STUDI DAN IMPLEMENTASI TANDATANGAN DIGITAL DENGAN MENGGUNAKAN ALGORITMA ELGAMAL

Nama: Ricky Gilbert Fernando
NIM: 13505077

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail: if15077@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang studi dan implementasi tandatangan digital dengan menggunakan algoritma ElGamal. Algoritma ElGamal termasuk algoritma kriptografi kunci nirsimetri yang dibangun berdasarkan protokol pertukaran kunci Diffie-Hellman. Algoritma ini dibangun oleh Taher Elgamal pada 1984. Walaupun penciptanya, Taher Elgamal tidak mendaftarkan algoritmanya untuk paten, pemilik paten Diffie-Hellman mengharuskan algoritma ini untuk dikenakan paten mereka. Tidak ada alasan yang jelas kenapa orang menamakan algoritma ini sebagai "ElGamal" walaupun nama terakhir penciptanya, Elgamal, tidak memiliki huruf besar G.

Secara umum, algoritma ElGamal mendasarkan keamanan algoritmanya pada kesulitan untuk komputasi logaritma diskrit dan penggunaan bilangan besar sebagai kunci-kuncinya. Algoritma ElGamal digunakan dalam perangkat lunak GNU *Privacy Guard* yang gratis, beberapa versi dari PGP, dan kriptosistem lainnya. Algoritma tandatangan digital, walau merupakan skema tandatangan dan tidak melakukan enkripsi apapun, memiliki kesamaan dengan ElGamal dalam banyak hal. ElGamal bisa didefinisikan sebagai suatu grup siklis G. ElGamal merupakan sebuah contoh algoritma enkripsi kunci nirsimetris yang secara semantik aman untuk digunakan. Hal ini masih probabilistik, yang berarti sebuah *plaintext* dapat dienkripsi menjadi banyak *ciphertext*, dengan konsekuensi sebuah proses enkripsi ElGamal menghasilkan ukuran 2 (dua) kali lebih besar. Algoritma ElGamal tidak aman untuk serangan *chosen ciphertext*, sehingga biasanya algoritma-algoritma turunan dari ElGamal, seperti *Cramer-Shoup* kriptosistem, lebih sering digunakan.

Implementasi tandatangan digital dengan menggunakan algoritma kriptografi kunci nirsimetri biasanya dilakukan dengan cara menggunakan fungsi *hash* (*hash function*) yang digabungkan dengan algoritma kriptografi kunci nirsimetri tersebut. Dengan adanya tandatangan digital, dokumen biner yang diberikan pengirim kepada penerima dapat dijamin keaslian isi berkasnya, identitas pengirim, dan berkas telah diterima oleh penerima. Fungsi hash akan menghasilkan sebuah message digest yang secara unik menyatakan pesan tersebut dalam ukuran yang lebih kecil. Pembubuhan tandatangan digital dilakukan dengan cara mengenkripsi digest dengan kunci privat pengirim. Dengan cara tersebut, pengecekan keaslian dokumen dan pengirim dapat dilakukan.

Kata kunci: tandatangan digital, ElGamal, algoritma, studi, implementasi, kriptosistem.

1. Pendahuluan

Pertukaran data digital merupakan hal yang sering dilakukan pada saat ini. Data yang dipertukarkan dapat berupa teks (surat elektronik), audio, video, atau berkas biner lainnya. Dalam pengirimannya, berkas dapat dimanipulasi isinya oleh pihak ketiga sehingga pesan dengan makna yang berbeda akan diterima penerima dan tujuan pengiriman tidak tercapai. Oleh karena itu, dibutuhkan sebuah mekanisme untuk menjamin bahwa pesan yang diterima penerima, disebut sebagai Bob, memang benar berasal dari pengirim, disebut sebagai Alice.

