BAB 1
PENDAHULUAN

1.1. Latar Belakang Masalah

Saat ini komputer hampir dapat dijumpai di setiap kantor pemerintah, perusahaan, sekolah, atau bahkan rumah tangga. Perkembangan teknologi komputer yang pesat, khususnya di bidang perangkat lunak, membuat komputer menjadi semakin *user friendly* dan telah menjadikannya suatu kebutuhan bagi kalangan tertentu, misalnya kalangan bisnis. Dalam melakukan pekerjaan mereka sangat tergantung pada komputer. Komputer tidak lagi hanya digunakan sebagai pengganti mesin tik ataupun alat hitung, namun kini juga banyak digunakan dalam membantu pembuatan keputusan penting. Akibatnya, informasi yang disimpan memerlukan pengamanan yang dapat melindungi terhadap akses orang yang tidak berhak.

Salah satu cara yang dapat dilakukan untuk melindungi informasi tersebut adalah dengan menggunakan enkripsi. Enkripsi adalah suatu cara atau proses untuk menyandikan/mengkodekan data/informasi ke dalam suatu bentuk untuk menyembunyikan substansinya.

1.2. Perumusan Masalah

Yang menjadi permasalahan dalam tugas akhir ini adalah "Bagaimana mengimplementasikan *Elliptic Curve Cryptosystem* dalam sebuah program enkripsi/dekripsi ?"

1.3. Manfaat

Dengan menggunakan program yang dikembangkan dalam tugas akhir ini, pemakai akan memperoleh manfaat sebagai berikut :

- Pemakai dapat mengamankan datanya sehingga tidak dapat diakses oleh orang yang tidak berhak.
- Pemakai tidak perlu mempelajari tata cara pemakaian program yang rumit, karena hanya terdapat beberapa perintah yang mudah diingat.

- Pemakai tidak perlu membayar untuk dapat menggunakan program ini.
- Kunci untuk dekripsi/enkripsi berupa *passphrase*, sehingga memudahkan pemakai mengingatnya.

1.4. Tujuan

Tujuan pembuatan tugas akhir ini adalah:

a. Menerapkan Elliptic Curve Cryptosystem ke dalam sebuah program enkripsi/dekripsi.
b. Meneliti kecepatan program dengan menggunakan kurva eliptik berukuran 160-bit dan 192-bit pada berbagai konfigurasi sistem komputer.
c. Meneliti perubahan ukuran berkas sebelum dan setelah enkripsi dengan menggunakan kurva eliptik yang berukuran 160-bit dan 192-bit.

1.5. Hipotesis

Adapun yang menjadi hipotesis pada tugas akhir ini adalah :

1. Elliptic Curve Cryptosystem dapat diimplementasikan untuk membuat program enkripsi/dekripsi.
2. Proses enkripsi/dekripsi pada ECC 160-bit lebih cepat daripada proses enkripsi/dekripsi pada ECC 192-bit.

1.6. Spesifikasi

Spesifikasi tugas akhir yang dibuat ini adalah :

1. Program dibuat berbasiskan bahasa pemrograman C/C++.
2. Program dibuat dengan menggunakan *compiler* Microsoft Visual C++ 6.0.
3. Menggunakan *Library for computational number theory* LiDIA versi 1.3.3 (atau yang lebih baru, dari TU-Darmstadt, Jerman).
4. Data yang diproses berupa data teks dan biner.
5. Data yang akan dienkripsi/didekripsi dibaca per blok.
6. Berkas yang telah dienkripsi/didekripsi ditulis ke berkas lain.
7. Kunci yang dimasukkan pemakai berupa *passphrase* yang kemudian akan di-*hash* dengan menggunakan MD5.

1.7. Metodologi Penelitian

Metode penelitian yang digunakan dalam pembuatan tugas akhir ini adalah:

1. Pustaka, yaitu mempelajari referensi yang berhubungan dengan kriptografi, MD5, bahasa Visual C++ 6.0, dan *elliptic curve cryptosystem* dari buku-buku, majalah, jurnal, maupun Internet.
2. Percobaan, yaitu dengan menguji program yang dihasilkan ke berbagai konfigurasi sistem komputer untuk mengetahui kecepatan program.

1.8. Sistematika Penulisan

Sistematika penulisan tugas akhir ini adalah sebagai berikut:

Bab 1. Pendahuluan

Bab ini meliputi latar belakang masalah, perumusan masalah, manfaat tugas akhir, tujuan tugas akhir, teori singkat, hipotesis, spesifikasi, metodologi penelitian, dan sistematika penulisan tugas akhir yang akan dibuat.

Bab 2. Dasar Teori

Bab ini akan menguraikan dasar teori yang berkaitan dengan kriptografi dan *elliptic curve cryptosystem*.

Bab 3. Algoritma dan Desain

Bab ini berisi algoritma dan desain program yang akan dibuat.

Bab 4. Implementasi dan Analisis

Dalam bab ini dijelaskan implementasi program yang dibuat, meliputi cara pemakaian, pengujian proses enkripsi / dekripsi serta analisisnya.

Bab 5. Penutup

Bab ini berisi kesimpulan dan saran yang dapat diambil dari pembuatan tugas akhir ini.

BAB 2
DASAR TEORI

Teknologi enkripsi yang dulunya hanya dapat diakses oleh dunia militer namun kini telah pula dapat digunakan oleh kalangan non militer. Hal ini membawa dampak semakin banyak digunakannya enkripsi sebagai salah satu cara mengamankan data.

2.1. Enkripsi dan Dekripsi

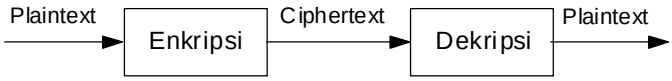
Enkripsi adalah suatu proses untuk menyamarkan sebuah pesan (*message*) sedemikian rupa untuk menyembunyikan pesan aslinya dengan menggunakan sebuah kunci. Dekripsi adalah kebalikan dari enkripsi, yaitu proses pengembalian sebuah pesan terenkripsi menjadi pesan aslinya.

Pada enkripsi dikenal dua buah istilah penting, yaitu *plaintext* dan *ciphertext*. *Plaintext* adalah pesan yang dienkripsi, sedangkan *ciphertext* adalah pesan hasil enkripsi. *Plaintext* biasa disimbolkan sebagai M (*Message*) atau P (*Plaintext*), yang dapat berupa suatu aliran bit, berkas teks, berkas bitmap, berkas suara digital, ataupun berkas video digital. Sejauh yang diketahui komputer, M adalah data biner. Sedangkan *ciphertext* biasanya disimbolkan sebagai C (*Ciphertext*), dan juga merupakan data biner.

Jika enkripsi disimbolkan sebagai fungsi E (*Encryption*) dan dekripsi disimbolkan sebagai fungsi D (*Decryption*), maka dengan menggunakan notasi matematika, enkripsi dan dekripsi dapat ditulis sebagai berikut :

$E(P) = C$
$D(C) = P$

Proses enkripsi dan dekripsi pada *plaintext* dan *ciphertext* dapat dilihat pada Gambar 2.1.



Gambar 2.1. Proses enkripsi dan dekripsi[1]

[1]Bruce Schneier, Applied Cryptography, 2nd edition, New York: John-Wiley & Sons, 1997, hlm. 1.

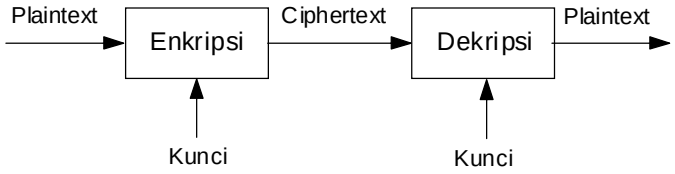
2.2. Algoritma dan Kunci

Kriptografi[2] menggunakan kunci (*key*) untuk melakukan proses enkripsi dan dekripsi. Dengan demikian algoritma kriptografi (*cipher*) yang digunakan dapat dipublikasikan. Kunci (disimbolkan sebagai K) pada kriptografi berupa satu nilai dari sejumlah bilangan yang banyak jumlahnya. Daerah jangkauan nilai-nilai kunci yang mungkin disebut *keyspace*.

Dengan adanya penggunaan kunci, maka notasi matematika untuk fungsi enkripsi dan dekripsi dapat ditulis sebagai berikut :

$$E_K(M) = C$$
$$D_K(C) = M$$

Gambar 2.2 memperlihatkan proses enkripsi dan dekripsi dengan menggunakan kunci.



Gambar 2.2. Enkripsi dan Dekripsi dengan Kunci[3]

Pada kriptografi, kunci yang dimasukkan pemakai (*user key*) sering dinyatakan sebagai sebuah *password* ataupun sebagai sebuah *passphrase*. Penggunaan *passphrase* untuk menyatakan suatu kunci pemakai lebih baik daripada menggunakan *password*[4], karena *passphrase* umumnya memiliki ukuran yang tidak dibatasi dan dapat berupa kalimat sehingga mudah diingat oleh pemakai. Sementara *password* umumnya memiliki ukuran yang tetap.

Berdasarkan penggunaan kunci untuk proses enkripsi dan dekripsinya, algoritma kriptografi dapat dibedakan menjadi dua yaitu: *symmetric algorithm* (*secret key algorithm*) dan *asymmetric algorithm* (*public key algorithm*).

- *Symmetric Algorithm*

Symmetric algorithm atau disebut *juga secret key algorithm* adalah algoritma yang kunci enkripsinya dapat dihitung dari kunci dekripsi dan begitu pula sebaliknya, kunci dekripsi dapat dihitung dari kunci enkripsi. Pada sebagian

[2] Kriptografi (*cryptography*) adalah seni dan ilmu pengetahuan untuk menjaga keamanan suatu pesan.
[3] *Ibid*.

besar *symmetric algorithm* kunci enkripsi dan kunci dekripsi adalah sama. *Symmetric algorithm* memerlukan kesepakatan antara pengirim dan penerima pesan pada suatu kunci sebelum dapat berkomunikasi secara aman. Keamanan *symmetric algorithm* tergantung pada rahasia kunci. Pemecahan kunci berarti memungkinkan setiap orang dapat mengenkripsi dan mendekripsi pesan dengan mudah.

Symmetric algorithm dapat dikelompokkan menjadi dua jenis, yaitu *stream cipher* dan *block cipher*. *Stream cipher* beroperasi bit per bit (atau byte per byte) pada satu waktu. Sedangkan *block cipher* beroperasi per kelompok-kelompok bit yang disebut blok (*block*) pada satu waktu.

- *Asymmetric Algorithm*

Asymmetric algorithm atau disebut juga *public key algorithm* didesain agar memudahkan dalam distribusi kunci yang digunakan untuk enkripsi dan dekripsi. Kunci dekripsi pada *public key algorithm* secara praktis tidak dapat dihitung dari kunci enkripsi. Algoritma ini disebut "*public key*" karena kunci dapat dibuat menjadi publik. Setiap orang dapat menggunakan kunci enkripsi untuk mengenkripsi pesan, tetapi hanya orang yang memiliki kunci dekripsi yang dapat mendekripsi pesan tersebut. Pada sistem ini kunci enkripsi sering disebut kunci publik (*public key*), dan kunci dekripsi disebut kunci rahasia (*private key*).

2.3. *One-Way Hash Function*

Suatu *hash function* adalah sebuah fungsi matematika, yang mengambil sebuah panjang variabel string input, yang disebut *pre-image* dan mengkonversikannya ke sebuah string output dengan panjang yang tetap dan biasanya lebih kecil, yang disebut *message digest*[5].

Hash function digunakan untuk melakukan *fingerprint* pada *pre-image*, yaitu menghasilkan sebuah nilai yang dapat menandai (mewakili) *pre-image* sesungguhnya.

---

[4] *Ibid.*, hlm. 174.
[5] *Ibid.*, hlm. 30.

Fungsi *hash* satu arah (o*ne-way hash function*) adalah *hash function* yang bekerja satu arah, yaitu suatu *hash function* yang dengan mudah dapat menghitung *hash value* dari *pre-image*, tetapi sangat sukar untuk menghitung *pre-image* dari *hash value*.

Sebuah fungsi *hash* satu arah, `H(M)`, beroperasi pada suatu *pre-image* pesan `M` dengan panjang sembarang, dan mengembalikan nilai *hash* `h` yang memiliki panjang tetap. Dalam notasi matematika fungsi *hash* satu arah dapat ditulis sebagai:

`h = H(M), dengan h memiliki panjang b`

Ada banyak fungsi yang mampu menerima input dengan panjang sembarang dan menghasilkan output dengan panjang tetap, tetapi fungsi *hash* satu arah memiliki karakteristik tambahan yang membuatnya satu arah[6] :

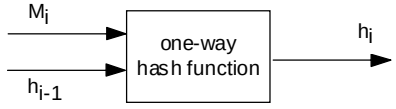
```
Diberikan M, mudah menghitung h.
Diberikan h, sulit menghitung M agar H(M) = h.
Diberikan M, sulit menemukan pesan lain, M', agar H(M) = H(M').
```

Dalam dunia nyata, fungsi *hash* satu arah dikembangkan berdasarkan ide sebuah fungsi kompresi. Fungsi satu arah ini menghasilkan nilai *hash* berukuran `n` bila diberikan input berukuran `b`. Input untuk fungsi kompresi adalah suatu blok pesan dan hasil blok teks sebelumnya. Sehingga *hash* suatu blok M, adalah

$$h_i = f(M_i, h_{i-1})$$

dengan

h_i = nilai *hash* saat ini.
M_i = blok pesan saat ini.
h_{i-1} = nilai *hash* blok teks sebelumnya.



