STUDI DAN IMPLEMENTASI *THE BLOWFISH ENCRYPTION ALGORITHM* DALAM BAHASA PEMROGRAMAN C++

Muhammad Riza Putra – NIM : 13505108

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
*E-mail : **if15108@students.if.itb.ac.id***

Abstraksi

Makalah ini membahas tentang pengenalan, pemahaman dan implementasi ***The Blowfish Encryption Algorithm*** dalam bahasa pemrograman C++**.** Blowfish merupakan suatu metode enkripsi yang mirip dengan DES (*DES-like cipher*) dan diciptakan oleh **Bruce Schneier** yang ditujukan untuk mikroposesor besar (32 bit ke atas dengan *cache* data yang besar). Blowfish dikembangkan untuk memenuhi kriteria disain sebagai berikut:

- Cepat, pada implementasi yang optimal Blowfish dapat mencapai kecepatan 26 *clock cycle* per byte.
- Kompak, Blowfish dapat berjalan pada memori kurang dari 5 KB.
- Sederhana, Blowfish hanya menggunakan operasi yang simpel: penambahan (*addition*), XOR, dan penelusuran tabel (*table lookup*) pada *operand* 32 bit. Desainnya mudah untuk dianalisa yang membuatnya resisten terhadap kesalahan implementasi.
- Keamanan yang variabel, panjang kunci Blowfish dapat bervariasi dan dapat mencapai 448 bit (56 byte).

Blowfish dioptimasikan untuk aplikasi dimana kunci tidak sering berubah, seperti jalur komunikasi atau enkripsi file otomatis. Blowfish jauh lebih cepat dari DES bila diimplementasikan pada 32 bit mikroprosesor dengan cache data yang besar, seperti Pentium dan Power PC, Blowfish tidak cocok untuk aplikasi seperti *packet switching*, dengan perubahan kunci yang sering, atau sebagai fungsi *hash* satu arah. Kebutuhan memorinya yang besar tidak memungkinkan untuk aplikasi kartu pintar (*smart card*).

***Kata kunci**: The Blowfish Encryption Algorithm, Bruce Schneier, Advanced Encryption Standard, DES, XOR encryption, Sub-Key, SBoxes, enkripsi, dekripsi.*

1. Pendahuluan

"Paranoia is very useful in this work. ...If your cryptographic system can survive the paranoia model, it has at least a fighting chance of surviving in the real world." - Niels Ferguson & Bruce Schneier.

Salah satu hal yang penting dalam komunikasi menggunakan computer untuk menjamin kerahasian data adalah enkripsi. Enkripsi adalah sebuah proses yang melakukan perubahan sebuah kode dari yang bisa dimengerti menjadi sebuah kode yang tidak bisa dimengerti (tidak terbaca). Enkripsi dapat diartikan sebagai kode atau chiper. Sebuah sistem pengkodean menggunakan suatu table atau kamus yang telah didefinisikan untuk mengganti kata dari informasi atau yang merupakan bagian dari informasi yang dikirim. Sebuah chiper menggunakan suatu algoritma yang dapat mengkodekan semua aliran data (stream) bit dari sebuah pesan menjadi cryptogram yang tidak dimengerti (unitelligible). Karena teknik cipher merupakan suatu sistem yang telah siap untuk di automasi, maka teknik ini digunakan dalam sistem keamanan komputer dan network.
Enkripsi merupakan proses mengkodekan data sehingga maknanya menjadi tidak jelas/sulit dibaca. Enkripsi juga dapat diartikan mengubah sebuah kata atau kalimat menjadi sandi/kode-kode tertentu dengan menggunakan algoritma tertentu pula. Tujuan dari enkripsi

adalah meningkatkan dan menjaga keamanan data baik yang disimpan maupun yang dikirim.

Algoritma penyajian data yang akan penulis terapkan dalam pembahasan selanjutnya adalah metode enkripsi yang lebih dikenal dengan sebutan ***The Blowfish Encryption Algorithm***.

