Camellia を利用した暗号ソフト

（C）宇山　靖政
2012.04.08

Camellia は NTT と三菱電機が共同開発した優れた暗号ソフトです。

Camellia 紹介の HP では、
Camellia（カメリア）は、世界のトップクラスの暗号研究者を抱える NTT と三菱電機が共同で 2000 年に開発した共通鍵ブロック暗号です。技術的に高い安全性を有するのは当然のこと、効率性と実用性にも優れており、さまざまなプラットフォーム上でのソフトウェアにより高速に実装することができます。ハードウェア実装においても、高速実装はもとよりコンパクトかつ低消費電力型の実装が可能です。
これらの技術的優位性は、例えば欧州連合推奨暗号選定プロジェクト NESSIE において「米国政府標準暗号 AES と多くの点で同等の安全性と性能を有している」と評価されるなど、国際的にも認められています。現在では、AES と同等の安全性・処理性能を有しているほぼ唯一の暗号として国際的にも認知されつつあり、多くの国際的な標準暗号・推奨暗号に選定されています。
とりわけ、日本国産暗号としては、初めてインターネット標準暗号（IETF Standard Track RFC）として承認されました。
また、オープンソースの提供も積極的に実施しており、現在では国産暗号としては初めて OpenSSL, Firefox, Linux, FreeBSD をはじめとする国際的にも主要なオープンソースソフトウェアに搭載されています。さらには欧米企業等との連携を促進するため、NTT は MIT ケルベロスコンソーシアムへ加盟しました。

NTT および三菱電機は、国内外の様々な製品やサービスに広く利用できる環境づくりに貢献し、Camellia の普及・促進により、低コストで安全な高度情報流通社会の実現に向けて主導的役割を果たすため、Camellia 基本特許の無償化を実施しております。なお、従来は Camellia を搭載する製品を開発・事業化する企業・法人様を主な対象に基本特許無償許諾契約に基づく特許無償化を行ってきました。しかしながら、今般の NTT と三菱電機の合意に基づき、今後は、Camellia の利用者におかれては、基本特許無償許諾契約を締結せずとも Camellia 基本特許を無償にてご利用いただけることといたしました。詳しくは知的財産権情報のページをご覧ください。

として、紹介されています。

ソースコードを利用させていただきます。ありがとうございます。

なお、利用に当っては、現在の国際情勢を考慮して次のような告知が出されています。

2009 年 4 月 8 日

【重要】

外国為替及び外国貿易法に基づく規制強化に伴い、近々北朝鮮に対する輸出が全面的に禁止される可能性があります。

今後の政府発表に十分ご注意ください。

つきましては、弊社暗号技術を搭載したセキュリティ製品の北朝鮮に対する輸出、ならびに弊社暗号技術のオープンソースの北朝鮮での利用を全面的に自粛していただきますよう合わせてお願い申し上げます。

【重要告知】

1. NTT 製の Camellia オープンソースは、Camellia の仕様および 32 ビット CPU 用高速化手法に基づいて正しく暗号化・復号を行うようにプログラミングされていますが、暗号利用モードや鍵管理の仕組み、実装攻撃に対する対策技術などは組み込まれておりません。また、本オープンソースを利用して生じたいかなる損害についても、NTT がその責を負うことは一切ありませんので、ご了承のうえ、お使いください。
2. Camellia 技術およびそのオープンソースはいずれも「貿易関係貿易外取引等に関する省令」等により許可を要しない役務取引等に認められた公知情報に該当しますが、これらが組み込まれた暗号装置は「外国為替および外国貿易法」が定める規制貨物（輸出令別表第 1 の 9 の項(7)に掲げる貨物）に該当します。これらの暗号装置を、日本国外へ持ち出す場合、または国内外の非居住者に提供する場合には、同法に基づく輸出許可等必要な手続きをお取りください。 詳しくはこちらをご覧ください。

比較しやすいように、3種類のソースコードを掲載いたします。
1． 本来の、Camellia のソースコード
2． CmlEC のソースコード
3． CmlDC のソースコード
このうち、2，3についての二次著作権の主張はいたしません。ご自由に変更して下さい。ただし、自作のソフトの中にソースコードを組み込む場合や、鍵の長さが56ビットを超える場合には、貿易管理令についてもご確認下さい。