Gambar 2.3. One-way Hash Function[7]

One-way hash function yang digunakan di sini (MD5) menghasilkan *hash value* berukuran 128-bit. Namun bila dalam aplikasinya diperlukan *hash value*

[6]*Ibid.*, hlm. 429.
[7]*Ibid.*, hlm. 430.

berukuran lebih dari 128 bit dapat digunakan metode berikut untuk menghasilkan *hash value* yang lebih besar :[8]

1. Buat *hash value* dari sebuah pesan, menggunakan sebuah *one-way hash function.*
2. Tambahkan sebuah string ke dalam pesan.
3. Buat *hash value* dari penggabungan (*concatenation*) pesan dan string itu.
4. Buat *hash value* yang lebih besar yang berisi gabungan antara *hash value* yang dihasilkan dari langkah (1) dengan *hash value* yang dihasilkan dari langkah (3).
5. Ulangi langkah (1) sampai (3) sebanyak yang diinginkan.

2.4. MD5[9]

Program yang dibuat ini menggunakan MD5 sebagai *one-way hash function.* Algoritma MD5 didesain agar berjalan cukup cepat pada mesin 32-bit. Selain itu algoritma MD5 tidak memerlukan tabel substitusi yang besar, algoritma ini dapat dikodekan secara cukup kompak.

Algoritma ini merupakan pengembangan algoritma *one-way hash function* MD4. MD5 sedikit lebih lambat daripada MD4, namun lebih konversatif dalam desain. Oleh karena MD4 dirancang agar berjalan cepat, ia memiliki risiko terhadap serangan *cryptanalysis*. Sedangkan MD5 memiliki kecepatan yang sedikit berkurang dengan peningkatan keamanan.

2.5. *Finite Field*[10]

Suatu *finite field* (atau Galois Field) adalah suatu himpunan dengan jumlah elemen terbatas dan dapat dilakukan operasi-operasi aljabar —seperti pengurangan, perkalian, penambahan, dan pembagian dengan elemen tidak nol— terhadap elemen-elemen yang ada. Selain itu dalam *finite field*, hukum-hukum aljabar seperti asosiatif, komutatif, dan distributif berlaku.

---

[8] *Ibid.*

[9] Ronald L. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, MIT dan RSA Data Security, Inc., April 1992.

[10] Robert Hofer, "Finite Fields", TU-Graz, 1999.
(http://www.sbox.tu-graz.ac.at/home/j/jonny/project/crypto/math/finite.htm)

Orde *finite field* menyatakan banyaknya elemen yang terdapat di dalamnya. Untuk menyatakan *finite field* berorde q dapat ditulis dengan F_q atau GF(q), dengan q dapat berupa bilangan prima atau prima^n. Ketika q merupakan suatu bilangan prima, maka field GF(p) disebut *finite prime field*. Field GF(p) umumnya dinyatakan sebagai himpunan integer modulo p[†].

Dimisalkan p adalah suatu bilangan prima. *Finite field* F_p terdiri dari sekumpulan integer {0,1, 2, ..., p-1} dengan operasi-operasi aritmatika sebagai berikut :

- Penambahan (*addition*) : Jika a, b $\in$ F_p maka a + b = r, dengan r adalah sisa pembagian ketika a+b dibagi dengan p dan $0 \leq r \leq p-1$. Operasi ini dikenal sebagai operasi penambahan modulo p.
- Pengurangan (*subtraction*) : Jika a, b $\in$ F_p maka a - b = r, dengan r adalah sisa pembagian ketika a-b dibagi dengan p bila a-b > 0, sedangkan bila a-b < 0, r adalah (a-b) + p. Operasi ini dikenal sebagai operasi pengurangan modulo p.
- Perkalian (*multiplication*) : Jika a, b $\in$ F_p maka a * b = s, dengan s adalah sisa pembagian ketika a*b dibagi dengan p dan $0 \leq s < p$. Operasi ini dikenal sebagai operasi perkalian modulo p.
- Inversi (*inversion*) : Jika a adalah elemen tidak nol dalam F_p, inversi a modulo p, yang dinyatakan sebagai a^{-1}, adalah suatu integer unik c $\in$ F_p dengan a*c = 1 mod p.

Sebuah contoh *Finite Field* adalah F_{13}, elemen-elemen *finite field* ini adalah {0,1, 2, ... ,12}. Contoh operasi aritmatika dalam field ini adalah :

- 10 + 5 = 15 mod 13 = 2.
- 3 - 5 = -2 = -2 + (1)*13 = 11 mod 13.
- 2 * 12 = 24 mod 13 = 11.
- 7^{-1} = 2 mod 13.

Berikut ini akan dijelaskan sekilas mengenai operasi inversi. Operasi inversi adalah operasi untuk mencari suatu bilangan integer unik c dalam F_p yang menjadikan a*c = 1 mod p. Dalam contoh di atas, ingin diketahui inversi dari 7,

[†] integer modulo p adalah sisa pembagian bila integer tersebut dibagi p.

bila persamaan tersebut dirubah ke dalam bentuk matematis adalah sebagai berikut :

$$7*c \bmod 13 = 1$$

Dengan cara mencoba-coba (*trial and error*) ditemukan bahwa c = 2.

Tentu saja metode coba-coba yang digunakan di atas sangat tidak memuaskan karena relatif lambat dan mudah salah terlebih untuk bilangan yang besar. Untuk itu dapat digunakan *Extended Euclidian Algorithm*. Suatu bilangan A merupakan inverse bilangan N bila faktor persekutuan terbesar keduanya adalah 1 atau GCD (A, N) = 1.

Algoritmanya adalah sebagai berikut [11]:

Diberikan bilangan integer positif u dan v, algoritma ini menentukan suatu vektor (u_1,u_2,u_3) sehingga $uu_1 + vu_2 = u_3 = gcd\ (u,v)$. Perhitungan menggunakan vektor bantuan (v_1,v_2,v_3), (t_1,t_2,t_3); seluruh vektor dimanipulasi sehingga hubungan

$$ut_1+vt_2 = t_3, uu_1+vu_2=u_3, \quad uv_1+vv_2=v_3$$

berlaku di seluruh perhitungan.

Input : u dan v
Output : gcd (u,v)

1. Set $(u_1,u_2,u_3) \leftarrow (1,0,u)$
   Set $(v_1,v_2,v_3) \leftarrow (0,1,v)$
2. if v3 = 0 then berhenti.
3. Set q $\leftarrow \lfloor u_3 / v_3 \rfloor$ †
   $(t_1,t_2,t_3) \leftarrow (u_1,u_2,u_3) - (v_1,v_2,v_3) *q$ ‡
   $(u_1,u_2,u_3) \leftarrow (v_1,v_2,v_3)$
   $(v_1,v_2,v_3) \leftarrow (t_1,t_2,t_3)$
4. Kembali ke langkah 2.

Sebagai contoh, misalkan u = 40902, v = 24140. Pada langkah 2 didapat :

q	u1	u2	u3	v1	v2	v3
-	1	0	40902	0	1	24140
1	0	1	24140	1	-1	16762

[11] Donald E. Knuth, The Art of Computer Programming, Vol. 2, New York: Addison-Wesley Publishing, Inc., 1981, hlm. 325.

† $\lfloor x \rfloor$ = integer terbesar kurang dari atau sama dengan bilangan real x. Contoh, $\lfloor 5,3 \rfloor = 5$.

‡ $(t_1,t_2,t_3) \leftarrow (u_1,u_2,u_3) - (v_1,v_2,v_3) *q$, memiliki arti $t_1 \leftarrow (u_1-v_1*q)$, $t_2 \leftarrow (u_2-v_2*q)$, $t_3 \leftarrow (u_3-v_3*q)$.

1	1	-1	16762	-1	2	7378
2	-1	2	7378	3	-5	2006
3	3	-5	2006	-10	17	1360
1	-10	17	1360	13	-22	646
2	13	-22	646	-36	61	68
9	-36	61	68	337	-571	34
2	337	-571	34	-710	1203	0

Solusinya adalah 337*40902 - 571*24140 = 34 = gcd(40902, 24140).

2.6. *Elliptic Curve Cryptosystem*

Kurva-kurva eliptik (*elliptic curves*) telah dipelajari secara intensif dalam bidang teori bilangan dan geometri aljabar oleh para ahli matematika selama lebih dari satu abad. Teori-teori telah banyak dikembangkan mengenai mereka, dan mereka telah menjadi dasar bagi perkembangan baru dalam ilmu matematika, pembuktian teorema terakhir Fermat (*Fermat's Last Theorem*)[12].

Elliptic Curve Public Key Cryptosystems (ECPKC)[13] diusulkan secara independen masing-masing oleh Victor Miller dan Neil Koblitz pada tahun 1985. Sejak tahun tersebut, ECPKC telah dievaluasi secara menyeluruh oleh para ahli kriptografi, ahli matematika, dan ahli komputer di seluruh dunia, sehingga timbul kepercayaan terhadap sistem baru ini. Beberapa tahun terakhir ini, implementasi komersial pertama telah muncul, baik sebagai *toolkit* maupun sebagai aplikasi seperti keamanan email, keamanan web, kartu pintar, dan sebagainya [14].

ECPKC ini memiliki beberapa keunggulan dibandingkan sistem yang serupa seperti RSA antara lain [15]: untuk tingkat keamanan yang ekivalen, ECPKC memerlukan ukuran kunci yang lebih kecil, akibatnya kecepatan lebih tinggi, konsumsi daya yang lebih rendah, adanya penghematan *bandwidth*.

---

[12] Erik de Win dan Bart Preneel, "Elliptic Curve Public Key Cryptosystems - an introduction". (http://cosic.esat.kuleuven.be/~preneel/)
[13] Selanjutnya ECPKC disebut *Elliptic Curve Cryptosystem* (ECC)
[14] *Ibid.*
[15] Alfred Menezes, "Elliptic Curve Cryptosystems—Ready for Prime Time", *Dr. Dobb's Journal*, January, 1998.

Keuntungan-keuntungan tersebut sangat berguna untuk aplikasi-aplikasi yang memiliki keterbatasan pada bandwidth, kapasitas pemrosesan, ketersediaan sumber tenaga, dan ruang. Aplikasi-aplikasi tersebut antara lain : kartu chip, *electronic commerce*, server web, telepon seluler, dan *pager*[16].

2.6.1. Kurva Eliptik dalam Fp

Jika $p > 3$ adalah suatu bilangan prima dan $a, b \in F_p$ dengan

$$4a^3 + 27b^2 \neq 0 \qquad (2.1)$$

dalam F_p. Suatu kurva eliptik $E(F_p)$ dalam F_p yang didefinisikan oleh parameter a dan b adalah himpunan penyelesaian (x,y), $x,y \in F_p$, yang memenuhi :

$$y^2 = x^3 + ax + b \qquad (2.2)$$

bersama dengan suatu *point* khusus, φ, yang disebut *point at infinity*.

Contoh suatu kurva eliptik dalam F_{13} :

$y^2 = x^3 + x + 1$ adalah persamaan untuk kurva eliptik E dalam F_{13}. Di sini a=1 dan b=1. Himpunan penyelesaian untuk persamaan tersebut adalah :

(0,1) (0,12) (1,4) (1,9) (4,2) (4,11) (5,1) (5,12) (7,0) (8,1)
(8,12) (10,6) (10,7) (11,2) (11,11) (12,5) (12,8)

$E(F_{13})$ memiliki 18 *point*, termasuk point at infinity φ.