2. Deskripsi singkat Mengenai Blowfish

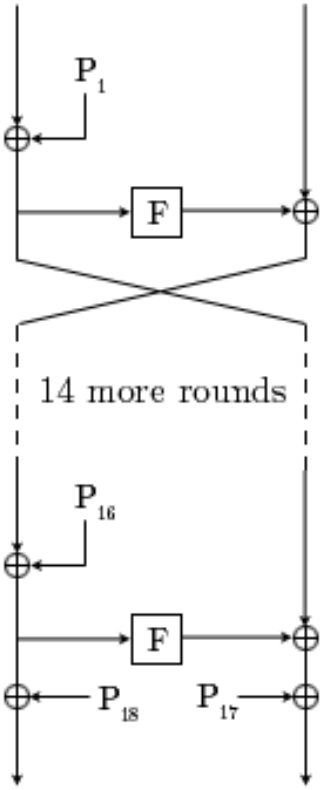


Diagram Struktur Blowfish

Blowfish merupakan blok cipher 64-bit dengan panjang kunci variabel. Algoritma ini terdiri dari dua bagian: *key expansion* dan enkripsi data. *Key expansion* merubah kunci yang dapat mencapai 448 bit menjadi beberapa array subkunci (*subkey*) dengan total 4168 byte.

Enkripsi data terdiri dari iterasi fungsi sederhana sebanyak 16 kali. Setiap putaran terdiri dari permutasi kunci-*dependent* dan substitusi kunci- dan data-*dependent*. Semua operasi adalah penambahan dan XOR pada variable 32-bit. Tambahan operasi lainnya hanyalah empat penelusuran tabel (*table lookup*) array berindeks untuk setiap putaran.

Blowfish menggunakan subkunci yang besar. Kunci ini harus dihitung sebelum enkripsi atau dekripsi data.

Array P terdiri dari delapan belas 32-bit subkunci:

$P_1, P_2, \ldots, P_{18}$

Empat 32-bit S-box masing-masing mempunyai 256 entri:

$S_{1,0}, S_{1,1}, \ldots, S_{1,255}$

$S_{2,0}, S_{2,1}, \ldots, S_{2,255}$

$S_{3,0}, S_{3,1}, \ldots, S_{3,255}$

$S_{4,0}, S_{4,1}, \ldots, S_{4,255}$

Metoda selengkapnya untuk menghitung subkunci ini akan dijelaskan pada bagian bawah.

Blowfish merupakan algoritma yang menerapkan jaringan Feistel (*Feistel network*) yang terdiri dari 16 putaran. Input merupakan elemen 64 bit, *X*. Untuk mengenkrip:

Bagi *X* menjadi dua 32-bit: X_L, X_R

untuk $i = 1$ sampai 16

$X_L = X_L \text{ xor } P_i$

$X_R = F(X_L) \text{ xor } X_R$

Tukar X_L dan X_R

Tukar X_L dan X_R (batalkan penukaran terakhir)

$X_R = X_R \text{ xor } P_{17}$

$X_L = X_L \text{ xor } P_{18}$

Kombinasikan kembali X_L dan X_R

Fungsi F adalah sebagai berikut:

Bagi X_L, menjadi empat bagian 8-bit: *a*, *b*, *c* dan *d*

$F(X_L) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \text{ xor } S_{3,c}) + S_{4,c} \bmod 2^{32}$

Dekripsi sama persis dengan enkripsi, kecuali P_1, P_2, . . . , P_{18} digunakan pada urutan yang terbalik.

Subkunci dihitung menggunakan algoritma Blowfish, metodanya adalah sebagai berikut:

1. Pertama-tama inisialisasi P-array dan kemudian empat S-box secara berurutan dengan string yang tetap. String ini terdiri digit hexadesimal dari pi.
2. XOR P_1 dengan 32 bit pertama kunci, XOR P_2 dengan 32 bit kedua dari kunci dan seterusnya untuk setiap bit dari kunci (sampai P_{18}). Ulangi terhadap bit kunci sampai seluruh P-array di XOR dengan bit kunci.
3. Enkrip semua string nol dengan algoritma Blowfish dengan menggunakan subkunci seperti dijelaskan pada langkah (1) dan (2).
4. Ganti P1 dan P2 dengan keluaran dari langkah (3)
5. Enkrip keluaran dari langkah (3) dengan algoritma Blowfish dengan subkunci yang sudah dimodifikasi.
6. Ganti P_3 dan P_4 dengan keluaran dari langkah (5).
7. Lanjutkan proses tersebut, ganti seluruh elemen dari P-array, dan kemudian seluruh keempat S-box berurutan, dengan keluaran yang berubah secara kontinyu dari algoritma Blowfish.