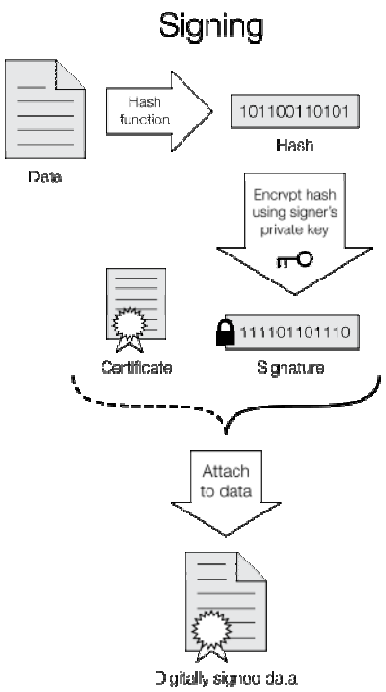
Tandatangan digital hanya dapat digunakan untuk sebuah dokumen atau tidak boleh ada reproduksi tandatangan digital pada dokumen yang berbeda. Hal ini cukup penting sehingga walaupun diketahui, tandantangan digital tidak dapat disalahgunakan oleh pihak ketiga, disebut sebagai Oscar, untuk menandatangani dokumen lain.

Terdapat sejumlah karakteristik tandatangan digital [MUN06], yaitu:
a) Tanda-tangan adalah bukti yang otentik.
b) Tanda tangan tidak dapat dilupakan.

c) Tanda-tangan tidak dapat dipindah untuk digunakan ulang.
d) Dokumen yang telah ditandatangani tidak dapat diubah.
e) Tanda-tangan tidak dapat disangkal(*repudiation*).

Bentuk tandatangan digital bukan merupakan hasil *scanning* tandatangan pengirim, tetapi merupakan parameter-parameter dari algoritma kunci publik. Algoritma kunci publik menggunakan 2 (dua) kunci yang berbeda untuk proses enkripsi dan dekripsinya. Ketika Alice akan mengenkripsi pesan, dia menggunakan kunci publik milik Bob, yang diketahui umum. Jika Bob ingin membaca pesan dari Alice, dia hanya perlu mendekripsi pesan tersebut dengan menggunakan kunci privat miliknya yang tidak boleh diketahui oleh orang lain. Pembangkitan kunci publik dapat didasarkan pada algoritma yang digunakan. Contoh algoritma kunci publik adalah RSA, ElGamal, Rabin, dan lain-lain.

Proses tandatangan digital dengan menggunakan kriptografi kunci publik dan fungsi *hash* berjalan dengan mengikuti skema berikut:

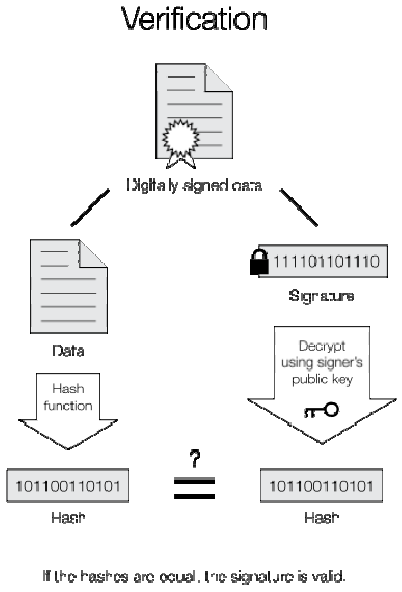


Gambar 1-1 Skema Pembubuhan Tandatangan Digital

Proses pembubuhan tandatangan digital sebagai berikut:

a) Dokumen yang akan dibubuhi dihitung terlebih dahulu nilai *hash*-nya, misalkan *h*.
b) Lalu nilai *hash h* tersebut dienkripsi dengan menggunakan kunci privat Alice menghasilkan tandatangan digital Alice yang unik untuk dokumen tersebut, misalkan *DS*.
c) Nilai *DS* diletakkan pada dokumen, bisa pada awal atau akhir dokumen.
d) Dokumen telah ditandatangan digital dan siap dikirim.

Ketika Bob menerima dokumen tersebut, Bob bisa saja membaca pesan tersebut, tetapi untuk menjamin bahwa pengirim dokumen tersebut memang benar Alice, Bob harus memastikannya. Berikut adalah skema *verification* dokumen yang telah ditandatangani:



Gambar 1-2 Skema *Verification* Dokumen

Proses verifikasi dokumen berjalan sebagai berikut:

a) Dokumen yang diterima Bob dapat dilihat sebagai 2 (dua) bagian, yaitu pesan dan tandatangan digital.
b) Bagian data dihitung nilai *hash*-nya.
c) Bagian tandatangan digital didekripsi dengan menggunakan kunci publik Alice.
d) Bandingkan nilai yang didapat pada b) dengan d). Jika kedua nilai tersebut sama, dokumen memang benar berasal dari Alice. Jika tidak, dokumen tidak berasal dari Alice atau dokumen telah diubah selama pengiriman.