2.6.2. Operasi-operasi pada kurva eliptik dalam Fp

- Operasi Penambahan (*Addition*)

  Berikut ini adalah sifat-sifat operasi penambahan pada kurva eliptik :

a. Jika φ adalah *point at infinity*, maka penjumlahan dua buah φ akan menghasilkan φ pula. Secara matematis dapat dituliskan sebagai berikut : φ + φ = φ.

b. Jika $P = (x_1,y_1) \in E(F_p)$, maka :

$$P + \varphi = \varphi + P = P$$

[16] Robert Hofer, "Systems based on ECDLP", TU-Graz,1999. (http://www.sbox.tu-graz.ac.at/home/j/jonny/project/crypto/math/ecdlp.htm)

c. Jika $P = (x_1,y_1) \in E(F_p)$, maka negasi P,–P = $(x_1,-y_1)$ juga merupakan *point* pada kurva. Penambahan dua buah *point* ini memberikan :

$$P + (-P) = P - P = \varphi$$

d. Jika $P = (x_1,y_1) \in E(F_p)$ dan $Q = (x_2,y_2) \in E(F_p)$ dengan $x_1 \neq x_2$, dan $Q \neq -P$, maka jumlah mereka :

$$P + Q = R = (x_3,y_3)$$

didefinisikan sebagai :

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1-x_3) - y_1$$

dengan

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

Sifat penting lain dari aturan penambahan ini adalah bahwa urutan penambahan tidak penting,

$$P + Q = (x_1,y_1) + (x_2,y_2) = (x_3,y_3) = (x_2,y_2) + (x_1,y_1) = Q + P$$

- Penggandaan (*Doubling*)

Jika $P = (x_1,y_1) \in E(F_p)$ adalah suatu *point* dengan $y_1 \neq 0$ dan P bukan φ, maka 2P = (x_3,y_3), dengan :

$$\lambda = \frac{(3x_1^{\,2} + a)}{2y_1}$$

$x_3 = \lambda^2 - 2x_1$, dan

$y_3 = -y_1 + \lambda\,(x_1-x_3)$

Jika P=φ atau y1=0, maka 2P=φ.

- Perkalian Skalar (*Scalar Multiplication*)

Point kurva eliptik tidak dapat dikalikan, namun dapat dilakukan *scalar multiplication*, yaitu penambahan berulang untuk *point* yang sama. Jika n adalah suatu integer positif dan P suatu point pada kurva eliptik, perkalian skalar nP adalah hasil penambahan P sejumlah n kali. Sehingga, 5P = P+P+P+P+P.

Perkalian skalar ini dapat diperluas untuk integer nol dan negatif yaitu :

$$0P = \varphi, \quad (-n)\ P = n\ (-P)$$

2.6.3 Contoh-contoh operasi pada kurva eliptik :

Berdasarkan contoh kurva eliptik F_{13} sebelumnya :

- Penambahan dua buah *point* yang berbeda

Jika P = (1,4) dan Q = (5,12), maka P+Q dapat dihitung dengan cara sebagai berikut :

$$\lambda = \frac{12-4}{5-1} = \frac{8}{4} = 2 \in F_{13}$$

$x_3 = 2^2 - 1 - 5 = 4\text{-}1\text{-}5 = -2 = 11 \pmod{13}$ dan

$y_3 = 2(1-(-2)) - 4 = 2(3) - 4 = 2 \pmod{13}$

Maka P+Q = (11,2)

- Penggandaan *Point*

Jika P = (1,4), maka 2*P dapat dihitung sebagai berikut :

$$\lambda = \frac{(3.1^2+1)}{2.4} = \frac{4}{8} = \frac{1}{2} = 7 \in F_{13}$$

$x_3 = 7^2 - 2.1 = 49 - 2 = 47 = 8 \pmod{13}$, dan

$y_3 = -4 + 6\,(1-7) = 9 + 6(-6) = 9\text{-}36 = -27 = 12 \pmod{13}$

- Perkalian Skalar

Jika k = 3 dan P=(1,4). Maka 3P dapat dihitung sebagai berikut :

$$3P = P+P+P = (2P)+P.$$

Dengan 2P = P+P = (8,12) (dari contoh penggandaan point) dan P = (1,4)

3P = (8,12) + (1,4) dapat dihitung sebagai berikut :

$$\lambda = \frac{12-4}{8-1} = \frac{8}{7} = 3 \in F_{13}$$

$x_3 = 3^2 - 1 - 8 = 0 \pmod{13}$ dan

$y_3 = 3(1-0) - 4 = 3 - 4 = -1 = 12 \pmod{13}$

Jadi 3P = (0,12).

2.6.4. Keamanan[17]

Saat ini terdapat tiga buah *Public Key Cryptosystem* yang dianggap aman dan efisien. Sistem-sistem tersebut bila diklasifikasikan menurut permasalahan matematika yang mendasarkannya adalah :

- *Integer Factorization Problem* (IFP), contohnya RSA.
- *Discrete Logarithm Problem* (DLP), contohnya ElGamal, DSA.
- *Elliptic Curve Discrete Logarithm Problem* (ECDLP), contohnya ECPKC. ECDLP ini akan dibahas lebih lanjut.

ECDLP secara sederhana dapat dijelaskan sebagai berikut :

Diketahui :

- Q sebuah point pada kurva eliptik E yang merupakan hasil perkalian antara suatu integer k modulo r [*] dengan point P atau secara matematis Q = k*P.
- Point P pada kurva eliptik E
- Kurva eliptik E.

Q dan P merupakan suatu kunci publik, sedangkan k merupakan kunci pribadi. ECDLP adalah menentukan k bila diberikan Q dan P.

Keamanan ECPKC terletak pada kesulitan *elliptic curve discrete logarithm problem*. Serupa dengan IFP dan DLP modulo *p*, tidak ada algoritma yang efisien untuk menyelesaikan *elliptic curve discrete logarithm problem*. Pada kenyataannya ECDLP dipercaya lebih sukar daripada IFP dan DLP modulo *p*. Kesulitan tambahan ini membuat ECPKC sebagai *public-key cryptographic system* terkuat yang dikenal saat ini. Keamanan moderat dapat diperoleh dari ECPKC dengan menggunakan kurva eliptik yang didefinisikan untuk suatu bilangan prima modulo p yang lebih pendek daripada 150 digit desimal.

2.7. Pemanfaatan ECC Untuk Program Enkripsi/Dekripsi[18]

Elliptic Curve Cryptosystem yang dibuat ini memiliki rutin untuk enkripsi dan dekripsi. Dalam proses enkripsi, pertama-tama dilakukan pembacaan suatu

[17] Certicom, Current Public-Key Cryptographic Systems, 1997. http://www.certicom.com

[*] Orde suatu point P pada kurva eliptik adalah bilangan integer positif terkecil sedemikian hingga r.P = φ (*point at infinity*).

[18] Volker Müller dan Sachar Paulus, Elliptische Kurven und Public Key Kryptographie, 1998.

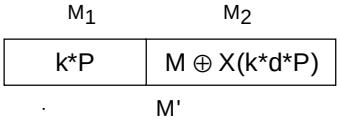
berkas kunci publik yang berisi kurva eliptik E, suatu *point* P yang berada pada E, suatu bilangan prima $p \in F_p$, dan kunci publik pemakai lain Q =d*P.

Kemudian dipilih suatu bilangan random $k \in \{2,\ldots,p-1\}$ yang berubah untuk setiap blok data, dan dihitung k*Q dan k*P, selanjutnya berkas data dibaca secara per blok (M) dan dienkripsi dengan cara :

$$M' = [k*P,\ M \oplus X(k*Q)]$$

Keterangan :
M = data yang akan dienkripsi (*plaintext*).
M' = blok data yang telah dienkripsi (*ciphertext*).
k = suatu bilangan random yang akan digunakan sebagai *session key* dengan $k \in \{2,\ldots,p-1\}$
Q = d*P
P = suatu *point* pada kurva $E(F_p)$
X(k*Q) = koordinat X untuk *point* yang dihasilkan dari perkalian k*Q.

M di-xor-kan dengan absis *point* yaitu k*Q, hasilnya berupa string yang lalu ditulis ke berkas dengan k*P ditambahkan sebelumnya. Hasil akhirnya secara sederhana dapat digambarkan sebagai berikut :



Gambar.2.4. Blok data yang telah dienkripsi

Keterangan :
M_1 : *session key*.
M_2 : Data terenkripsi.
M : Data yang belum terenkripsi.
M' : Blok data yang telah dienkripsi (*ciphertext*).
Proses ini akan terus dilakukan selama data yang dibaca masih ada.

Dalam proses dekripsi, pertama-tama dilakukan pembacaan suatu berkas kunci publik yang berisi kurva eliptik E, suatu *point* P yang berada pada E, dan suatu *field* bilangan prima p. Kemudian dibaca *ciphertext* seperti pada Gambar. 2.4. Lalu dihitung d*(k*P), dengan d adalah kunci privat yang dimasukkan oleh pemakai dalam bentuk *passphrase*, dan k*P berasal dari ciphertext. Satu buah blok data lalu dibaca (M'). Setelah itu dilakukan proses dekripsi untuk memperoleh M , dengan

$$M = [M_2 \oplus X(d*(k*P))]$$

M_2 di-xor-kan dengan absis *point* yaitu d*(k*P) sehingga diperoleh suatu string. Hasilnya (M) lalu ditulis ke berkas. Hasil akhirnya secara sederhana dapat digambarkan sebagai berikut :

$$\boxed{M_2 \oplus X(d*(k*P))}$$

M

Gambar. 2.5. Blok data yang telah didekripsi

Keterangan :
M_2 : Data terenkripsi.
M : Blok data yang telah didekripsi (plaintext).
Proses ini akan terus dilakukan selama data terenkripsi yang dibaca masih ada.

Contoh :

Dimisalkan Alice ingin mengirim suatu pesan terenkripsi dengan menggunakan *elliptic curve cryptosystems* kepada Bob.

Inisialisasi :

Kurva eliptik E : $y^2 = x^3 + x + 1$
Suatu *point* P=(1,4) pada E
Field yang digunakan (p) adalah 13
Kunci rahasia Bob adalah (d) = 4.

Enkripsi oleh Alice :

- Mula-mula Alice mengambil kunci publik Bob (dP), di sini dimisalkan dP=(11,11).
- Pesan yang akan dikirimkan (M) adalah 2 = $(0010)_2$.
- Kemudian program mengambil suatu bilangan random (k $\in$ {2,…,q-1}), misalkan k = 5.
- Dihitung k.(dP) = 5.(11,11) = (8,12). X(k.(dP)) = 8 = $(1000)_2$
- Dihitung k.P = 5.(1,4) = (5,1), ini adalah M_1.
- M $\oplus$ X(k.(dP)) = (0010 $\oplus$ 1000) = (1010), ini adalah M_2
- Pesan terenkripsinya (M') adalah ((5,1),1010).

Dekripsi oleh Bob :

Diambil kP=(5,1) dari pesan terenkripsi, dan dihitung d.(kP) = 4.(5,1) = (8,12). X(d.(kP)) adalah 8 = $(1000)_2$. Kemudian hasil ini di-*xor*-kan dengan M_2 :

$$1000 \oplus 1010 = (0010)_2$$

Terlihat bahwa hasil setelah dekripsi sama dengan pesan awal yang akan dikirim.

BAB 3
DESAIN DAN ALGORITMA

Di dalam bab ini akan dibahas struktur *class* utama yang digunakan serta beberapa algoritma utama yang digunakan dalam program yang dibuat. Selain itu akan dibahas pula mengenai desain tampilan program.

3.1. Struktur *Class Point*

Program yang dibuat memiliki sebuah *class* yang berfungsi untuk menangani operasi-operasi *point elliptic curve* yang akan digunakan dalam proses enkripsi dan dekripsi nantinya.

```
class point {
private:
        static bigmod *a4,*a6;
        bigmod x;
        bigmod y;
        bool is_zero;

public:
        static void init_curve (const bigmod& A4, const bigmod& A6);

        point() { is_zero=true; }

        point(const bigmod& Xp, const bigmod& Yp)
        { x = Xp; y = Yp; }

        void assign_zero(void)
        { is_zero = true; }

        friend void assign(point& H, const point & P)
        {
           H.x = P.x; H.y = P.y;
           H.is_zero = P.is_zero;
        }

        bool on_curve()
        {
          bigmod h;
          h = (y*y)-(x*x+ *a4)*x-*a6;
          return (h==0)? 1:0;
        }

        friend void get_x(bigmod& X, const point& P)
        {   X = P.x ;  }

        friend void get_x(bigint& X, const point& P)
        {   X = mantissa(P.x); }

        friend void get_y(bigmod& Y, const point & P)
        {   Y = P.y;   }

        friend void get_y(bigint& Y, const point& P)
        {   Y = mantissa(P.y); }

        void set_point(const bigmod & X, const bigmod & Y)
```

```
        {       is_zero = false;
                x = X; y = Y;
        }

        // --- overloaded operators ---
        friend int operator== (const point& P1, const point& P2);
        friend ostream& operator<< (ostream& c, const point& P);

        //--- operator functions ---
        friend void neg_point (point& H, const point& P);
        friend void add_point (point& H, const point& P1, const point& P2);
        friend void sub_point (point& H, const point& P1, const point& P2);
        friend void mul_point (point& H, const bigint& k, const point& P);
        friend void double_point (point& H, const point& P);
};
```

Data member `bigmod *a4` dan `*a6` digunakan untuk menyimpan koefisien a4 dan a6 persamaan kurva eliptik. Keduanya bersifat statik karena nilainya tidak akan diubah untuk satu kali proses program.