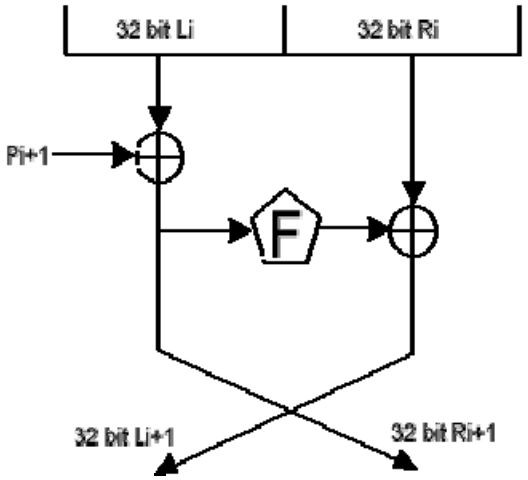
Total diperlukan 521 iterasi untuk menghasilkan semua subkunci yang dibutuhkan. Aplikasi kemudian dapat menyimpan subkunci ini dan tidak dibutuhkan langkah-langkah proses penurunan ini berulang kali, kecuali kunci yang digunakan berubah.

2.1 Enkripsi Blowfish

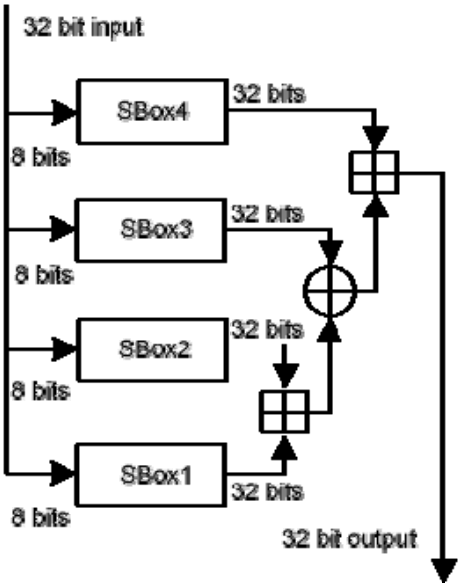
Blowfish membutuhkan 64 bit blok-blok plaintext sebagai masukannya dan menghasilkan 64 bit chipertext. Ukuran kunci untuk Blowfish dapat dipilih dalam range 32 bit sampai 448 bit yang mana semakin besar ukurannya maka semakin kuat keamanannya. Blok masukan dipecah dalam paro L0 dan R0 dimana tiap-tiap paro tersebut mengandung 32 bit. Blowfish dapat secara sederhananya digambarkan dengan algoritma berikut:

```
j = 1
loop from j to 16
         Rj = Lj-1 XoR Pj
         Lj = F(Rj) XoR Rj-1
end loop
L17 = R16 XoR P18
R17 = L16 XoR P17
```

Dimana P adalah sub-kunci dan F adalah fungsi kompleks. L17 dan R17 mengandung chipertext. Perhatikan bahwa ada 16 putaran Xor dan operasi fungsi F. Berikut diagram putaran Blowfish.



Fungsi kompleks F, ditunjukkan oleh gambar berikut.



2.2 Dekripsi Blowfish

Dekripsi untuk Blowfish bersifat maju kedepan. Ironisnya, dekripsi bekerja dalam arah algoritma yang sama seperti halnya dengan enkripsi, namun sebagai masukannya dalah chipertext. Walaupun begitu, seperti yang diharapkan, sub-kunci yang digunakan dalam urutan terbalik. Sehingga algoritma dekripsi Blowfish sebagai berikut:

```
j = 1
loop from j to 16
          Rj = Lj-1 XoR P19-j
          Lj = F(Rj)XoR Rj-1
end loop
L17 = R16 XoR P1
R17 = L16 XoR P2
```

2.3 Sub-Kunci , Sboxes dan P-array

Pembuatan sub-kunci dan Sboxes dapat djelaskan dalam 3 tahapan berikut. Anggap bahwa panjang kunci mungkin dari 32 bit sampai 448 bit. Kunci ini kemudian digunakan untuk membuat 4 Sboxes dan 18 32 bit sub-kunci. SbOxes memiliki 8 x 32 struktur yang mana totalnya adalah 256 32bit elemen. P-array disimpan dalam sub-kunci.

Langkah 1 : P-array diinisialisasikan secara terurut menggunakan bit dari konstanta pi. Misalkan P1 diisi dengan 32bit terkiri dari pi, dan begitu juga keempatnya. Akhirnya 4 Sboxes sudah diinisialisasi.

