)
                {
                    algorithm.ABCDE[i * 4 + j] =
dump[j];
                    algorithm.H1234[i * 4 + j] =
dump2[j];
                }
            }

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

Fungsi enkripsi RSA:

```
private byte[] RSAdigest(byte[] dig, BigInteger E,
BigInteger N)
        {
            byte[] EncryptData = new
byte[dig.Length];
            BigInteger[] EncryptTemp;
            Array.Copy(dig, EncryptData,
dig.Length);
            int blocklen = 0;
            BigInteger val = new BigInteger(256);
            while (val < (N - 1))
            {
                blocklen += 1;
                val = val * 256;
            }
            int len = EncryptData.Length /
blocklen;
            if ((EncryptData.Length % blocklen) !=
0)
                len += 1;
            EncryptTemp = new BigInteger[len];
            for (int i = 0; i < (len - 1); i++)
            {
                byte[] convbyte = new
byte[blocklen];
                Array.Copy(EncryptData, i *
blocklen, convbyte, 0, blocklen);
                EncryptTemp[i] = new
BigInteger(convbyte);
            }
            //last array
            byte[] EncryptResult = new
byte[blocklen];
            int nLeft = EncryptData.Length - ((len
- 1) * blocklen);
            Array.Copy(EncryptData, (len - 1) *
blocklen, EncryptResult, blocklen - nLeft, nLeft);
            EncryptTemp[len - 1] = new
BigInteger(EncryptResult);

            //calculate the encryption
            for (int i = 0; i < EncryptTemp.Length;
i++)
            {
                EncryptTemp[i] = new
BigInteger(EncryptTemp[i].modPow(E, N));
            }

            //convert to bytes
            EncryptResult = new
byte[EncryptTemp.Length * (blocklen + 1)];
            int counter = 0;
            for (int i = 0; i < EncryptTemp.Length;
i++)
            {
                byte[] convTemp =
bigIntToBytes(EncryptTemp[i], blocklen + 1);
                foreach (byte b in convTemp)
                {
                    EncryptResult[counter] = b;
                    counter++;
                }
            }

            return EncryptResult;
        }
```

Fungsi dekripsi RSA:

```
private byte[] decryptRSADigest(byte[] encDig,
BigInteger e, BigInteger N)
        {
            //calculate the length of each block
            int blocklen = 0;
            BigInteger val = new BigInteger(256);
            while (val < (N - 1))
            {
                blocklen += 1; ;
                val = val * 256;
            }

            //get data to decrypt
            byte[] temp = new byte[encDig.Length];
            Array.Copy(encDig, temp,
encDig.Length);

            //prepare array of BigInteger
            BigInteger[] DecryptTemp;
            DecryptTemp = new
BigInteger[temp.Length / (blocklen + 1)];
            for (int i = 0; i < DecryptTemp.Length;
i++)
            {
                byte[] convbyte = new byte[blocklen
+ 1];
                Array.Copy(temp, i * (blocklen +
1), convbyte, 0, blocklen + 1);
                DecryptTemp[i] = new
BigInteger(convbyte);
            }

            //calculate the decryption
            for (int i = 0; i < DecryptTemp.Length;
i++)
            {
                DecryptTemp[i] =
DecryptTemp[i].modPow(e, N);
            }

            //convert to bytes
            byte[] DecryptResult = new byte[20];
            int counter = 0;
            int arrCounter = 0;
            while (arrCounter < (DecryptTemp.Length
```

```
- 1))
            {
                byte[] conv =
bigIntToBytes(DecryptTemp[arrCounter], blocklen);
                for (int i = 0; i < blocklen; i++)
                {
                    DecryptResult[counter] =
conv[i];
                    counter++;
                }
                arrCounter++;
            }
            //last array
            byte[] last =
bigIntToBytes(DecryptTemp[arrCounter], blocklen);
            int sisa = 20 % blocklen;
            Array.Copy(last, blocklen - sisa,
DecryptResult, counter, sisa);

            return DecryptResult;
        }
```

Fungsi Duplikasi dan Penyisipan MAC pada File:

```
private void duplicateButton_Click(object sender,
EventArgs e)
        {
            FileStream fs = new
FileStream(this.filename, FileMode.Append,
FileAccess.Write);
            String digest =
bytesToString(this.encryptedDig);
            byte[] mac = this.stringToByte(digest);

            //append to file
            fs.Write(stringToByte("<mac>"), 0,
stringToByte("<mac>").Length);
            fs.Write(mac, 0, mac.Length);
            fs.Write(stringToByte("</mac>"), 0,
stringToByte("</mac>").Length);

            fs.Close();

            MessageBox.Show("Duplikasi Berhasil
Dilakukan");
        }
```

V. KESIMPULAN

Dari serangkaian analisis yang telah dilakukan oleh penulis, penulis dapat menyimpulkan beberapa hal, antara lain:

1. SAFC ini dapat digunakan untuk pengecekan autentikasi file. Aplikasi ini akan memberikan notifikasi apakah file tersebut merupakan duplikat yang sesuai atau tidak.
2. User dapat menggenerate key dan menyimpannya untuk digunakan pada penyisipan MAC di SAFC.
3. SAFC ini dapat mengecek apabila ada perubahan byte yang dilakukan pada file duplikat.

DAFTAR PUSTAKA

[1] *www.informatika.org/~rinaldi*. Tanggal akses: 8 Mei 2011
[2] http://www.jansonhendryli.net/download/rsa.pdf. Tanggal akses: 8 Mei 2011
[3] http://en.wikipedia.org/wiki/Hash_function. Tanggal akses: 8 Mei 2011
[4] http://id.wikipedia.org/wiki/RSA. Tanggal akses: 8 Mei 2011

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, 9 Mei 2011
ttd

Jonathan Ery Pradana
13508007