Variabel `bigmod x` dan `bigmod y` digunakan untuk menyimpan koordinat suatu *point* yang terdapat pada kurva eliptik.

Variabel `bool is_zero` digunakan untuk menyatakan apakah suatu *point* adalah φ atau bukan, bila is_zero bernilai *true* maka point tersebut adalah φ.

Fungsi `static init_curve (const bigmod& A4, const bigmod& A6)` digunakan untuk menetapkan nilai a4 dan a6 yang digunakan untuk menyatakan persamaan kurva eliptik.

`point()` dan `point(const bigmod& Xp, const bigmod& Yp)` adalah konstruktor untuk kelas point yang digunakan.

Fungsi `void assign(point& H, const point & P)` digunakan untuk menyalinkan suatu *point* ke *point* lain.

Fungsi `bool on_curve()` digunakan untuk memeriksa apakah *point* yang dimasukkan ada pada *elliptic curve* atau tidak. Bila *point* terletak pada kurva maka fungsi ini akan bernilai benar sebaliknya akan bernilai salah.

Fungsi `void get_x(bigmod& X, const point& P)` dan `void get_y(bigmod& Y, const point & P)` digunakan untuk mengambil nilai X dan nilai Y suatu *point* kemudian memasukkannya ke variabel bigmod X dan bigmod Y. Sementara fungsi `void get_x(bigint& X, const point& P)` dan `void get_y(bigint& Y, const point & P)` digunakan untuk mengambil nilai X dan nilai Y suatu *point* kemudian memasukkannya ke variabel bigint X dan bigint Y.

Fungsi `void set_point(const bigmod & X, const bigmod & Y)` digunakan untuk menetapkan koordinat suatu *point* .

Fungsi `int operator== (const point& P1, const point& P2)` adalah suatu operator overloading yang digunakan untuk menguji apakah suatu *point* pada argumen 1 sama dengan *point* lain pada argumen 2. Jika *point*-1 sama dengan *point*-2 maka nilai kembalian fungsi ini adalah 1 sebaliknya adalah 0.

Fungsi `ostream& operator<< (ostream& c, const point& P)` adalah operator overloading operator<< yang digunakan untuk menampilkan koordinat *point*.

Fungsi `void neg_point (point& H, const point& P), void add_point (point& H, const point& P1, const point& P2), void sub_point (point& H, const point& P1, const point& P2), void mul_point (point& H, const bigint& k, const point& P), void double_point (point& H, const point& P),` masing-masing digunakan untuk menegasi point, menambah dua buah point, mengurangkan point, mengalikan suatu point dengan suatu bilangan k, dan menggandakan suatu point.

3.2. Algoritma

Berikut ini adalah beberapa buah algoritma aritmatika *elliptic curve* yang menjadi dasar bagi proses enkripsi dan dekripsi program nantinya. Algoritma-algoritma tersebut antara lain adalah algoritma untuk mencari negasi sebuah *point* yang ada di *elliptic curve*, algoritma untuk penjumlahan dua buah *point*, algoritma pengurangan *point*, algoritma untuk perkalian sebuah *point* dengan sebuah integer yang sangat besar, algoritma MD5 yang digunakan untuk menangani *passphrase* yang dimasukkan oleh pemakai, algoritma untuk enkripsi berkas, algoritma dekripsi berkas, serta algoritma untuk menghapus berkas secara aman (*wipe file*).

3.2.1. Algoritma Negasi Titik (*Point Negation*)[1]

Input : sebuah titik P (x_0,y_0) pada kurva eliptik E.

Output : titik -P=(x_1,y_1)

[1] IEEE, Standard Specifications for Public Key Cryptography (draft version 12), Oktober 1999.

Algoritma :

1. if P = point at infinity then

   output=point at infinity

2. $x_1 \leftarrow x_0$.

   $y_1 \leftarrow -(y_0)$.

3. Output (x1,y1).

3.2.2. Algoritma Penambahan Point (*Point Addition*)[2]

Input : suatu bilangan prima $p > 3$

koefisien a, b untuk sebuah kurva eliptik E: $y^2 = x^3 + ax + b$ modulo p

titik $P_0 = (x_0,y_0)$ dan $P_1=(x_1,y_1)$ pada E

Output : titik $P_2 = P_0 + P_1$

Algoritma :

1. if $P_0 = \varphi$ then $P_2 \leftarrow P_1$ dan berhenti.
2. if $P_1 = \varphi$ then $P_2 \leftarrow P_0$ dan berhenti.
3. if $x_0 \neq x_1$ then
   3.1. set $\lambda \leftarrow (y_0-y_1)/(x_0-x_1) \bmod p$
   3.2. go to step 7.
4. if $y_0 \neq y_1$ or $y_1 = 0$ then $P_2 \leftarrow \varphi$ dan berhenti.
5. set $\lambda \leftarrow (3x_1^2+a)/(2y_1) \bmod p$
6. set $x_2 \leftarrow \lambda^2 - x_0 - x_1 \bmod p$
7. set $y_2 \leftarrow \lambda (x_1-x_2) - y_1 \bmod p$
8. output $P_2 \leftarrow (x_2,y_2)$

3.2.3. Algoritma Pengurangan Point (*Point Subtraction*)[3]

Input : suatu bilangan prima $p > 3$

koefisien a, b untuk sebuah kurva eliptik E: $y^2 = x^3 + ax + b$ modulo p

titik $P_0 = (x_0,y_0)$ dan $P_1=(x_1,y_1)$ pada E

Output : titik $P_2 = P_0 - P_1$

Algoritma :

1. if $P_0 = \varphi$ then $P_2 \leftarrow P_1$ dan berhenti.
2. if $P_1 = \varphi$ then $P_2 \leftarrow P_0$ dan berhenti.
3. if $x_0 \neq x_1$ then
   3.1. set $\lambda \leftarrow (y_0-y_1)/(x_0-x_1) \bmod p$
   3.2. go to step 7.

[2] Ibid.
[3] Ibid.

4. if $y_0 \neq y_1$ or $y_1 = 0$ then $P_2 \leftarrow \varphi$ dan berhenti.
5. set $\lambda \leftarrow (3x_1^2+a)/(2y_1) \bmod p$
6. set $x_2 \leftarrow \lambda^2 - x_0 - x_1 \bmod p$
7. set $y_2 \leftarrow \lambda (x_1-x_2) - y_1 \bmod p$
8. output $P_2 \leftarrow (x_2,y_2)$

3.2.4. Perkalian skalar kurva eliptik (*elliptic scalar multiplication*)[4]

Input : suatu integer n dan sebuah titik kurva eliptik P

Output : *point* nP

Algoritma :

1. If n = 0 then output φ dan berhenti.
2. If n < 0 then set Q ← (-P) dan k ← (-n), else set Q ← P dan k ← n.
3. Misalkan $h_L h_{L-1} \ldots h_1 h_0$ adalah representasi biner 3k, dengan MSB h_L adalah 1.
4. Misalkan $k_L k_{L-1} \ldots k_1 k_0$ merupakan representasi biner k.
5. Set S ← Q.
6. For i from L-1 downto 1 do
   Set S ← 2S.
   if h_i=1 and k_i=0 then S ← S + Q (algoritma 3.2.2).
   if h_i=0 and k_i=1 then S ← S - Q (algoritma 3.2.3).
7. Output S.

3.2.5. Algoritma MD5[5]

Dimisalkan terdapat input pesan berukuran b-bit, dengan b sembarang bilangan integer positif, b dapat bernilai 0 dan tidak perlu merupakan kelipatan delapan, dan dapat berukuran besar sekali. Dimisalkan pula bit-bit tersebut ditulis sebagai berikut :

$$m_0 \quad m_1 \quad \ldots \quad m_{b-1}$$

Kelima langkah berikut ini dilakukan untuk menghitung *message digest* pesan:

- Langkah 1 : Menambah *Padding Bits*

  Pesan di-*padded* (diperpanjang) hingga ukurannya (dalam bit) kongruen dengan 448, modulo 512.

[4] Ibid.
[5] Ronald L. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, MIT dan RSA Data Security, Inc., April 1992. (ftp://ftp.isi.edu/in-notes/rfc1421.txt)

Padding dilakukan dengan cara sebagai berikut : suatu bit tunggal "1" ditambahkan ke pesan, dan kemudian bit-bit "0" ditambahkan sehingga panjang *padded message* dalam bit menjadi kongruen dengan 448, modulo 512. Secara umum, paling sedikit ditambahkan satu bit dan paling banyak 512 bit.

- Langkah 2 : Menambah Panjang

  Suatu representasi 64-bit b (panjang pesan sebelum ditambah bit padding) ditambahkan ke hasil langkah sebelumnya. Jika b lebih dari 2^{64}, maka hanya 64 bit orde terendah b yang digunakan. Bit-bit ini ditambahkan sebagai dua buah word 32-bit dan word orde terendah ditambahkan terlebih dulu.

  Pada langkah ini pesan yang dihasilkan (setelah ditambah dengan bit-bit dan b) memiliki panjang yang merupakan kelipatan 512 bit. Secara ekivalen, pesan ini memiliki panjang yang merupakan kelipatan 16 (32-bit) word. Misalkan M[0..N-1] menyatakan word pesan yang dihasilkan, dengan N adalah kelipatan 16.

- Langkah 3 : Inisialisasi Buffer MD

  Sebuah *buffer* empat-word (A, B, C, D) digunakan untuk menghitung message digest. Di sini A, B, C, dan D merupakan register 32-bit. Register-register ini diinisialisasi ke nilai-nilai heksadesimal berikut dengan byte orde rendah terlebih dulu :

```
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10
```

- Langkah 4 : Proses Pesan dalam blok 16-word

  Pertama-tama didefinisikan empat buah fungsi tambahan yang menerima tiga word 32-bit sebagai input dan menghasilkan satu word 32-bit sebagai output.

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))
```

  Dalam setiap posisi bit F bertindak sebagai kondisional: if X then Y else Z. Fungsi F dapat didefinisikan menggunakan + (and) selain v (or) karena XY

and not(X)Z tidak akan pernah memiliki 1 dalam posisi bit yang sama. Perhatikan bahwa jika bit X, Y, dan Z independen dan tidak bias, setiap bit F(X,Y,Z) akan independen dan tidak bias.

Fungsi G, H, dan I serupa dengan fungsi F, yaitu mereka dapat bertindak dalam *bitwise parallel* untuk menghasilkan output dari bit-bit X, Y, dan Z, dalam suatu cara sehingga jika bit koresponden X, Y, dan Z independen dan tidak bias, maka setiap bit G(X,Y,Z), H(X,Y,Z), dan I(X,Y,Z) akan independen dan tidak bias. Perhatikan bahwa fungsi H adalah *bit-wise* xor atau fungsi *parity* untuk input-inputnya.

Langkah ini menggunakan tabel 64-elemen T[1...64] yang dihasilkan dari fungsi sin. T[i] menyatakan elemen ke-i tabel, yang setara dengan bagian integer dari operasi `4294967296` dikalikan `abs(sin(i))`, dengan i dalam radian.

Lakukan proses berikut :

```
/* Proses setiap blok 16-word. */
For i = 0 to N/16-1 do

/* Salin blok i ke X. */
For j = 0 to 15 do
   Set X[j] to M[i*16+j].
end /* loop j */

/* Simpan A sbg AA, B sbg BB, C sbg CC, dan D sbg DD. */
AA = A
BB = B
CC = C
DD = D

/* Ronde 1. */
/* Misalkan [abcd k s i] menyatakan operasi
     a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Lakukan 16 operasi berikut. */
[ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
[ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
[ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

/* Ronde 2. */
/* Misalkan [abcd k s i] menyatakan operasi
     a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* Lakukan 16 operasi berikut. */
[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
[ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
[ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
[ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]
```

```
    /* Ronde 3. */
    /* Misalkan [abcd k s t] menyatakan operasi
         a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
    /* Lakukan 16 operasi berikut. */
    [ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
    [ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
    [ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
    [ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

    /* Ronde 4. */
    /* Let [abcd k s t] denote the operation
         a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
    /* Lakukan 16 operasi berikut. */
     [ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
     [ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
     [ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
     [ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]