Langkah 2 : Sebuah logika XoR dikendalikan dengan elemen-elemen array dari kunci dan sub-kunci elemen P-array. Misalkan, Pi = Pi XoR Kj…

Langkah 3 : Sekarang seharusnya sudah terdapat 64 bit blok. Ambil blok tersebut dan enkripsikan dengan menggunakan proses Blowfish. Pi dan Pi +1 kemudian akan diganti dengan hasil ini dan kemudian I diinkrementasikan. Lanjutkan langkah ini sampai semua elemen P-array telah diganti dan kemudian urutkan semua 4 Sboxes.

3. Studi mengenai Algoritma Blowfish

3.1 Algoritma Blowfish

Algoritma Blowfish terdiri atas dua bagian: bagian ekspansi kunci dan bagian enkripsi data. Ekspansi kunci mengubah kunci sampai maksimal 448 bit menjadi beberapa array subkunci total sampai 4168 byte.

Enkripsi data dilakukan melalui suatu susunanjaringan Feistel dengan jumlah putaran 16 kali. Setiap putaran terdiri dari key-dependentpermutation dan key-and-data-dependent substitution. Semua operasi adalah operasi Xor dan penjumlahan pada 32-bit words. Operasi tambahan adalah empat buah lookups data array untuk setiap putaran. Blowfish memiliki:

1. Delapan belas buah P-array yang berisi 32 bit subkunci.
2. Empat buah 32-bit S-boxes dengan 256 entri.

3.2 Subkunci

Blowfish menggunakan subkunci yang berjumlah banyak. Subkunci ini harus dikomputasi sebelum enkripsi maupun dekripsi data.

Algoritma pembangkitan subkuncinya adalah sebagai berikut:

1. Inisialisasi P-array pertama dan 4 buah S-boxes dengan suatu angka konstan. Angka ini harus mengandung digit heksadesimal dari bilangan pi (kecuali angka 3 di depan koma). Contohnya:

   P1 = 0x243f6a88

   P2 = 0x85a308d3

   P3 = 0x13198a2e

   P4 = 0x03707344

2. Lakukan operasi Xor P1 dengan 32 bit pertama dari kunci, P2 dengan 32 bit berikutnya, dan seterusnya.
3. Enkripsi semua string yang bernilai 0 dengan algoritma Blowfish menggunakan subkunci pada langkah 1 dan 2.
4. Ganti P1 dan P2 dengan hasil dari langkah 3.
5. Enkripsi hasil dari langkah 3 menggunakan algoritma Blowfish dengan subkunci yang telah dimodifikasi.

6. Ganti P3 dan P4 dengan hasil dari langkah 5.
7. Lakukan langkah-langkah di atas berulang kali, ganti semua entri P-array dan keempat S-boxes dengan hasil dari algoritma Blowfish.

Total ada 521 iterasi yang diperlukan untuk membangkitkan semua subkunci. Dalam notasi algoritmik, pembangkitan subkuncinya adalah:

```
Input:
   K : The key - 32 bits or more
   PI: The binary
representation of the fractional
portion of "pi"
     = 3.1415927... - 3.0
     = 2/16 + 4*/16**2 + 3/16**3 +
15/16**4 + ...
     =
0x243f6a8885a308d313198a2e03707
344...

Output:
    P1, P2, ..., P18: 18 32-bit sub-
keys

    S1[], S2[], S3[], S4[]: 4 S-
boxes, 32-bit 256-element arrays


Algorithm:
    (P1, P2, ..., P18, S1[],
    S2[], S3[], S4[]) = PI
    K' = (K, K, K, ...), Repeat the
key to 18*32 bits long
    (P1, P2, ..., P18) = (P1,
    P2, ..., P18) XOR K'

    T = (0x000000, 0x000000),

Setting initial clear text

    T   =   Blowfish(T),   Applying
Blowfish algorithm

    (P1, P2) = T, Updating first two
sub-keys

    T   =   Blowfish(T),   Applying
Blowfish again

    (P3, P4) = T

    ......

    T = Blowfish(T)

    (P17, P18) = T

    T = Blowfish(T)

    (S1[0], S1[1]) = T

    T = Blowfish(T)

    (S1[2], S1[3]) = T

    ......

    T = Blowfish(T)

    (S1[254], S1[255]) = T

    T = Blowfish(T)

    (S2[0], S2[1]) = T

    ......

    T = Blowfish(T)

    S4[254], S4[255]) = T
```