2. Algoritma ElGamal

Mirip dengan algoritma RSA yang mendasarkan kekuatannya pada permasalahan teori bilangan yang sulit, algoritma ElGamal dibangun dengan mendasarkan kekuatannya pada sulitnya memecahkan logaritma diskrit.

2.1 Logaritma Diskrit

Ketika bekerja dengan bilangan *real*,

$$\log_b y$$

memiliki nilai x sehingga

$$b^x = y.$$

Diberikan sebuah nilai b dan n, dimana $b < n$, logaritma diskrit dari bilangan y dalam basis b adalah x, sehingga

$$b^x \equiv y \bmod n$$

Proses penghitungan di atas dapat dilakukan dengan cepat, tetapi proses kebalikannya, yaitu

$$x = \log_b y \bmod n$$

lebih sulit untuk dilakukan. Kriptosistem ElGamal mendasarkan kekuatannya pada kesulitan untuk menghitung proses logaritma diskrit tersebut.

2.2 Properti ElGamal

Didefinisikan sebuah kriptosistem ElGamal sebagai:

$$\kappa = \{(p,\alpha,a,\beta) : \beta \equiv \alpha^a \pmod p\}$$

Keterangan:

a) p, sebagai basis modulus dan merupakan bilangan prima.
b) α adalah elemen primitif dari Z_p*.
c) a adalah bilangan *random*, dimana $a < p$ dan bersifat rahasia.
d) β adalah bilangan *random*, dimana $\beta < p$.

2.3 Pembubuhan Tandatangan Digital

Proses pembubuhan tandatangan digital pada ElGamal berbeda dengan pembubuhan yang menggunakan algoritma RSA. Jika menggunakan RSA, berkas dihitung terlebih dahulu nilai *hash*-nya, lalu nilai *hash* tersebut dienkripsi dengan kunci privat pengirim. Tetapi pada ElGamal, telah disediakan sebuah mekanisme khusus untuk pembubuhan tandatangan digital, sehingga berkas tidak perlu dihitung nilai *hash*-nya, tetapi langsung mengikuti skema ElGamal yang telah tersedia,

Misalkan sebuah bilangan x akan ditandatangan digital, dengan menggunakan kriptosistem ElGamal dengan $\kappa = (p,\alpha,a,\beta)$ dan sebuah bilangan *random* $k \in Z_{p-1}$* yang bersifat rahasia, didefinisikan proses *signature* dokumen tersebut adalah sebagai berikut:

$$sig_\kappa(x,k) = (\gamma,\delta),$$

dimana

$$\gamma = \alpha^k \bmod p$$

dan

$$\delta = (x - a\gamma)k^{-1} \bmod (p-1).$$

Proses *signature* dengan kriptosistem ElGamal menghasilkan 2 (dua) bilangan γ dan δ. Berbeda dengan *signature* pada RSA, digital *signature* pada ElGamal mengembalikan keduanya.

Skema *signature* ElGamal bersifat non-deterministik, begitu pula dengan kriptosistem kunci publik ElGamal. Hal ini berarti terdapat banyak tandatangan digital yang valid untuk sebuah pesan masukan. Algoritma verifikasi harus mampu menerima semua masukan tandatangan yang valid sebagai tandatangan yang otentik.

2.4 Verifikasi Tandatangan Digital

Proses verifikasi tandatangan digital dengan algoritma ElGamal menerima 3 (tiga) masukan, yaitu pesan (x), γ, dan δ. Proses verifikasi dilakukan dengan mencocokan hasil dari 2 (dua) proses penghitungan yang dijelaskan dalam skema berikut:

$$ver_\kappa(x,\gamma,\delta) = true \Leftrightarrow \beta^\gamma \gamma^\delta \equiv \alpha^x \pmod p.$$