    /* Kemudian lakukan penambahan berikut. (Naikkan setiap register
    dengan nilai yang dimiliki sebelum blok ini dimulai) */
    A = A + AA
    B = B + BB
    C = C + CC
    D = D + DD

  end /* loop i */
```

- Langkah 5 : Hasil

Message digest yang dihasilkan sebagai output adalah A, B, C, D. Dimulai dari byte orde rendah A, dan berakhir dengan byte berorde tinggi D.

3.2.6.Algoritma Enkripsi Blok

Algoritma ini berfungsi untuk mengenkripsi suatu buah blok pesan yang besarnya sekitar 20 atau 24 karakter.

Input : pesan yang akan dienkripsi dengan ukuran 20 atau 24 karakter.
koordinat X dari perkalian k dengan kunci publik user (d.P), X(k.d.P).
pointer ke berkas hasil.
Output : pesan terenkripsi pada berkas hasil.

1. Mulai.
2. Ambil satu buah blok pesan.
3. Lakukan proses XOR blok pesan dengan X(kdP).
4. Tulis hasilnya ke berkas hasil.
5. Selesai.

3.2.7. Algoritma Dekripsi Blok

Algoritma ini berfungsi untuk mendekripsi sebuah blok pesan yang besarnya sekitar 20 atau 24 karakter.

Input : pesan yang akan didekripsi dengan ukuran 20 atau 24 karakter.
koordinat X dari perkalian d dengan k.P, X(d.k.P).
pointer ke berkas hasil.

Output : pesan terdekripsi pada berkas hasil.
1. Mulai.
2. Ambil satu buah blok pesan.
3. Lakukan proses XOR blok pesan dengan X(d.k.P).
4. Tulis hasilnya ke berkas hasil.
5. Selesai.

3.2.8. Algoritma Enkripsi Berkas

Algoritma ini berfungsi untuk mengenkripsi sebuah berkas yang dimasukkan pemakai.

Input : nama pemakai yang akan menerima berkas terenkripsi.
nama berkas yang akan dienkripsi.
Output : berkas terenkripsi dengan nama berkas asli+".tec".
1. Mulai.
2. Ambil parameter kurva dan sebuah *point* P dari berkas kunci publik.
3. Inisialisasi proses enkripsi.
4. Tulis *header* dan informasi lain ke dalam berkas hasil.
5. Lakukan proses berikut selama blok pesan berkas input masih ada :
   5.1. pilih k sembarang.
   5.2. hitung k.P
   5.3. tuliskan hasil perkalian ke berkas hasil.
   5.4. ambil sebuah blok pesan.
   5.5. enkripsi blok pesan.
   5.6. simpan hasilnya ke berkas hasil.
6. Selesai.

3.2.9. Algoritma Dekripsi Berkas

Algoritma ini berfungsi untuk mendekripsi sebuah berkas yang dimasukkan pemakai.

Input : nama pemakai yang akan menerima berkas terdekripsi.
nama berkas yang akan didekripsi.
Output : berkas terdekripsi.
1. Mulai.
2. Ambil parameter kurva dan sebuah *point* P dari berkas kunci publik.
3. Inisialisasi proses enkripsi.
4. Lakukan proses berikut selama blok pesan berkas input masih ada :
   4.1. ambil k.P dari berkas input.
   4.2. hitung d.k.P
   4.3. ambil sebuah blok pesan.
   4.4. dekripsi blok pesan.
   4.5. simpan hasilnya ke berkas hasil.
6. Selesai.

3.2.10. Algoritma Ambil Field (*get field*)

Algoritma ini berfungsi untuk mengambil sebuah field yang ada di dalam suatu berkas.

Input : berkas yang akan diambil fieldnya.
sebuah user-name sebagai kata kunci.
field_num, field yang akan diambil isinya.
Output : isi field yang diinginkan.

1. Mulai.
2. Set separator_field ← "#"
3. Buka berkas
4. Jika ada kesalahan :
   4.1. tampilkan pesan kesalahan.
   4.2. ke langkah 7
5. Jika tidak ada kesalahan :
   while (not end_of_file)
   - ambil satu baris data dari berkas.
   - set field_counter ← 1
   - pisahkan field berdasarkan separator_field
   - name ← field[1]
   - apakah name = username ?
   - ya :
     while (field_counter < field_num)
     pisahkan field berdasarkan separator_field
     result ← field[field_counter]
     field_counter ← field_counter+1
     endwhile
     keluar dari loop
   - tidak :
     ambil satu baris berikutnya
   endwhile
6. Berikan result
7. Tutup berkas
8. Selesai

3.2.11. Algoritma Hapus Berkas (w*ipe file*)

Algoritma ini berfungsi untuk menghapus berkas agar isinya tidak dapat dikembalikan lagi.

Input : sebuah berkas yang akan dihapus.

Output : berkas terhapus.

Algoritma :

1. Mulai
2. Buka berkas
3. Jika ada kesalahan, lakukan langkah berikut :
   3.1. cetak kesalahan
   3.2. ke langkah 6
4. Jika tidak ada kesalahan, lakukan langkah berikut :
   4.1. set ch ← " "
   4.2. set i ← 0
   4.3. panjang_berkas = filesize(berkas)
   4.4. while (i<panjang_berkas)
        tulis ch ke berkas
        i = i +1
      end while
5. Tutup berkas
6. Hapus berkas
7. Selesai

3.3. Desain Program

3.3.1. Perintah-perintah

Perintah-perintah program yang dibuat terdiri dari :

- Enkripsi (*encrypt*) : perintah ini digunakan bila pemakai ingin mengenkripsi sebuah berkas yang telah dipilihnya.
- Dekripsi (*decrypt*) : perintah ini digunakan bila pemakai ingin mendekripsi berkas yang telah dipilihnya.
- Buat Kunci Publik (*generate public key*) : perintah ini digunakan untuk membuat sebuah kunci publik pemakai.
- Cari *Point* (*find point*) : perintah ini digunakan untuk mencari *point* yang ada pada suatu *elliptic curve*.
- Periksa *Point* (*test point*) : perintah ini digunakan untuk menguji apakah *point* yang dimasukkan ada pada suatu *elliptic curve*.

- Hapus Berkas (*wipe file*) : perintah ini merupakan perintah yang dapat ditambahkan pada perintah enkripsi untuk menghapus berkas yang akan dienkripsi, sehingga setelah proses enkripsi selesai, berkas yang ada hanya berkas yang terenkripsi saja.

3.3.2. Enkripsi (*encrypt*)

Untuk dapat memakai perintah ini, pemakai perlu memasukkan nama berkas yang ingin dienkripsi, hasilnya akan disimpan dalam sebuah berkas dengan *extention* "tef".

3.3.3. Dekripsi (*decrypt*)

Untuk dapat mendekripsi sebuah berkas, pemakai harus memasukkan sebuah berkas yang telah terenkripsi dengan program ini, yang biasanya ditandai dengan *extention* "tef". Hasil dekripsi disimpan dengan *extension* "out". Sebagai contoh dilakukan enkripsi terhadap berkas "uwthesis01.ps", hasil enkripsinya adalah berkas"uwthesis01.ps.tef". Kemudian dilakukan dekripsi terhadap berkas ini, hasilnya tersimpan dalam berkas "uwthesis01.ps.out".

3.3.4. Buat Kunci Publik (*generate public key*)

Perintah ini digunakan untuk membuat kunci publik. Dalam *public key cryptography*, dibutuhkan dua buah kunci, yaitu kunci publik dan kunci pribadi. Dalam proses enkripsi/dekripsi, kunci publik biasanya digunakan oleh pihak lain untuk mengenkripsi data, sementara kunci pribadi biasanya digunakan untuk mendekripsi data tersebut.

Untuk dapat membuat kunci publik pemakai perlu memasukkan informasi mengenai parameter *elliptic curve*, koordinat *point* pada kurva, dan sebuah *passphrase* yang digunakan sebagai kunci pribadi. Informasi ini akan disimpan dalam berkas kunci publik.

3.3.5. Cari *Point* (*find point*) dan Periksa *Point* (*test point*)

Perintah ini digunakan untuk mencari sebuah *point* yang terletak pada *elliptic curve*. Pemakai perlu memasukkan informasi mengenai *prime field* serta

parameter kurva. Hasil akhirnya akan disimpan dalam berkas "ec.point". Informasi tentang *point* ini kemudian dapat digunakan dalam perintah "Buat Kunci Publik".Sedangkan perintah "Periksa *Point*" digunakan untuk memeriksa apakah sebuah *point* terletak pada *elliptic curve*. Pemakai perlu memasukkan informasi mengenai *prime field* serta parameter kurva.

3.3.6. Hapus File (*wipe file*)

Perintah ini ditambahkan ke dalam program dengan harapan dapat meningkatkan keamanan berkas yang akan dienkripsi maupun yang telah dienkripsi.

BAB 4
IMPLEMENTASI DAN ANALISIS

4.1. Implementasi

4.1.1. Program

Program dibuat dengan menggunakan *compiler* Microsoft Visual C++ versi 6.0. Program ini dapat langsung dijalankan pada sistem operasi Microsoft Windows 9x tanpa perlu *compiler* Microsoft Visual C++, karena telah berupa program *executable*.

Program ini menggunakan *computational number theory library* LiDIA versi 1.3.3. LiDIA ini terutama digunakan pada operasi-operasi matematika *elliptic curve*. Selain itu, program ini juga memanfaatkan rutin MD5 yang berasal dari RSA untuk menangani *passphrase* yang dimasukkan pemakai.

Program ini dibuat dengan tujuan utama sebagai sarana penelitian terhadap *elliptic curve cryptography*, sehingga program tidak dioptimisasi baik untuk kecepatan maupun ukuran.

4.1.2. Komponen-komponen Program

Program ini terdiri dari lima buah berkas kode sumber C++ dan empat buah berkas *header*, yaitu :

a. Berkas kode sumber untuk operasi-operasi *elliptic curve*, kode sumber ini berfungsi untuk menangani operasi-operasi pada *elliptic curve*, yaitu operasi penambahan, pengurangan, penggandaan, serta perkalian. Kode sumber ini banyak memanfaatkan rutin-rutin yang ada pada *library* LiDIA. Kode sumber ini dapat ditemukan pada berkas "ecc.cpp". *Header* dapat dijumpai pada berkas "ecc.h".
b. Berkas kode sumber untuk penanganan *passphrase* yang dimasukkan oleh pemakai. Kode sumber ini merupakan modifikasi kode sumber MD5 yang berasal

dari RSA. Kode sumber ini dapat ditemukan pada berkas "md5.cpp". *Header* dapat dijumpai pada berkas "md5.h".

c. Berkas kode sumber untuk operasi enkripsi dan dekripsi. Kode sumber ini berguna untuk melakukan operasi enkripsi dan dekripsi terhadap suatu berkas yang dimasukkan oleh pemakai. Kode sumber ini dapat dijumpai pada berkas "crypt.cpp". *Header* dapat dijumpai pada berkas "crypt.h".
d. Berkas kode sumber utilitas. Kode sumber ini berisi rutin-rutin untuk menangani operasi-operasi yang tidak tercakup dalam berkas-berkas yang telah dijelaskan di atas, misalnya rutin untuk membuat kunci publik. Kode sumber ini dapat dijumpai pada berkas "utils.cpp". *Header* dapat dijumpai pada berkas "utils.h".
e. Berkas kode sumber utama. Kode sumber ini merupakan bagian utama program, yang berfungsi menyatukan seluruh kode sumber-kode sumber lainnya. Kode sumber ini dapat dijumpai pada berkas "aztecc.cpp". *Header* dapat dijumpai pada berkas "aztecc.h".

4.1.3. Menu Program

Program ini terdiri dari empat buah menu utama, yaitu :

- Menu Enkripsi. Menu ini digunakan untuk melakukan proses enkripsi file.
- Menu Dekripsi. Menu ini digunakan untuk melakukan proses dekripsi file.
- Menu Membuat Kunci Publik. Menu ini digunakan bila pemakai ingin membuat kunci publik, yang kemudian akan digunakan dalam proses enkripsi dan dekripsi.
- Menu Mencari *Point*. Menu ini digunakan untuk mencari sebuah *point* berdasarkan informasi yang telah dimasukkan oleh pemakai.
- Menu Menguji *Point*. Menu ini digunakan untuk menguji apakah sebuah *point* yang dimasukkan oleh pemakai berada pada kurva eliptik yang ditentukan.

4.1.4. Modifikasi Program

Untuk memodifikasi program ini dalam Microsoft Visual C++ dapat dilakukan langkah-langkah sebagai berikut :

- Membuat sebuah direktori untuk menampung *source code* program. Pada sistem operasi Windows, masuklah ke MS-DOS Prompt, lalu berikan perintah berikut :

  ```
  C:\WINDOWS>cd \
  ```

- Menyalinkan berkas terkompresi, "aztecc100s.exe" yang berada di *floppy disk* ke direktori teratas tersebut dengan cara :

  ```
  C:\> copy a:aztecc100s.exe .
  ```

- Membuka berkas terkompresi yang telah disalinkan tadi dengan cara :

  ```
  C:\> aztecc100s
  ```

  Berkas-berkas yang diperlukan telah dibuka dan berada dalam direktori "aztecc\src".

- Masuk ke dalam Microsoft Visual C++. Buka berkas "aztecc.dsw" yang ada di direktori C:\aztecc\src.
- Program siap dimodifikasi.

4.1.5. Instalasi Program

Instalasi program dapat dilakukan dengan cara sebagai berikut :

- Membuat sebuah direktori untuk menampung program beserta berkas kunci publik. Pada sistem operasi Windows, masuklah ke MS-DOS Prompt, lalu berikan perintah berikut :

  ```
  C:\WINDOWS>cd \
  ```

- Menyalinkan berkas terkompresi, "aztecc100b.exe" yang berada di *floppy disk* ke direktori teratas tersebut dengan cara :

  ```
  C:\> copy a:aztecc100b.exe .
  ```

- Membuka berkas terkompresi yang berisi program dengan cara :

  ```
  C:\> aztecc100b
  ```

  Berkas-berkas yang diperlukan telah dibuka dan berada di direktori "C:\aztecc\bin".

- Program telah siap digunakan.

4.1.6. Menghapus Program

Menghapus program dapat dilakukan dengan cara sebagai berikut :

- Pastikan tidak ada berkas-berkas yang masih terenkripsi dengan program ini, karena berkas-berkas tersebut tidak dapat didekripsi bila program ini sudah terhapus.
- Masuklah ke MS-DOS Prompt. Lalu pindah ke direktori utama tempat program tersimpan. Dalam contoh di atas, program tersimpan dalam direktori C:\aztecc sehingga direktori teratas adalah C:\.