1． 本来の、Camellia のソースコード

camellia_ref.c

```
/**************************************************
 *                                                *
 *      Camellia Block Encryption Algorithm       *
 *        in ANSI-C Language : Camellia.c   *
 *                                                *
 *        Version M1.02 September 24 2001   *
 *  Copyright Mitsubishi Electric Corp 2000-2001  *
 *                                                *
 **************************************************/
```

```
typedef unsigned char Byte;
typedef unsigned long Word;

void Camellia_Ekeygen( const int, const Byte *, Byte * );
void Camellia_Encrypt( const int, const Byte *, const Byte *, Byte * );
void Camellia_Decrypt( const int, const Byte *, const Byte *, Byte * );
void Camellia_Feistel( const Byte *, const Byte *, Byte * );
void Camellia_FLlayer( Byte *, const Byte *, const Byte * );

void ByteWord( const Byte *, Word * );
void WordByte( const Word *, Byte * );
void XorBlock( const Byte *, const Byte *, Byte * );
void SwapHalf( Byte * );
void RotBlock( const Word *, const int, Word * );

const Byte SIGMA[48] = {
0xa0,0x9e,0x66,0x7f,0x3b,0xcc,0x90,0x8b,
0xb6,0x7a,0xe8,0x58,0x4c,0xaa,0x73,0xb2,
0xc6,0xef,0x37,0x2f,0xe9,0x4f,0x82,0xbe,
0x54,0xff,0x53,0xa5,0xf1,0xd3,0x6f,0x1c,
0x10,0xe5,0x27,0xfa,0xde,0x68,0x2d,0x1d,
0xb0,0x56,0x88,0xc2,0xb3,0xe6,0xc1,0xfd};

const int KSFT1[26] = {
0,64,0,64,15,79,15,79,30,94,45,109,45,124,60,124,77,13,
94,30,94,30,111,47,111,47 };
const int KIDX1[26] = {
0,0,4,4,0,0,4,4,4,4,0,0,4,0,4,4,0,0,0,0,4,4,0,0,4,4 };
const int KSFT2[34] = {
0,64,0,64,15,79,15,79,30,94,30,94,45,109,45,109,60,124,
60,124,60,124,77,13,77,13,94,30,94,30,111,47,111,47 };
const int KIDX2[34] = {
0,0,12,12,8,8,4,4,8,8,12,12,0,0,4,4,0,0,8,8,12,12,
0,0,4,4,8,8,4,4,0,0,12,12 };

const Byte SBOX[256] = {
112,130, 44,236,179, 39,192,229,228,133, 87, 53,234, 12,174, 65,
 35,239,107,147, 69, 25,165, 33,237, 14, 79, 78, 29,101,146,189,
134,184,175,143,124,235, 31,206, 62, 48,220, 95, 94,197, 11, 26,
166,225, 57,202,213, 71, 93, 61,217,  1, 90,214, 81, 86,108, 77,
139, 13,154,102,251,204,176, 45,116, 18, 43, 32,240,177,132,153,
223, 76,203,194, 52,126,118,  5,109,183,169, 49,209, 23,  4,215,
 20, 88, 58, 97,222, 27, 17, 28, 50, 15,156, 22, 83, 24,242, 34,
254, 68,207,178,195,181,122,145, 36,  8,232,168, 96,252,105, 80,
170,208,160,125,161,137, 98,151, 84, 91, 30,149,224,255,100,210,
 16,196,  0, 72,163,247,117,219,138,  3,230,218,  9, 63,221,148,
135, 92,131,  2,205, 74,144, 51,115,103,246,243,157,127,191,226,
 82,155,216, 38,200, 55,198, 59,129,150,111, 75, 19,190, 99, 46,
233,121,167,140,159,110,188,142, 41,245,249,182, 47,253,180, 89,
120,152,  6,106,231, 70,113,186,212, 37,171, 66,136,162,141,250,
114,  7,185, 85,248,238,172, 10, 54, 73, 42,104, 60, 56,241,164,
```

```
 64, 40,211,123,187,201, 67,193, 21,227,173,244,119,199,128,158};

#define SBOX1(n) SBOX[(n)]
#define SBOX2(n) (Byte)((SBOX[(n)]>>7^SBOX[(n)]<<1)&0xff)
#define SBOX3(n) (Byte)((SBOX[(n)]>>1^SBOX[(n)]<<7)&0xff)
#define SBOX4(n) SBOX[((n)<<1^(n)>>7)&0xff]

void Camellia_Ekeygen( const int n, const Byte *k, Byte *e )
{
        Byte t[64];
        Word u[20];
        int  i;