Skema verifikasi di atas dapat dibuktikan. Jika tandatangan digital dibangun dengan benar, verifikasi akan sukses berjalan sebab

$$\begin{aligned} \beta^\gamma \gamma^\delta &\equiv \alpha^{a\gamma} \alpha^{k\delta} \pmod p \\ &\equiv \alpha^x \pmod p, \end{aligned}$$

dimana digunakan fakta bahwa

$$a\gamma + k\delta \equiv x \pmod{p-1}.$$

2.5 Contoh Penggunaan

Diberikan p = 467, α = 2, a = 127, dapat dihitung

$$\begin{aligned} \beta &\equiv \alpha^a \pmod p \\ &\equiv 2^{127} \bmod 467 \\ &\equiv 132. \end{aligned}$$

Misalkan Alice ingin menandatangani pesan x = 100 dan dia memilih bilangan *random* k = 213 (perlu diperhatikan gcd(213,466) = 1 dan 213^{-1} mod 466 = 431). Maka

$$\gamma = 2^{213} \bmod 467 = 29$$

dan

$$\delta = (100 - 127 * 29)431 \bmod 466 = 51.$$

Semua orang dapat meverifikasi tandatangan ini dengan mengecek bahwa

$$132^{29}29^{51} \equiv 189 \pmod{467}$$

dan

$$2^{100} = 189 \pmod{467}$$

Sehingga tandatangan tersebut valid.

3. Implementasi ElGamal untuk Tandatangan Digital

Bab ini akan membahas mengenai implementasi skema tandatangan digital ElGamal, termasuk membahas mengenai lingkungan pengembangan, yaitu perangkat keras dan perangkat lunak. Perangkat lunak yang dikembangkan bernama ElGamalDS.

3.1 Perangkat keras

Perangkat keras yang digunakan dalam pengembangan perangkat lunak, yaitu:

a) Intel Q6600 @ 2.4GHz.
b) Memori 2GB.

3.2 Perangkat lunak

Perangkat lunak yang digunakan:

a) JDK 1.6.0u13 sebagai *software development kit*.
b) Notepad++ sebagai *editor*.

3.3 Implementasi ElGamal

Dalam pengembangannya, karena perangkat lunak dibangun dengan menggunakan bahasa Java, kelas utama yang menangani skema tandatangan digital ElGamal adalah ElGamalDS. Keseluruhan kelas dimasukkan dalam sebuah berkas bernama ElGamalDS.java.

Parameter yang dibangkitkan program berukuran 256 bit dengan tujuan untuk meningkatkan keamanan tandatangan digital. Karena bilangan yang digunakan termasuk bilangan besar dan tidak ada kelas primitif Java yang mampu menanganinya, digunakan sebuah kelas bantuan BigInteger yang mampu menangani penyimpanan bilangan-bilangan besar dan menyediakan fungsi-fungsi serta prosedur komputasi yang diperlukan.

Source code program dapat dilihat pada lampiran makalah atau dapat di-download pada situs penulis: (http://students.if.itb.ac.id/~if15077/web/elgamal/elgamal.zip) yang dapat diakses melalui internal ITB. Secara garis besar, program berjalan sebagai berikut:

a) Program membangkitkan parameter ElGamal, yaitu *p, α,* dan *a*.
b) Program membaca berkas yang ingin ditandatangani sebagai masukan dari *user*.
c) Program menandatangani berkas dengan menggunakan parameter yang telah dimiliki dan menghasilkan sebuah berkas baru yang berisi tandatangan digital dari berkas tersebut.
d) Program melakukan verifikasi tandatangan digital dengan cara membandingkan 2 (dua) buah proses komputasi dimana jika hasilnya sama, maka verifikasi berhasil dan jika tidak, gagal.

Berikut adalah deskripsi dari masing-masing kelas dalam ElGamalDS.java:

1. *class ElGamalDS*, berfungsi untuk menjalankan fungsi-fungsi utama ElGamal, seperti *sign* dan *verify*.
   Deskripsi *method* dalam *class ElGamalDS*:

Tabel 3-1 Deskripsi Class ElGamalDS

Nama Method	Param	Deskripsi
ElGamalDS	-	Konstruktor ElGamalDS yang menyediakan pembangkitan parameter.
ElGamalDS	*p, secretKey, alpha*	Konstruktor ElGamalDS dengan masukan user.
toString	-	Menampilkan isi kelas ElGamalDS dalam tampilan yang lebih mudah dimengerti.
sign	*message, filename*	*Method* untuk menandatangani berkas *filename* yang direpresentasikan dalam BigInteger *message*.
verify	*message*	*Method* untuk meverifikasi class ElGamal dengan BigInteger *message*.