```
C:\WINDOWS>cd \
```

- Hapuslah direktori tersebut dengan cara :

```
C:\>deltree aztecc
```

Lalu akan dimunculkan pertanyaan untuk konfirmasi :

```
Delete directory "aztecc" and all its subdirectories? [yn] y
Deleting aztecc...
C:\>
```

- Program telah berhasil dihapus.

4.1.7. Pemakaian Program

- Membuat Kunci Publik

Sebelum dapat melakukan operasi enkripsi/dekripsi, pemakai perlu terlebih dahulu membuat sebuah kunci publik, yang kemudian dimasukkan ke dalam berkas "pubkey.pk", yang merupakan berkas berisikan kunci publik.

Untuk membuat kunci publik, pemakai memilih menu -g yang diberikan pada baris perintah program, kemudian memberikan opsi ukuran kunci dan nama berkas kunci publik. Bila kedua opsi dikosongkan, maka secara *default* ukuran kunci adalah 160-bit dan nama berkasnya adalah "pubkey.pk". Pemakai harus memastikan bahwa kunci publik yang dimasukkannya belum ada dalam kunci publik, karena program tidak memiliki fasilitas penanganan hal ini. Selain itu, pemakai hanya dapat membuat satu buah kunci publik saja untuk satu buah *username*.

```
E:\test>aztecc -g
```

```
============================================================
AZTECC - an implementation of elliptic curve cryptosystem
Version 1.0 - Copyright (c) 1999-2000 by Tedi Heriyanto
This is a freeware edition. Use it with your own risk.
============================================================

Before you can use encryption/decryption you must generate your public key.
The following process will guide you through generating your public key.

To achieve a cryptographically secure elliptic curve cryptosystem, you must
supply a relatively big prime finite field (q), a good a4, a6
and a point that lies on the curve.

GENERATE PUBLIC KEY
-------------------
Please enter your user name : tdh
Please enter prime field (q):
                          1461501637330902918203684832716283019655932542133

Please enter numbers to the following :
Enter a4 : 1425585077258738319373335796649123776680168404251
Enter a6 : 1157518572960462695019885206412388528052434336480
Enter xP : 552309223205390039036191152356468692153096936343
Enter yP : 157570912242768897257744956754779686263802042201

INPUT PASSPHRASE :
------------------
Hallo =>tdh<= please enter your passphrase (up to 2047 chars)
This is your PRIVATE KEY. Do not forget it.
If you forget it you'll never be able to decrypt the encrypted files.

When you're done, press <ENTER>.
Begin entering your passphrase ==>

Done computing your private key.
Done saving your public key to pubkey.pk
```

Gambar 4-1 Pemakai memilih perintah g

Keterangan :
Yang bergaris bawah adalah input yang dimasukkan oleh pemakai

Kemudian pemakai perlu mengisikan beberapa informasi mengenai nama pemakai, *elliptic curve* dan parameternya, yaitu *prime finite field* yang akan digunakan, nilai a4 dan a6 yang akan membentuk persamaan *elliptic curve*, serta koordinat X dan Y sebuah *point* pada *elliptic curve*. Di akhir pengisian, pemakai diminta untuk memasukkan sebuah *passphrase* yang akan menjadi kunci pribadinya. Passphrase yang dimasukkan ini kemudian akan di-*hash* dan nilainya digunakan dalam mencari sebuah *point* lain pada *elliptic curve*.

Isi berkas yang dihasilkan adalah sebagai berikut :

```
tdh#93432f0ad67b79c0282ccd083fb808e5#14255850772587383193733357966491237766801 68
404251#11575185729604626950198852064123885280524343364 80#5523092232053900390361 9
11523564686921530969363 43#15757091224276889725774495675477968626380204220 1#85110
70142510553203499376696108205997558758317 06#11651420186353936426722560130131270 7
8876784376082#14615016373309029182036848327162830196559325421 33
```

Gambar 4-2 Isi berkas kunci publik

- Enkripsi

Untuk dapat melakukan operasi enkripsi, pemakai perlu memberikan informasi mengenai ukuran kunci yang diinginkan, 160-bit atau 192-bit, serta berkas yang akan dienkripsi. Selain itu berkas kunci publik, pubkey.pk, harus tersedia. Berkas ini dapat diperoleh dengan cara menambahkan berkas kunci publik yang telah dibuat ke dalam berkas pubkey.pk. Bila operasi enkripsi berhasil, maka akan tercipta sebuah berkas baru dengan ekstensi ".tef".

Dalam melakukan operasi enkripsi ini, berkas asli tidak dihapus, bila pemakai ingin meningkatkan keamanan berkas terenkripsi, maka terdapat pilihan untuk menghapus berkas asli, yaitu -w.

Sebelum Enkripsi

```
AZTECC   EXE       135,168  01-01-00  1:00a aztecc.exe
GENFILE  EXE        40,960  01-01-00  1:00a genfile.exe
PUBKEY   PK            435  01-01-00  1:00a pubkey.pk
T100     TXT           100  01-01-00  1:00a t100.txt
         4 file(s)        176,663 bytes
         2 dir(s)   1,600,532,480 bytes free
```

Proses Enkripsi

```
E:\test>aztecc -e t100.txt -u tdh
=======================================================
AZTECC - an implementation of elliptic curve cryptosystem
Version 1.0 - Copyright (c) 1999-2000 by Tedi Heriyanto
This is a freeware edition. Use it with your own risk.
=======================================================
Encrypting file : t100.txt
The result will be stored in : t100.txt.tef
Processing : .
Done encrypting =>t100.txt<=
Elapsed time : 5 sec 49 hsec
```

Setelah Enkripsi

```
AZTECC   EXE       135,168  01-01-00  1:00a aztecc.exe
GENFILE  EXE        40,960  01-01-00  1:00a genfile.exe
T100TX~1 TEF           377  01-01-00  1:00a t100.txt.tef
PUBKEY   PK            435  01-01-00  1:00a pubkey.pk
T100     TXT           100  01-01-00  1:00a t100.txt
         5 file(s)        177,040 bytes
         2 dir(s)   1,600,532,480 bytes free
```

Gambar 4-3 Operasi Enkripsi

Isi berkas sebelum dan sesudah proses enkripsi adalah sebagai berikut :

t100.txt

```
00000  61 62 63 64 ¦ 65 66 67 68 ¦ 69 6A 6B 6C ¦ 6D 6E 6F 70   abcdefghijklmnop
00010  71 72 73 74 ¦ 75 76 77 78 ¦ 79 7A 61 62 ¦ 63 64 65 66   qrstuvwxyzabcdef
00020  67 68 69 6A ¦ 6B 6C 6D 6E ¦ 6F 70 71 72 ¦ 73 74 75 76   ghijklmnopqrstuv
00030  77 78 79 7A ¦ 61 62 63 64 ¦ 65 66 67 68 ¦ 69 6A 6B 6C   wxyzabcdefghijkl
00040  6D 6E 6F 70 ¦ 71 72 73 74 ¦ 75 76 77 78 ¦ 79 7A 61 62   mnopqrstuvwxyzab
00050  63 64 65 66 ¦ 67 68 69 6A ¦ 6B 6C 6D 6E ¦ 6F 70 71 72   cdefghijklmnopqr
00060  73 74 0D 0A ¦                                           st..
```

t100.txt.tef

```
00000  74 64 68 20 ¦ 74 31 30 30 ¦ 2E 74 78 74 ¦ 20 31 30 30   tdh t100.txt 100
00010  0A 4B B2 D1 ¦ 3C 6A CD 48 ¦ FB 37 0D C2 ¦ 4C 1C D9 6B   .K..<j.H.7..L..k
00020  5A 9E E5 79 ¦ AD 9B CF 51 ¦ 63 54 A1 60 ¦ 7B A1 14 CA   Z..y...QcT.`{...
00030  F0 C1 AB 44 ¦ DC 1E 83 CC ¦ 12 DE A8 7C ¦ 2A 29 47 6D   ...D.......|*)Gm
00040  D9 E7 20 56 ¦ 93 17 D6 3E ¦ 9A 7E 89 59 ¦ 5B CD 3D F8   .. V...>.~.Y[.=.
00050  1C 4E 6F D7 ¦ 2E 3C C5 47 ¦ B4 72 CE 47 ¦ F4 75 46 6C   .No..<.G.r.G.uFl
00060  82 01 57 1E ¦ 76 B6 42 C4 ¦ E8 A3 0E 9E ¦ DB 33 61 5D   ..W.v.B......3a]
00070  D3 76 3C 15 ¦ F3 DB 96 FF ¦ 19 7D B6 56 ¦ 17 EE 2D BB   .v<......}.V..-.
00080  01 9B C4 6D ¦ 63 FC 63 68 ¦ 84 3B 47 49 ¦ 9D B2 DE B8   ...mc.ch.;GI....
00090  83 D8 64 B7 ¦ 1D F6 71 39 ¦ 81 DE 3E EF ¦ C3 41 C7 17   ..d...q9..>..A..
000A0  4B 03 DA 48 ¦ D6 C8 05 6A ¦ C4 A8 0A E0 ¦ 58 EB E0 42   K..H...j....X..B
000B0  70 61 EE F5 ¦ C3 49 DD F6 ¦ 38 E3 AA 1B ¦ 03 0D A8 C7   pa...I..8.......
000C0  59 7E FA 8B ¦ D4 D0 9E 6D ¦ 8E C5 FD CB ¦ B0 1C D0 B6   Y~.....m........
000D0  09 21 08 8B ¦ DE 5F 97 DD ¦ 61 CF A9 B7 ¦ 76 67 E1 4F   .!...._..a...vg.O
000E0  F9 33 5E A3 ¦ 8B 85 69 C4 ¦ AC 7E 1E 3F ¦ 00 83 78 99   .3^...i..~.?..x.
000F0  39 50 2B 47 ¦ C0 C9 35 6A ¦ 42 A7 33 37 ¦ 8F 07 27 24   9P+G..5jB.37..'$
00100  D0 83 C8 6E ¦ AE 19 24 94 ¦ BC 10 77 49 ¦ 29 AF A3 B1   ...n..$...wI)...
00110  3E 16 89 4A ¦ 81 21 D2 70 ¦ 8F 0E EA C8 ¦ AA 23 3E A6   >..J.!.p.....#>.
00120  EE B6 47 B8 ¦ FE F4 01 43 ¦ 55 29 5D 4E ¦ 09 4E 3D 24   ..G....CU)]N.N=$
00130  52 C5 08 8B ¦ A5 E9 0B C7 ¦ B1 63 A6 E1 ¦ 95 1B F5 FB   R........c......
00140  C8 8A 5B 23 ¦ C5 DD 93 F9 ¦ 9F 10 F1 2A ¦ 5B F6 2F 14   ..[#.......*[./.
00150  1D 44 14 74 ¦ 05 84 BF 01 ¦ 6C 12 02 FF ¦ 4E 74 7C F3   .D.t....l...Nt|.
00160  39 AA 55 5F ¦ 7A B8 9F 6F ¦ 74 03 BD 0B ¦ BB F5 8C 12   9.U_z..ot.......
00170  24 F0 EF 1E ¦ C1 9C A9 7B ¦ 2A                          $......{*
```

Gambar 4-4 Isi berkas terenkripsi

- Dekripsi

Untuk dapat melakukan operasi dekripsi, pemakai perlu memberikan informasi mengenai ukuran kunci yang diinginkan, 160-bit atau 192-bit, serta berkas yang akan didekripsi. Bila operasi dekripsi berhasil, maka akan tercipta sebuah berkas baru dengan ekstensi "out".

Sebelum Dekripsi

```
AZTECC   EXE       135,168  01-01-00  1:00a aztecc.exe
GENFILE  EXE        40,960  01-01-00  1:00a genfile.exe
T100TX~1 TEF           377  01-01-00  1:00a t100.txt.tef
PUBKEY   PK            435  01-01-00  1:00a pubkey.pk
T100     TXT           100  01-01-00  1:00a t100.txt
         5 file(s)        177,040 bytes
         2 dir(s)   1,600,536,576 bytes free
```

Proses Dekripsi

```
E:\test>aztecc -d t100.txt.tef
=======================================================
AZTECC - an implementation of elliptic curve cryptosystem
Version 1.0 - Copyright (c) 1999-2000 by Tedi Heriyanto
```

```
This is a freeware edition. Use it with your own risk.
=========================================================
INPUT PASSPHRASE :
------------------
Hallo =>tdh<= please enter your passphrase (up to 2047 chars)
This is your PRIVATE KEY. Do not forget it.
If you forget it you'll never be able to decrypt the encrypted files.