3.3 Enkripsi dan Dekripsi

Blowfish merupakan suatu jaringan Feistel dengan 16 putaran. Inputnya adalah data 64 bit, x. Algoritma enkripsinya adalah sebagai berikut:

```
Bagi x menjadi xL dan xR yang
berukuran 32 bit.

for i=1 to 16

      xL = xL  Pi

      xR = F(xL)  xR

      Tukar xL dengan xR

Tukar xL dengan xR /*Batalkan
pertukaran terakhir*/

xR = xR P17

xL = xL P18

Gabung xL dengan xR
```

Fungsi F yang dipakai adalah sebagai berikut:

- Bagilah xL menjadi 4 dengan ukuranmasing-masing 8 bit a,b,c,d.
- F(xL) = ((S1, a + S2, b mod 232) S3, c) + S4, d mod 232

Algoritma dekripsi sama saja dengan algoritma untuk enkripsi data. Hanya saja penggunaan P1, P2, P3, ..., P18 adalah kebalikan dari enkripsi.

4. Penerapan Algoritma Blowfish pada Bahasa Pemrograman

Berikut source code algoritma Blowfish syang ditulis dalam bahasa pemrograman C++.

C++ Header File : blowfish.h

```
// Header File
// blowfish.h     interface file for blowfish.cpp
// _THE BLOWFISH ENCRYPTION ALGORITHM_
// by Bruce Schneier

#define MAXKEYBYTES    56             // 448 bits max
#define NPASS           16            // SBox passes

#define DWORD                 unsigned long
#define WORD           unsigned short
#define BYTE           unsigned char

class CBlowFish
{
private:
       DWORD          * PArray ;
       DWORD          (* SBoxes)[256];
       void           Blowfish_encipher (DWORD *xl, DWORD *xr) ;
       void           Blowfish_decipher (DWORD *xl, DWORD *xr) ;

public:
                      CBlowFish () ;
                      ~CBlowFish () ;
       void           Initialize (BYTE key[], int keybytes) ;
       DWORD          GetOutputLength (DWORD lInputLong) ;
       DWORD          Encode (BYTE * pInput, BYTE * pOutput, DWORD lSize) ;
       void           Decode (BYTE * pInput, BYTE * pOutput, DWORD lSize) ;
} ;

// choose a byte order for your hardware
#define ORDER_DCBA     // chosing Intel in this case

#ifdef ORDER_DCBA      // DCBA - little endian - intel
       union aword {
         DWORD dword;
         BYTE byte [4];
         struct {
           unsigned int byte3:8;
           unsigned int byte2:8;
           unsigned int byte1:8;
           unsigned int byte0:8;
         } w;
       };
#endif

#ifdef ORDER_ABCD      // ABCD - big endian - motorola
       union aword {
         DWORD dword;
         BYTE byte [4];
         struct {
           unsigned int byte0:8;
           unsigned int byte1:8;
           unsigned int byte2:8;
           unsigned int byte3:8;
         } w;
       };
#endif

#ifdef ORDER_BADC      // BADC - vax
       union aword {
         DWORD dword;
         BYTE byte [4];
         struct {
           unsigned int byte1:8;
           unsigned int byte0:8;
           unsigned int byte3:8;
           unsigned int byte2:8;
         } w;
};
#endif
```