        if( n == 128 ){
                for( i=0 ; i<16; i++ ) t[i] = k[i];
                for( i=16; i<32; i++ ) t[i] = 0;
        }
        else if( n == 192 ){
                for( i=0 ; i<24; i++ ) t[i] = k[i];
                for( i=24; i<32; i++ ) t[i] = k[i-8]^0xff;
        }
        else if( n == 256 ){
                for( i=0 ; i<32; i++ ) t[i] = k[i];
        }

        XorBlock( t+0, t+16, t+32 );

        Camellia_Feistel( t+32, SIGMA+0, t+40 );
        Camellia_Feistel( t+40, SIGMA+8, t+32 );

        XorBlock( t+32, t+0, t+32 );

        Camellia_Feistel( t+32, SIGMA+16, t+40 );
        Camellia_Feistel( t+40, SIGMA+24, t+32 );

        ByteWord( t+0,  u+0 );
        ByteWord( t+32, u+4 );

        if( n == 128 ){
                for( i=0; i<26; i+=2 ){
                        RotBlock( u+KIDX1[i+0], KSFT1[i+0], u+16 );
                        RotBlock( u+KIDX1[i+1], KSFT1[i+1], u+18 );
                        WordByte( u+16, e+i*8 );
                }
        }
        else{
                XorBlock( t+32, t+16, t+48 );

                Camellia_Feistel( t+48, SIGMA+32, t+56 );
                Camellia_Feistel( t+56, SIGMA+40, t+48 );

                ByteWord( t+16, u+8  );
```

```
                ByteWord( t+48, u+12 );

                for( i=0; i<34; i+=2 ){
                        RotBlock( u+KIDX2[i+0], KSFT2[i+0], u+16 );
                        RotBlock( u+KIDX2[i+1], KSFT2[i+1], u+18 );
                        WordByte( u+16, e+(i<<3) );
                }
        }
}

void Camellia_Encrypt( const int n, const Byte *p, const Byte *e, Byte *c )
{
        int i;

        XorBlock( p, e+0, c );

        for( i=0; i<3; i++ ){
                Camellia_Feistel( c+0, e+16+(i<<4), c+8 );
                Camellia_Feistel( c+8, e+24+(i<<4), c+0 );
        }

        Camellia_FLlayer( c, e+64, e+72 );

        for( i=0; i<3; i++ ){
                Camellia_Feistel( c+0, e+80+(i<<4), c+8 );
                Camellia_Feistel( c+8, e+88+(i<<4), c+0 );
        }

        Camellia_FLlayer( c, e+128, e+136 );

        for( i=0; i<3; i++ ){
                Camellia_Feistel( c+0, e+144+(i<<4), c+8 );
                Camellia_Feistel( c+8, e+152+(i<<4), c+0 );
        }

        if( n == 128 ){
                SwapHalf( c );
                XorBlock( c, e+192, c );
        }
        else{
                Camellia_FLlayer( c, e+192, e+200 );

                for( i=0; i<3; i++ ){
                        Camellia_Feistel( c+0, e+208+(i<<4), c+8 );
                        Camellia_Feistel( c+8, e+216+(i<<4), c+0 );
                }

                SwapHalf( c );
                XorBlock( c, e+256, c );
        }
}
```

```
void Camellia_Decrypt( const int n, const Byte *c, const Byte *e, Byte *p )
{
	int i;

	if( n == 128 ){
		XorBlock( c, e+192, p );
	}
	else{
		XorBlock( c, e+256, p );

		for( i=2; i>=0; i-- ){
			Camellia_Feistel( p+0, e+216+(i<<4), p+8 );
			Camellia_Feistel( p+8, e+208+(i<<4), p+0 );
		}

		Camellia_FLlayer( p, e+200, e+192 );
	}

	for( i=2; i>=0; i-- ){
		Camellia_Feistel( p+0, e+152+(i<<4), p+8 );
		Camellia_Feistel( p+8, e+144+(i<<4), p+0 );
	}

	Camellia_FLlayer( p, e+136, e+128 );

	for( i=2; i>=0; i-- ){