2. class ReadFile, berfungsi untuk membuka sebuah berkas dan mengembalikan nilai *byte* dari berkas tersebut.

Tabel 3-2 Deskripsi Class ReadFile

Nama Method	Param	Deskripsi
readFile	*filename*	Membaca berkas *filename* dan mengembalikan representasi berkas dalam *array of byte*.

4. Analisis Algoritma ElGamal

Analisis performansi algoritma ElGamal mirip dengan algoritma RSA yang termasuk cepat dalam proses penandatanganan dan verifikasi. Diberikan sebuah *n* sebagai modulus yang

digunakan dalam kriptosistem ElGamal. Masing-masing proses dalam ElGamal membutuhkan *O(log n)* operasi aritmatik.

Untuk mendapatkan tingkat keamanan yang cukup tinggi, panjang bit *p* yang digunakan minimal 256 *bit* supaya proses logaritma diskritnya lebih sulit untuk dipecahkan. Penggunaan panjang *bit p* dapat disesuaikan dengan keperluan tandatangan digital. Jika dokumen yang ingin ditandatangani adalah berkas dengan tingkat privasi rendah, *p* dapat sepanjang 64 hingga 128 *bit* dengan pertimbangan untuk kecepatan penandatanganan dan verifikasi.

Tingkat keamanan skema tandatangan digital ElGamal termasuk tinggi. Misalkan Oscar berusaha untuk membubuhkan tandatangan digital ke sebuah pesan *x*, tanpa mengetahui *a*. Jika Oscar memilih sebuah nilai γ dan berusaha untuk menemukan δ yang bersesuaian, Oscar harus melakukan komputasi logaritma diskrit:

$$log_{\gamma} \alpha^{x}\beta^{-\gamma}.$$

Di lain hal, jika Oscar pertama memilih δ dan mencoba untuk menemukan γ, Oscar akan mencoba untuk menyelesaikan persamaan

$$\beta^{\gamma} \gamma^{\delta} \equiv \alpha^{x} \ (mod\ p)$$

untuk γ yang tidak diketahui. Masalah tersebut tidak memiliki solusi yang dapat dipecahkan. Tetapi, hal terlihat tidak berkaitan dengan masalah yang biasanya dipelajari, seperti masalah Logaritma Diskrit. Tetap terdapat kemungkinan dimana terdapat sebuah cara untuk menghitung γ dan δ dengan sebuah cara sehingga (γ,δ) menjadi tandatangan digitalnya. Tidak ada seorangpun yang pernah menemukan sebuah cara untuk melakukannya, tetapi tidak seorang juga yang pernah membuktikan bahwa hal tersebut tidak dapat dilakukan.

Jika Oscar memilih γ dan δ lalu mencoba untuk menemukan *x*, Oscar kembali menghadapi sebuah kasus dari masalah Logaritma Diskrit dalam bentuk komputasi

$$log_{\alpha} \beta^{\gamma}\gamma^{\delta}.$$

Oleh karena itu, Oscar tidak bisa menandatangani pesan random dengan menggunakan cara ini.

Tetapi terdapat sebuah cara dimana Oscar dapat memalsukan tandatangan digital untuk sebuah pesan random dengan memilih γ, δ, dan *x* berurutan. Misalkan *i* dan *j* adalah bilangan, dimana $0 \le i \le p-2$, $0 \le j \le p-2$, dan gcd(*j*,*p-1*) = 1. Lalu lakukan komputasi berikut:

$$\gamma = \alpha^{i}\beta^{j} \ (mod\ p)$$
$$\delta = -\gamma j^{-1} \ (mod\ p-1)$$
$$x = -\gamma i^{j-1} \ (mod\ p-1),$$

dimana j^{-1} dikomputasi dalam mod (p-1). Oscar dapat mengklaim (γ,δ) sebagai tandatangan yang valid untuk pesan *x*. Hal ini dapat dibuktikan dengan pengecekan verifikasi:

$$\begin{aligned} \beta^{\gamma}\gamma^{\delta} &\equiv \beta^{\gamma}(\alpha^{i}\beta^{j})^{-\gamma j-1} \ (mod\ p) \\ &\equiv \beta^{\gamma}\alpha^{-i\gamma j-1}\beta^{-\gamma} \ (mod\ p) \\ &\equiv \alpha^{-i\gamma j-1} \ (mod\ p) \\ &\equiv \alpha^{x} \ (mod\ p) \end{aligned}$$