C++ SourceFile : blowfish.cpp

```
// blowfish.cpp   C++ class implementation of the BLOWFISH encryption algorithm
// _THE BLOWFISH ENCRYPTION ALGORITHM_
// by Bruce Schneier
// Revised code--3/20/94
// Converted to C++ class 5/96, Jim Conger

#include "blowfish.h"
#include "blowfish.h2" // holds the random digit tables

#define S(x,i) (SBoxes[i][x.w.byte##i])
#define bf_F(x) (((S(x,0) + S(x,1)) ^ S(x,2)) + S(x,3))
#define ROUND(a,b,n) (a.dword ^= bf_F(b) ^ PArray[n])


CBlowFish::CBlowFish ()
{
       PArray = new DWORD [18] ;
       SBoxes = new DWORD [4][256] ;
}

CBlowFish::~CBlowFish ()
{
       delete PArray ;
       delete [] SBoxes ;
}

       // the low level (private) encryption function
void CBlowFish::Blowfish_encipher (DWORD *xl, DWORD *xr)
{
       union aword  Xl, Xr ;

       Xl.dword = *xl ;
       Xr.dword = *xr ;

       Xl.dword ^= PArray [0];
       ROUND (Xr, Xl, 1) ;  ROUND (Xl, Xr, 2) ;
       ROUND (Xr, Xl, 3) ;  ROUND (Xl, Xr, 4) ;
       ROUND (Xr, Xl, 5) ;  ROUND (Xl, Xr, 6) ;
       ROUND (Xr, Xl, 7) ;  ROUND (Xl, Xr, 8) ;
       ROUND (Xr, Xl, 9) ;  ROUND (Xl, Xr, 10) ;
       ROUND (Xr, Xl, 11) ; ROUND (Xl, Xr, 12) ;
       ROUND (Xr, Xl, 13) ; ROUND (Xl, Xr, 14) ;
       ROUND (Xr, Xl, 15) ; ROUND (Xl, Xr, 16) ;
       Xr.dword ^= PArray [17] ;

       *xr = Xl.dword ;
       *xl = Xr.dword ;
}

       // the low level (private) decryption function
void CBlowFish::Blowfish_decipher (DWORD *xl, DWORD *xr)
{
   union aword  Xl ;
   union aword  Xr ;

   Xl.dword = *xl ;
   Xr.dword = *xr ;

   Xl.dword ^= PArray [17] ;
   ROUND (Xr, Xl, 16) ;  ROUND (Xl, Xr, 15) ;
   ROUND (Xr, Xl, 14) ;  ROUND (Xl, Xr, 13) ;
   ROUND (Xr, Xl, 12) ;  ROUND (Xl, Xr, 11) ;
   ROUND (Xr, Xl, 10) ;  ROUND (Xl, Xr, 9) ;
   ROUND (Xr, Xl, 8) ;   ROUND (Xl, Xr, 7) ;
   ROUND (Xr, Xl, 6) ;   ROUND (Xl, Xr, 5) ;
   ROUND (Xr, Xl, 4) ;   ROUND (Xl, Xr, 3) ;
   ROUND (Xr, Xl, 2) ;   ROUND (Xl, Xr, 1) ;
   Xr.dword ^= PArray[0];

   *xl = Xr.dword;
   *xr = Xl.dword;
}
```

```
        // constructs the enctryption sieve
void CBlowFish::Initialize (BYTE key[], int keybytes)
{
        int             i, j ;
        DWORD           data, datal, datar ;
        union aword temp ;

        // first fill arrays from data tables
        for (i = 0 ; i < 18 ; i++)
                PArray [i] = bf_P [i] ;

        for (i = 0 ; i < 4 ; i++)
        {
                for (j = 0 ; j < 256 ; j++)
                        SBoxes [i][j] = bf_S [i][j] ;
        }


        j = 0 ;
        for (i = 0 ; i < NPASS + 2 ; ++i)
        {
                temp.dword = 0 ;
                temp.w.byte0 = key[j];
                temp.w.byte1 = key[(j+1) % keybytes] ;
                temp.w.byte2 = key[(j+2) % keybytes] ;
                temp.w.byte3 = key[(j+3) % keybytes] ;
                data = temp.dword ;
                PArray [i] ^= data ;
                j = (j + 4) % keybytes ;
        }

        datal = 0 ;
        datar = 0 ;

        for (i = 0 ; i < NPASS + 2 ; i += 2)
        {
                Blowfish_encipher (&datal, &datar) ;
                PArray [i] = datal ;
                PArray [i + 1] = datar ;
        }

        for (i = 0 ; i < 4 ; ++i)
        {
                for (j = 0 ; j < 256 ; j += 2)
                {
                  Blowfish_encipher (&datal, &datar) ;
                  SBoxes [i][j] = datal ;
                  SBoxes [i][j + 1] = datar ;
                }
        }
}

        // get output length, which must be even MOD 8
DWORD CBlowFish::GetOutputLength (DWORD lInputLong)
{
        DWORD   lVal ;

        lVal = lInputLong % 8 ;       // find out if uneven number of bytes at
the end
        if (lVal != 0)
                return lInputLong + 8 - lVal ;
        else
                return lInputLong ;
}

        // Encode pIntput into pOutput.  Input length in lSize.  Returned value
        // is length of output which will be even MOD 8 bytes.  Input
buffer and
        // output buffer can be the same, but be sure buffer length is even MOD
8.
```

```
DWORD CBlowFish::Encode (BYTE * pInput, BYTE * pOutput, DWORD lSize)
{
	DWORD	lCount, lOutSize, lGoodBytes ;
	BYTE	*pi, *po ;
	int		i, j ;
	int		SameDest = (pInput == pOutput ? 1 : 0) ;

	lOutSize = GetOutputLength (lSize) ;
	for (lCount = 0 ; lCount < lOutSize ; lCount += 8)
	{
		if (SameDest)  // if encoded data is being written into input