When you're done, press <ENTER>.
Begin entering your passphrase ==>

Decrypting file : t100.txt.tef
Result will be stored in : t100.txt.out
Processing : .
Done decrypting =>t100.txt.tef<=
Elapsed time : 3 sec 62 hsec

                                  Setelah Dekripsi
AZTECC   EXE       135,168  01-01-00  1:00a aztecc.exe
GENFILE  EXE        40,960  01-01-00  1:00a genfile.exe
T100TX~1 TEF           377  01-01-00  1:00a t100.txt.tef
PUBKEY   PK            435  01-01-00  1:00a pubkey.pk
T100     TXT           100  01-01-00  1:00a t100.txt
T100TX~1 OUT           100  01-01-00  1:00a t100.txt.out
         6 file(s)        177,140 bytes
         2 dir(s)   1,600,532,480 bytes free
```

Gambar 4-5 Operasi Dekripsi

Isi berkas sesudah proses dekripsi adalah sebagai berikut :

```
00000  61 62 63 64 ¦ 65 66 67 68 ¦ 69 6A 6B 6C ¦ 6D 6E 6F 70   abcdefghijklmnop
00010  71 72 73 74 ¦ 75 76 77 78 ¦ 79 7A 61 62 ¦ 63 64 65 66   qrstuvwxyzabcdef
00020  67 68 69 6A ¦ 6B 6C 6D 6E ¦ 6F 70 71 72 ¦ 73 74 75 76   ghijklmnopqrstuv
00030  77 78 79 7A ¦ 61 62 63 64 ¦ 65 66 67 68 ¦ 69 6A 6B 6C   wxyzabcdefghijkl
00040  6D 6E 6F 70 ¦ 71 72 73 74 ¦ 75 76 77 78 ¦ 79 7A 61 62   mnopqrstuvwxyzab
00050  63 64 65 66 ¦ 67 68 69 6A ¦ 6B 6C 6D 6E ¦ 6F 70 71 72   cdefghijklmnopqr
00060  73 74 0D 0A ¦                                           st..

E:\test>diff t100.txt t100.txt.out
```

Gambar 4-6 Isi berkas terdekripsi

- Mencari *Point*

Untuk menggunakan menu ini, pemakai harus memasukkan informasi mengenai *prime field* yang digunakan dan koefisien persamaan kurva. Bila semua informasi tersebut telah dimasukkan, maka program akan mencari *point* dalam kurva eliptik tersebut, hasilnya akan ditampilkan ke layar.

```
E:\test>aztecc -f
=========================================================
AZTECC - an implementation of elliptic curve cryptosystem
Version 1.0 - Copyright (c) 1999-2000 by Tedi Heriyanto
This is a freeware edition. Use it with your own risk.
=========================================================
The elliptic curve used in this program has the following equation :
               y^2 = x^3 + a*x + b (mod q)
```

```
where q is a large prime number (160-bit or 192-bit).

FIND POINT
----------
Please enter prime field (p).Its length should be 160 or 192-bit :
627710173538668076383578942320766641610235544446403438995 7

Please input numbers to the following :
Coefficient a = 56726283015975321563605701574554303128030256739027848457 20
Coefficient b = 25186665606482171024216890126456895585774894579696592642 51
Point P :
(24357430723586790696002234873906790762282787091973394769 2,
48575231505669121480308354513597034603324980109432115591 9)
```

Gambar 4-7 Mencari sebuah *point*

- Menguji *Point*

Menu ini digunakan bila pemakai ingin memastikan bahwa *point* yang didapatnya benar-benar berada di kurva eliptik. Sebelum dapat menguji *point*, pemakai perlu memasukkan informasi mengenai *prime finited field* yang digunakan, koefisien persamaan kurva eliptik dan koordinat *point* yang ingin diuji. Setelah semua informasi tersebut dimasukkan, program akan memprosesnya, dan hasilnya akan ditampilkan pada layar.

```
E:\test>aztecc -t
=========================================================
AZTECC - an implementation of elliptic curve cryptosystem
Version 1.0 - Copyright (c) 1999-2000 by Tedi Heriyanto
This is a freeware edition. Use it with your own risk.
=========================================================
The elliptic curve used in this program has the following equation :
                y^2 = x^3 + a*x + b (mod q)
where q is a large prime number (160-bit or 192-bit).

TEST POINT
----------
Please enter prime field (p).Its length should be 160 or 192-bit :
627710173538668076383578942320766641610235544446403438995 7

Please input numbers to the following :
Coefficient a = 56726283015975321563605701574554303128030256739027848457 20
Coefficient b = 25186665606482171024216890126456895585774894579696592642 51
X-coordinate = 24357430723586790696002234873906790762282787091973394769 2
Y-coordinate = 48575231505669121480308354513597034603324980109432115591 9

Point (24357430723586790696002234873906790762282787091973394769 2,485752315056691
21480308354513597034603324980109432115591 9) is on the curve.
```

Gambar 4-8 Menguji sebuah *point*

4.1.8. Penjelasan Proses Enkripsi/Dekripsi

4.1.8.1. Enkripsi

Proses enkripsi pada program ini memerlukan informasi tentang *point* (P), kunci publik *user* yang akan menerima berkas (dP), koefisien persamaan kurva

eliptik, serta *prime field* yang digunakan. Semua informasi ini diperoleh dari *ring* kunci publik.

Secara sederhana, *ring* kunci publik yang digunakan memiliki struktur sebagai berikut :

username | *hash* (d) | a | b | x | y | x(dP) | y(dP) | q

dengan :

username adalah nama penerima berkas yang akan dienkripsi.
hash(d) adalah digest untuk kunci pribadi user tersebut.
a adalah koefisien a dalam persamaan kurva eliptik.
b adalah koefisien b dalam persamaan kurva eliptik.
x adalah koordinat x sebuah *point* pada kurva eliptik.
y adalah koordinat y sebuah *point* pada kurva eliptik.
x(dP) adalah koordinat x perkalian *point* P dengan kunci pribadi d.
y(dP) adalah koordinat y perkalian *point* P dengan kunci pribadi d.
q adalah *prime field* yang digunakan dalam seluruh operasi.

Setelah seluruh informasi tersedia, maka program akan melakukan inisialisasi kurva eliptik yaitu menetapkan modulo yang digunakan, menetapkan suatu *point*, menetapkan sebuah *point* berdasarkan kunci publik yang diberikan *user*.

Kemudian program menetapkan suatu bilangan k sebagai *session key*, lalu menghitung perkalian k dengan dP. Sebuah blok data diambil dan dienkripsi dengan cara meng-*xor*-kannya dengan koordinat X perkalian k.dP. Secara sederhana, algoritma proses ini adalah sebagai berikut :

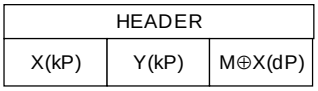
```
Selama blok belum habis, lakukan proses berikut :
    h <-- digit terendah koordinat k.dP.
    h <-- h xor text
    tulis h ke berkas.
```

Dalam bahasa C, algoritma tersebut dapat diimplementasikan sebagai berikut :

```
void encrypt_block(unsigned char* text, bigint xkdp, FILE* fp)
{
        unsigned long h;
        for(int i=0;i<blocksize;i+=4)
        {
                h = xkdp.least_significant_digit();
                h ^= ( (unsigned long) text[i]
                        | (unsigned long) text[i+1] << 8
                        | (unsigned long) text[i+2] << 16
                        | (unsigned long) text[i+3] << 24);
                fprintf(fp,"%c%c%c%c",char(h & 0x000000FF),
                        char((h>>8) & 0x000000FF),
                        char((h>>16) & 0x000000FF),
                        char(h>>24));
                xkdp >>= 32;

        }
}
```

Berkas terenkripsi memiliki format sebagai berikut :



dengan *header* berisi informasi id program, nama *user* yang akan menerima berkas, nama berkas, serta ukuran berkas.

4.1.8.2. Dekripsi

Proses dekripsi pada program ini memerlukan informasi tentang *username*, *point* (P), hash (d), koefisien persamaan kurva eliptik, serta *prime field* yang digunakan. Semua informasi ini diperoleh dari *ring* kunci publik.

Kemudian program membaca *ring* kunci publik serta informasi `kP` yang ada pada *ciphertext*. Setelah seluruh informasi tersedia, maka program akan melakukan inisialisasi kurva eliptik, meminta masukan kunci pribadi pemakai yang berupa `d`, lalu menghitung `d*kP` untuk setiap blok, koordinat X untuk d*kP di-*xor*-kan dengan blok pesan terenkripsi, hasilnya yang berupa *plaintext* ditulis ke berkas.

Secara sederhana, algoritma proses ini adalah sebagai berikut :

```
Selama blok belum habis, lakukan proses berikut :
    h <-- digit terendah koordinat d.kP.
    h <-- h xor pesan
    tulis h ke berkas.
```

Dalam bahasa C, algoritma tersebut dapat diimplementasikan sebagai berikut :

```
void decrypt_block(unsigned char* text, bigint xdkp)
{
        unsigned long h;

        for(int i=0;i<blocksize;i+=4)
        {
                h = xdkp.least_significant_digit();
                h ^= ((unsigned long) text[i]
                        |(unsigned long) text[i+1] << 8
                        |(unsigned long) text[i+2] << 16
                        |(unsigned long) text[i+3] << 24);

                text[i]   = char(h & 0x000000FF);
                text[i+1] = char((h>>8) & 0x000000FF);
                text[i+2] = char((h>>16) & 0x000000FF);
                text[i+3] = char(h>>24);