Cara tersebut memang menghasilkan tandatangan palsu yang valid, tetapi tidak ada cara yang memungkinkan lawan untuk memalsukan tandatangan untuk sebuah pesan yang diinginkannya tanpa memecahkan problem logaritma diskrit. Oleh karena itu, tidak terdapat ancaman untuk keamanan skema tandatangan digital ElGamal.

Pada subbab 2.3 Pembubuhan Tandatangan Digital, telah dijelaskan bahwa variabel *k* bersifat rahasia. Jika *k* diketahui, Oscar dapat langsung menghitung nilai *a*:

$$a = (x - k\delta)\ \gamma^{-1} \ (mod\ p-1)$$

sehingga sistem telah terpecahkan dan Oscar dapat memalsukan tandatangan.

Salah penggunaan dari sistem, seperti menggunakan nilai *k* yang sama untuk menandatangani pesan yang berbeda dapat menyebabkan sistem terpecahkan.

5. Kesimpulan dan Saran

Bab ini akan berisi kesimpulan dan saran yang didapat setelah mengembangkan program ElGamalDS.

5.1 Kesimpulan

Kesimpulan yang dapat ditarik:

a) ElGamal merupakan algoritma yang cocok untuk skema tandatangan digital.
b) ElGamal termasuk algoritma yang cepat dalam komputasinya.
c) ElGamal aman digunakan selama tidak terjadi kesalahan penggunaan dan pembocoran informasi rahasia sistem ElGamal.

d) Ukuran tandatangan pada ElGamal 2 (dua) kali ukuran tandatangan digital dengan RSA.

5.2 Saran

Saran yang dapat diberikan:

a) Kerahasiaan data-data sensitif dalam kriptosistem harus selalu dijaga agar kriptosistem tidak dapat dipecahkan.
b) Untuk memperkecil ukuran tandatangan digital hasil ElGamal, dapat dilakukan penghitungan *hash* sehingga didapat *message digest* yang lebih kecil.

DAFTAR PUSTAKA

[NIS09] http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf, waktu akses: 17 Mei 2009, 13:00 WIB. Durasi: 1 jam.

[MUN06] Munir, Rinaldi. 2006. Diktat Kuliah IF5054 Kriptografi.

[STI95] Stinson, Douglas Robert. 1995. *Cryptography: Theory and Practice*. Florida: CRC Press.