buffer
		{
			if (lCount < lSize - 7)	// if not dealing with
uneven
bytes at end
			{
				Blowfish_encipher ((DWORD *) pInput,
					(DWORD *) (pInput + 4)) ;
			}
			else		// pad end of data with null bytes to
complete encryption
			{
				po = pInput + lSize ;  // point at byte
past the
end of actual data
				j = (int) (lOutSize - lSize) ;	// number of
bytes to set to null
				for (i = 0 ; i < j ; i++)
					*po++ = 0 ;
				Blowfish_encipher ((DWORD *) pInput,
					(DWORD *) (pInput + 4)) ;
			}
			pInput += 8 ;
		}
		else			// output buffer not equal to input
buffer, so must copy
		{		 // input to output buffer prior to encrypting
			if (lCount < lSize - 7)	// if not dealing with
uneven
bytes at end
			{
				pi = pInput ;
				po = pOutput ;
				for (i = 0 ; i < 8 ; i++)
// copy bytes to output
					*po++ = *pi++ ;
				Blowfish_encipher ((DWORD *) pOutput, // now
encrypt them
					(DWORD *) (pOutput + 4)) ;
			}
			else		// pad end of data with null bytes to
complete encryption
			{
				lGoodBytes = lSize - lCount ; // number of
remaining data bytes
				po = pOutput ;
				for (i = 0 ; i < (int) lGoodBytes ; i++)
					*po++ = *pInput++ ;
				for (j = i ; j < 8 ; j++)
					*po++ = 0 ;
				Blowfish_encipher ((DWORD *) pOutput,
					(DWORD *) (pOutput + 4)) ;
			}
			pInput += 8 ;
			pOutput += 8 ;
		}
	}
	return lOutSize ;
 }
```

```
DWORD CBlowFish::Encode (BYTE * pInput, BYTE * pOutput, DWORD lSize)
{
	DWORD	lCount, lOutSize, lGoodBytes ;
	BYTE	*pi, *po ;
	int		i, j ;
	int		SameDest = (pInput == pOutput ? 1 : 0) ;

	lOutSize = GetOutputLength (lSize) ;
	for (lCount = 0 ; lCount < lOutSize ; lCount += 8)
	{
		if (SameDest)  // if encoded data is being written into input
buffer
		{
			if (lCount < lSize - 7)	// if not dealing with
uneven
bytes at end
			{
				Blowfish_encipher ((DWORD *) pInput,
					(DWORD *) (pInput + 4)) ;
			}
			else		// pad end of data with null bytes to
complete encryption
			{
				po = pInput + lSize ;  // point at byte
past the
end of actual data
				j = (int) (lOutSize - lSize) ;	// number of
bytes to set to null
				for (i = 0 ; i < j ; i++)
					*po++ = 0 ;
				Blowfish_encipher ((DWORD *) pInput,
					(DWORD *) (pInput + 4)) ;
			}
			pInput += 8 ;
		}
		else			// output buffer not equal to input
buffer, so must copy
		{		 // input to output buffer prior to encrypting
			if (lCount < lSize - 7)	// if not dealing with
uneven
bytes at end
			{
				pi = pInput ;
				po = pOutput ;
				for (i = 0 ; i < 8 ; i++)
// copy bytes to output
					*po++ = *pi++ ;
				Blowfish_encipher ((DWORD *) pOutput, // now
encrypt them
					(DWORD *) (pOutput + 4)) ;
			}
			else		// pad end of data with null bytes to
complete encryption
			{
				lGoodBytes = lSize - lCount ; // number of
remaining data bytes
				po = pOutput ;
				for (i = 0 ; i < (int) lGoodBytes ; i++)
					*po++ = *pInput++ ;
				for (j = i ; j < 8 ; j++)
					*po++ = 0 ;
				Blowfish_encipher ((DWORD *) pOutput,
					(DWORD *) (pOutput + 4)) ;
			}
			pInput += 8 ;
			pOutput += 8 ;
		}
	}
	return lOutSize ;
 }
```

```
// Decode pIntput into pOutput.  Input length in lSize.  Input
buffer and
        // output buffer can be the same, but be sure buffer length is even MOD
8.
void CBlowFish::Decode (BYTE * pInput, BYTE * pOutput, DWORD lSize)
{
        DWORD   lCount ;
        BYTE    *pi, *po ;
        int             i ;
        int             SameDest = (pInput == pOutput ? 1 : 0) ;

        for (lCount = 0 ; lCount < lSize ; lCount += 8)
        {
                if (SameDest)  // if encoded data is being written into input
buffer
                {
                        Blowfish_decipher ((DWORD *) pInput,
                                (DWORD *) (pInput + 4)) ;
                        pInput += 8 ;
                }
                else                    // output buffer not equal to input
buffer
                {                        // so copy input to output before decoding
                        pi = pInput ;
                        po = pOutput ;
                        for (i = 0 ; i < 8 ; i++)
                                *po++ = *pi++ ;
                        Blowfish_decipher ((DWORD *) pOutput,
                                (DWORD *) (pOutput + 4)) ;
                        pInput += 8 ;
                        pOutput += 8 ;
                }
        }
}
```