                xdkp >>= 32;
        }
}
```

4.2. Analisis

4.2.1. Hubungan antara Kecepatan Prosesor dengan Waktu Proses Enkripsi/Dekripsi

Metode: Dilakukan uji proses enkripsi/dekripsi dengan menggunakan kunci berukuran 160-bit pada beberapa macam prosesor dengan besar RAM 32 MB. Uji dilakukan terhadap berkas teks. Waktu yang dibutuhkan untuk melakukan proses enkripsi/dekripsi dicatat ke dalam Tabel 4.1. Kesemua uji dilakukan di bawah Sistem Operasi Microsoft Windows 9x.

Hasil :

Komputer	Proses	Waktu
Cyrix 150 MHz	Enkripsi Berkas Teks†	7:38,30
	Dekripsi Berkas Teks	2:56,42
Celeron-MMX 300A MHz	Enkripsi Berkas Teks	2:14,74
	Dekripsi Berkas Teks	58,93
P-II 333 MHz	Enkripsi Berkas Teks	2:13,00
	Dekripsi Berkas Teks	49,00

Tabel 4-1 Perbandingan Kecepatan Enkripsi/Dekripsi pada beberapa komputer

Keterangan :
†Berkas Teks yang digunakan berukuran 10 KB.

Analisis :

Berdasarkan tabel di atas, terlihat bahwa jenis prosesor yang digunakan mempengaruhi kecepatan proses enkripsi dan dekripsi. Proses enkripsi membutuhkan waktu terlama, lebih dari tujuh menit. Sementara proses enkripsi tercepat membutuhkan waktu lebih dari dua menit, rasio waktu terlama dengan waktu tercepat sekitar 3,5.

Secara umum proses enkripsi membutuhkan waktu lebih lama daripada proses dekripsi. Berdasarkan tabel, waktu proses enkripsi sekitar tiga kali lebih lambat daripada proses dekripsi.

4.2.2. Hubungan antara Panjang Kunci dengan Waktu Proses Enkripsi/Dekripsi

Metode: Dilakukan uji proses enkripsi/dekripsi dengan menggunakan kunci berukuran 160-bit dan 192-bit pada komputer dengan prosesor Cyrix 150 MHz dan RAM 32 MB. Uji dilakukan terhadap berkas teks. Waktu yang

dibutuhkan untuk melakukan proses enkripsi/dekripsi dicatat ke dalam Tabel 4.2. Kesemua uji dilakukan di bawah Sistem Operasi Microsoft Windows 98.

Hasil :

Proses	Panjang Kunci	Waktu
Enkripsi *	160-bit	5:45,37
	192-bit	7:16,44
Enkripsi **	160-bit	5:47,18
	192-bit	7:16,11
Dekripsi *	160-bit	2:32,91
	192-bit	2:40,27
Dekripsi **	160-bit	2:31,98
	192-bit	2:40,71

Tabel 4-2 Perbandingan Panjang Kunci dengan Kecepatan Enkripsi/Dekripsi

Keterangan :
* Berkas yang digunakan adalah berkas biner dengan ukuran 10 KB.
** Berkas yang digunakan adalah berkas teks dengan ukuran 10 KB.

Analisis :

Berdasarkan tabel di atas, terlihat bahwa proses enkripsi dan dekripsi dengan menggunakan ukuran kunci 192-bit memerlukan waktu yang lebih lama daripada proses enkripsi dan dekripsi dengan menggunakan ukuran kunci 160-bit. Hal ini dikarenakan dengan semakin besarnya ukuran kunci yang digunakan, maka operasi-operasi aritmatika kurva eliptik dilakukan sesuai dengan besarnya ukuran kunci.

Penggunaan ukuran kunci berukuran 192-bit, menyebabkan operasi dilakukan dengan jumlah bit 192 sementara untuk kunci berukuran 160-bit, operasi dilakukan dengan jumlah bit 160. Dengan semakin banyaknya bit yang digunakan dalam operasi, maka kecepatan operasi akan semakin berkurang. Jadi secara umum, dengan menggunakan ukuran kunci yang lebih besar, waktu operasi relatif akan lebih lama daripada bila menggunakan ukuran kunci yang lebih kecil.

Operasi enkripsi dan dekripsi pada berkas teks maupun biner hampir memiliki waktu yang sama. Perbedaan mungkin disebabkan oleh perubahan beban pada sistem komputer yang digunakan.

4.2.3. Hubungan antara Panjang Kunci dengan Ukuran Berkas

Metode : Dilakukan uji hubungan antara panjang kunci dengan ukuran berkas sebelum dan setelah proses enkripsi, menggunakan kunci berukuran 160-bit dan 192-bit, pada komputer dengan prosesor Cyrix 150 MHz dan RAM 32 MB. Uji dilakukan terhadap berkas teks. Ukuran berkas sebelum dan setelah proses enkripsi dicatat ke dalam Tabel 4.3, selain itu perbedaan antara keduanya juga dicatat. Kesemua uji dilakukan di bawah Sistem Operasi Microsoft Windows 98.

Hasil :

Panjang Kunci	[2]Ukuran Berkas Sebelum Enkripsi (byte)	[3]Ukuran Berkas Setelah Enkripsi (byte)	Rasio (3/2)
160-bit	100	377	3,77
	1.000	3.079	3,02
	2.000	6.079	3,02
	5.000	15.079	3,02
	10.000	30.081	3,01
192-bit	100	377	3,77
	1.000	3.043	3,04
	2.000	6.067	3,03
	5.000	15.067	3,01
	10.000	30.045	3,00

Tabel 4-3 Hubungan antara Panjang Kunci dengan Ukuran Berkas

Analisis :

Berdasarkan tabel di atas, terlihat bahwa ukuran berkas sebelum dan setelah proses enkripsi berbeda. Hal ini dikarenakan pada saat proses enkripsi, berkas semula diberi informasi tentang *user*, nama berkas dan ukuran berkas, selain itu di dalam berkas yang terenkripsi terdapat informasi mengenai koordinat X kP dan koordinat Y kP, yang masing-masing berukuran satu *blocksize* sehingga di dalam berkas terenkripsi minimal memiliki ukuran = (1+1+1) *blocksize*. Dengan demikian secara keseluruhan berkas terenkripsi memiliki ukuran tiga kali lebih besar daripada berkas semula. Di dalam tabel, terlihat bahwa rasio antara berkas sebelum enkripsi dan sesudah enkripsi adalah sekitar tiga.

Secara umum terlihat bahwa dengan semakin besarnya ukuran berkas, maka rasio antara berkas setelah enkripsi dan berkas sebelum enkripsi menjadi semakin menurun dan mendekati angka tiga.

Berdasarkan tabel, terlihat pula bahwa dengan menggunakan ukuran kunci yang lebih besar, berkas hasil enkripsi secara umum menjadi lebih kecil. Hal ini dapat dijelaskan sebagai berikut :

Pada proses enkripsi dengan menggunakan ukuran kunci 160-bit, berkas dibagi ke dalam beberapa buah blok yang masing-masing berukuran 20 byte. Jadi untuk ukuran berkas 1000 byte, jumlah blok ada 50 buah, dengan tiap blok memiliki ukuran 60 byte, karena menyimpan informasi koordinat X (kP), koordinat Y (kP) dan pesan yang masing-masing berukuran 20 karakter. Oleh karena berkas adalah berkas teks, pada akhir berkas terdapat tanda *Carriage Return* dan *Linefeed* (13H dan 10H), tanda ini memerlukan sebuah blok lagi. Sehingga ukuran berkas terenkripsi adalah : 51 * 60 = 3060 dan ditambah dengan informasi *header* pada awal berkas terenkripsi sebesar 19 byte. Ukuran besar terenkripsi adalah 3060+19 = 3079 byte.

Sementara pada proses enkripsi dengan menggunakan ukuran kunci 192-bit, berkas dibagi ke dalam beberapa buah blok yang masing-masing berukuran 24 byte. Jadi untuk ukuran berkas 1000 byte, jumlah blok ada 42 buah, dengan tiap blok memiliki ukuran 72 byte, karena menyimpan informasi koordinat X (kP), koordinat Y (kP) dan pesan yang masing-masing berukuran 24 karakter. Tanda akhir berkas teks yaitu *Carriage Return* dan *Linefeed* (13H dan 10H) tidak memerlukan sebuah blok lagi, karena ukuran blok mampu menampungnya. Sehingga ukuran berkas terenkripsi adalah : 42 * 72 = 3024 dan ditambah dengan informasi *header* pada awal berkas terenkripsi sebesar 19 byte. Ukuran besar terenkripsi adalah 3024+19 = 3043 byte.

4.2.4. Hubungan antara Ukuran Berkas dengan Waktu Enkripsi/Dekripsi

Metode : Dilakukan uji hubungan antara ukuran berkas dengan waktu proses enkripsi, menggunakan kunci berukuran 160-bit dan 192-bit, pada komputer dengan prosesor Cyrix 150 MHz dan RAM 32 MB. Uji dilakukan terhadap berkas teks. Berbagai ukuran berkas dienkripsi/didekripsi kemudian waktu

prosesnya dicatat ke dalam Tabel 4.4. Kesemua uji dilakukan di bawah Sistem Operasi Microsoft Windows 98.

Hasil :

Ukuran Kunci	Ukuran Berkas (byte)	Waktu Enkripsi (dt)	Waktu Dekripsi (dt)
160-bit	100	4,89	3,84
	1.000	35,43	17,24
	2.000	1:09,87	32,95
	5.000	2:51,81	1:16,79
	10.000	7:38,30	2:56,42
192-bit	100	5,55	4,17
	1.000	44,16	17,96
	2.000	1:30,41	34,00
	5.000	3:37,70	1:21,70
	10.000	8:35,31	3:30,69

Tabel 4-4 Hubungan antara ukuran berkas dengan waktu enkripsi/dekripsi

Analisis :

Berdasarkan tabel di atas, terlihat bahwa dengan semakin besarnya ukuran berkas, maka waktu yang dibutuhkan untuk enkripsi ataupun dekripsi juga semakin bertambah. Hal ini dikarenakan dengan semakin besarnya ukuran berkas, maka semakin banyak jumlah blok yang ada di dalamnya, dengan tiap blok terdiri dari 20 atau 24 karakter.

Jumlah blok yang semakin banyak menyebabkan program membutuhkan waktu proses yang lebih lama karena :

Untuk proses enkripsi :

- program mencari suatu bilangan random k, untuk tiap blok.
- menghitung perkalian k dengan suatu *point* P (kP).
- menghitung perkalian k dengan suatu kunci publik (dP).
- meng-xor-kan koordinat X perkalian tersebut dengan pesan.
- menuliskan (k.P) dan hasil enkripsi ke berkas.

Untuk proses dekripsi :

- program menerima d.
- menghitung perkalian d dengan kP

- meng-*xor*-kan koordinat X perkalian tersebut dengan pesan terenkripsi.
- menuliskan hasil dekripsi ke berkas.

Selain itu, berdasarkan tabel terlihat pula bahwa proses enkripsi memerlukan waktu yang relatif lebih lama daripada waktu untuk proses dekripsi, hal ini dikarenakan proses enkripsi melakukan lebih banyak perhitungan daripada proses dekripsi. Proses enkripsi perlu menghitung perkalian k dengan *point* P dan perkalian k dengan kunci publik dP, sementara proses dekripsi hanya perlu menghitung perkalian kunci pribadi (d) dengan (kP). Akibatnya, proses enkripsi menjadi relatif lebih lambat dibandingkan proses dekripsi. Dengan menggunakan berkas yang semakin besar, maka perbedaan waktu antara proses enkripsi dan dekripsi menjadi semakin besar.

BAB 5
PENUTUP

5.1. Kesimpulan

Beberapa buah kesimpulan yang dapat diambil setelah melaksanakan Tugas Akhir ini adalah :

- Elliptic Curve Cryptosystem dapat digunakan dalam pembuatan program enkripsi/dekripsi.
- Ciphertext yang dihasilkan program tergantung pada *plaintext*, ukuran kurva yang digunakan, *passphrase* yang dimasukkan, parameter kurva dan *point* yang dipilih.
- Ciphertext yang dihasilkan memiliki ukuran lebih besar daripada *plaintext*, karena adanya penambahan informasi pada *ciphertext*.
- Proses enkripsi/dekripsi dengan ukuran kurva eliptik 160-bit relatif lebih cepat dibandingkan proses enkripsi/dekripsi dengan ukuran kurva eliptik 192-bit.
- Kecepatan program tergantung pada jenis komputer serta konfigurasi yang digunakan.
- Proses dekripsi relatif lebih cepat dibandingkan dengan proses enkripsi, karena jumlah operasi yang lebih sedikit.
- Penggunaan *one-way hash function* seperti MD5 dapat menambah fleksibilitas kunci yang dimasukkan oleh pemakai.

5.2. Saran

Adapun beberapa saran yang dapat digunakan untuk pengembangan lebih lanjut Tugas Akhir ini adalah sebagai berikut :

- Melakukan optimisasi pada program terutama pada fungsi-fungsi yang menangani aritmatika kurva eliptik.
- Mengurangi besarnya penambahan informasi pada *ciphertext*. Salah satu yang dapat dilakukan adalah dengan menggunakan metode *point compression*, sehingga yang tersimpan dalam *ciphertext* hanya salah satu koordinat *point*

saja. Di dalam program ini informasi suatu *point* terdiri dari koordinat X dan koordinat Y yang memiliki ukuran 20 atau 24 byte dan keduanya disimpan untuk setiap blok *ciphertext*.

- Untuk mempercepat proses enkripsi/dekripsi sebaiknya algoritma yang digunakan untuk enkripsi/dekripsi blok diganti dengan algoritma enkripsi *block cipher* seperti RC6, sehingga dapat mempercepat proses. Sedangkan kurva eliptik dapat digunakan untuk menangani manajemen kunci.
- Pemanfaatan dalam aplikasi-aplikasi ataupun peralatan-peralatan yang memiliki keterbatasan ruang atau memori.
- Perbaikan pada manajemen kunci publik dan kunci privat, termasuk peningkatan keamanan pada keduanya. Perbaikan tersebut dapat berupa penambahan fasilitas untuk melakukan ekspor file kunci publik ke suatu file dan fasilitas impor untuk menambahkan suatu kunci publik user lain ke dalam kunci publik yang ada di dalam komputer user (*public key ring*). Selain itu pada Tugas Akhir ini, kunci publik disimpan dalam suatu file teks biasa sehingga keamanannya kurang terjamin, di dalam pengembangan selanjutnya hal ini mungkin dapat diperbaiki.
- Perbaikan pada fasilitas penanganan kesalahan. Tugas Akhir yang dibuat ini belum secara ekstensif menelusuri kemungkinan adanya *bugs* tersembunyi pada program, selain itu fasilitas penanganan kesalahan yang ada masih belum *user friendly*, pesan kesalahan yang ditampilkan mungkin akan semakin membingunkan pemakai.
- Perbaikan pada interface yang digunakan. Tugas Akhir yang dibuat ini menggunakan *command-line interface* sehingga mungkin menyulitkan bagi mereka yang tidak terbiasa menggunakan aplikasi semacam ini. Untuk itu dalam pengembangan berikutnya dapat diadakan perbaikan pada interface ini, misalnya dengan penyertaan interface grafik interface grafik berbasis GNOME dan KDE pada sistem operasi Linux/BSD ataupun berbasis Microsoft Windows 9x/NT.