LAMPIRAN
SOURCE CODE

```
// File             : ElGamalDS.java
// Creator          : Ricky Gilbert F
// NIM              : 13505077
// E-mail: ricky.gilbert.f@gmail.com
// Deskripsi:
//                  Program ini bertugas untuk me-generate sebuah file yang merupakan tandatangan digital pengguna.
//                  User dapat me-set parameter ElGamal yang telah dimiliki atau dapat me-generate baru parameternya
//                  dengan menggunakan constructor yang telah disediakan.
//                  Source code ini dapat di-download di:
//                          http://students.if.itb.ac.id/~if15077/web/elgamal/elgamal.zip
//

import java.math.*;
import java.util.*;
import java.security.*;
import java.io.*;

public class ElGamalDS {
        private BigInteger p;                   // mod p
        private BigInteger alpha;               // variabel alpha
        private BigInteger beta;                // variabel beta
        private BigInteger secretKey;           // variabel a
        private byte[] resFile;                 // menyimpan nilai byte dr sebuah file
        private BigInteger r;                   // variabel random secret r
        private final int BIT_DEFAULT = 128;

        private BigInteger gamma;
        private BigInteger lambda;
        private Random sc = new SecureRandom();
  private Scanner scan = new Scanner(System.in);

        public ElGamalDS() {
                p = BigInteger.probablePrime(BIT_DEFAULT, sc);
    do {
        secretKey = new BigInteger(BIT_DEFAULT, sc);
    } while (secretKey.compareTo(p)>-1);
    alpha = new BigInteger(BIT_DEFAULT/2,sc);
    beta = alpha.modPow(secretKey,p);
        }

        public ElGamalDS(BigInteger p, BigInteger secretKey, BigInteger alpha) {
                this.p = new BigInteger(p.toString());
                this.secretKey = new BigInteger(secretKey.toString());
                this.alpha = new BigInteger(alpha.toString());
                this.beta = this.alpha.modPow(this.secretKey,this.p);
        }

        public String toString() {
                String res;
                res = "secretKey = " + secretKey + "\n";
    res += "p        = " + p + "\n";
    res += "alpha    = " + alpha + "\n";
    res += "beta     = " + beta + "\n";
    return res;
        }

        public void sign(BigInteger message, String filename) {
                do {
```

```
            r = new BigInteger(BIT_DEFAULT, sc); // random number k
        } while (r.gcd(p.subtract(BigInteger.ONE)).compareTo(BigInteger.ONE)!=0);

        gamma = alpha.modPow(r,p);
        lambda                                                                                                    =
((message.subtract(secretKey.multiply(gamma))).multiply(r.modInverse(p.subtract(BigInteger.ONE))))
.mod(p.subtract(BigInteger.ONE));
        System.out.println
("=================================================================");
        System.out.println ("*                    SIGN                    *");
        System.out.println
("=================================================================");
        System.out.println("Plaintext
                    = " + message);
        System.out.println("r
                            = " + r);
        System.out.println("GAMMA(alpha^k mod p)
            = " + gamma);
        System.out.println("LAMBDA[ (x-secretKey*gamma)*kinv mod (p-1) ]     = " + lambda);

        FileWriter fstream;
        BufferedWriter out;
        try {
            fstream = new FileWriter(filename+".ds");
            out = new BufferedWriter(fstream);
                        out.write(gamma.toString()+"<>");
                        out.write(lambda.toString());
                        out.flush();
            fstream.close();
            out.close();
        } catch (IOException e) {
            System.out.println ("IOException caught: " + e.getMessage());
        }
            }

            public void verify(BigInteger message) {
                    BigInteger ver1 = alpha.modPow(message,p);
        BigInteger ver2 = beta.modPow(gamma,p).multiply(gamma.modPow(lambda,p)).mod(p);
        System.out.println
("=================================================================");
        System.out.println ("*                    VERIFY                    *");
        System.out.println
("=================================================================");
        System.out.println ("Verification 1 = " + ver1);
                    System.out.println ("Verification 2 = " + ver2);
                    if (ver1.compareTo(ver2)==0) System.out.println ("Verifikasi berhasil...");
                    else System.out.println ("Verifikasi gagal...");
            }

    public static void main(String[] args) {
        ElGamalDS eg = new ElGamalDS();

        try {
                    //
                    // Read File
                    //
                    ReadFile rf = new ReadFile();
                    if (args.length==0) {
                        System.out.println ("Masukan nama file sebagai parameter program!");
                        System.exit(0);
```

```
                }
                eg.resFile = rf.readFile(args[0]);

                //
                // print status ElGamalDS
                //
                System.out.println (eg);

                //
                // Signature
                //
                BigInteger X = new BigInteger(eg.resFile);
                eg.sign(X,args[0]);

                //
                // Verification
                //
                eg.verify(X);

        } catch (IOException e) {
            System.out.println ("IOException caught: " + e.getMessage());
        } catch (Exception e) {
            System.out.println ("Exception caught: " + e.getMessage());
        }
    }
}

class ReadFile {
        public byte[] readFile(String filename) throws IOException {
                File file = new File(filename);
        DataInputStream dis = new DataInputStream(new FileInputStream(file));
        System.out.println ("Opening file " + filename);
        System.out.print ("File length = " + file.length() + " : ");
        int size = (int)file.length();
        if (size > Integer.MAX_VALUE){
            System.out.println("File is too large");
            System.out.println("Max allowed : " + Integer.MAX_VALUE);
            System.exit(0);
            } else {
                System.out.println ("OK");
            }
        int read = 0;
        int numRead = 0;
        byte res[] = new byte[size];
        while (read < res.length && (numRead=dis.read(res, read, res.length-read)) >= 0) {
            read = read + numRead;
            }
        dis.close();
        return res;
            }
}
```