5. Keamanan Blowfish

Tidak ada kelemahan yang berarti dari algoritma Blowfish yang dapat ditemukan sampai saat ini, kecuali adanya *weak key*, dimana dua entri dari S-box mempunyai nilai yang sama. Tidak ada cara untuk mencek *weak key* sebelum melakukan *key expansion*. Bila dikhawatirkan hal ini dapat mengurangi keamanannya maka dapat dibuat rutin untuk mengecek entri S-box, walaupun hal ini tidak perlu.

Sampai saat ini tidak ada *cryptanalysis* yang berhasil tehadap Blowfish, untuk amannya jangan menggunakan Blowfish dengan kurang dari 16 putaran (*round*). Karena Vincent Rijmen dalam tesisnya, memperkenalkan suatu teknik yang disebut *second-order differencial attack* yang dapat memecahkan kode 4 putaran blowfish, namun tidak untuk putaran yang lebih banyak lagi.

6. Kesimpulan

Kesimpulan yang dapat dimbil dari studi dan implementasi *Blowfish Encryption Algoritm* dalam bahasa pemrograman C++ ini antara lain:

1. Banyak faktor yang harus diperhatikan ketika ingin mengimplementasikan suatu metode enkripsi pada produk software berbasis keamanan. Kecepatan enkripsi, kesederhanaan, kekompakan, keamanan dan kekuatan kode, kemudahan dalam pengimplementasian, dan lain sebagainya. Salah satu metode atau algoritma yang dapat diandalkan untuk memenuhi segala persyaratan tersebut antara lain adalah Blowfish
2. *Round*(putaran) enkripsi memegang peranan penting dalam keamanan algoritma Blowfish. Semakin banyak dilakukan perulangan maka semakin mustahil pula kode keamanan Blowfish dapat dipecahkan(*crack*) oleh para *cryptanalysis* dan sekaligus para *cracker*.

3. Penulis menganjurkan dalam pengimplementasian algoritma Blowfish ini sedikitnya 16 *round* dianjurkan supaya kekuatan kode yang dihasilkan benar-benar terjamin keamanannya.

7. Daftar Pustaka

[1]. Schneier, Bruce. Applied Cryptography 2nd Edition. John Wiley & Sons, Inc. 1999

[2]. ***http://www.schneier.com/blowfish.html***. Tanggal akses : 27 Desember 2006

[3]. ***http://www.schneier.com/paper-blowfish- fse.html***. Tanggal akses : 27 Desember 2006

[4]. ***http://www.bimacipta.com/blowfish.htm***. Tanggal akses : 27 Desember 2006

[5]. Suryadharma, Yosef. Studi Algoritma Chiper Blok Kunci Simetri Blowfish Chiper. Program Studi Teknik Informatika Institut Teknologi Bandung. 2005

[6]. Munir, Rinaldi. Diktat Kuliah IF5054 Kriptografi. Program Studi Teknik Informatika Institut Teknologi Bandung. 2006

[7]. ***http://www.firstbackup.com/blowfish.htm***. Tanggal akses : 29 Desember 2006

[8]. ***http://www.finecrypt.net***. Tanggal Akses : 29 Desember 2006

[9]. ***http://en.wikipedia.org/wiki/Blowfish_(cipher)***. Tanggal akses : 29 Desembar 2006