		Camellia_Feistel( p+0, e+88+(i<<4), p+8 );
		Camellia_Feistel( p+8, e+80+(i<<4), p+0 );
	}

	Camellia_FLlayer( p, e+72, e+64 );

	for( i=2; i>=0; i-- ){
		Camellia_Feistel( p+0, e+24+(i<<4), p+8 );
		Camellia_Feistel( p+8, e+16+(i<<4), p+0 );
	}

	SwapHalf( p );
	XorBlock( p, e+0, p );
}

void Camellia_Feistel( const Byte *x, const Byte *k, Byte *y )
{
	Byte t[8];

	t[0] = SBOX1(x[0]^k[0]);
	t[1] = SBOX2(x[1]^k[1]);
	t[2] = SBOX3(x[2]^k[2]);
	t[3] = SBOX4(x[3]^k[3]);
	t[4] = SBOX2(x[4]^k[4]);
	t[5] = SBOX3(x[5]^k[5]);
	t[6] = SBOX4(x[6]^k[6]);
```

```
        t[7] = SBOX1(x[7]^k[7]);

        y[0] ^= t[0]^t[2]^t[3]^t[5]^t[6]^t[7];
        y[1] ^= t[0]^t[1]^t[3]^t[4]^t[6]^t[7];
        y[2] ^= t[0]^t[1]^t[2]^t[4]^t[5]^t[7];
        y[3] ^= t[1]^t[2]^t[3]^t[4]^t[5]^t[6];
        y[4] ^= t[0]^t[1]^t[5]^t[6]^t[7];
        y[5] ^= t[1]^t[2]^t[4]^t[6]^t[7];
        y[6] ^= t[2]^t[3]^t[4]^t[5]^t[7];
        y[7] ^= t[0]^t[3]^t[4]^t[5]^t[6];
}

void Camellia_FLlayer( Byte *x, const Byte *kl, const Byte *kr )
{
        Word t[4],u[4],v[4];

        ByteWord( x, t );
        ByteWord( kl, u );
        ByteWord( kr, v );

        t[1] ^= (t[0]&u[0])<<1^(t[0]&u[0])>>31;
        t[0] ^= t[1]|u[1];
        t[2] ^= t[3]|v[1];
        t[3] ^= (t[2]&v[0])<<1^(t[2]&v[0])>>31;

        WordByte( t, x );
}

void ByteWord( const Byte *x, Word *y )
{
        int i;
        for( i=0; i<4; i++ ){
                y[i] = ((Word)x[(i<<2)+0]<<24) + ((Word)x[(i<<2)+1]<<16)
                     + ((Word)x[(i<<2)+2]<<8 ) + ((Word)x[(i<<2)+3]<<0 );
        }
}

void WordByte( const Word *x, Byte *y )
{
        int i;
        for( i=0; i<4; i++ ){
                y[(i<<2)+0] = (Byte)(x[i]>>24&0xff);
                y[(i<<2)+1] = (Byte)(x[i]>>16&0xff);
                y[(i<<2)+2] = (Byte)(x[i]>> 8&0xff);
                y[(i<<2)+3] = (Byte)(x[i]>> 0&0xff);
        }
}

void RotBlock( const Word *x, const int n, Word *y )
{
        int r;
        if( r = (n & 31) ){
```

```
                y[0] = x[((n>>5)+0)&3]<<r^x[((n>>5)+1)&3]>>(32-r);
                y[1] = x[((n>>5)+1)&3]<<r^x[((n>>5)+2)&3]>>(32-r);
        }
        else{
                y[0] = x[((n>>5)+0)&3];
                y[1] = x[((n>>5)+1)&3];
        }
}

void SwapHalf( Byte *x )
{
        Byte t;
        int  i;
        for( i=0; i<8; i++ ){
                t = x[i];
                x[i] = x[8+i];
                x[8+i] = t;
        }
}

void XorBlock( const Byte *x, const Byte *y, Byte *z )
{
        int i;
        for( i=0; i<16; i++ ) z[i] = x[i] ^ y[i];
}
```

以上のソースコードは、古いものです。もっと新しいソースコードが公開されています。

2. CmlEC のソースコード

Camellia.cpp

```
/***************************************************
 *                                                 *
 *      Camellia Block Encryption Algorithm        *
 *        in ANSI-C Language : Camellia.c   *
 *                                                 *
 *        Version M1.02 September 24 2001   *
 *  Copyright Mitsubishi Electric Corp 2000-2001  *
 *                                                 *
 ***************************************************/
#include "stdafx.h"

typedef unsigned char Byte;
typedef unsigned long Word;

void Camellia_Ekeygen( const int, const Byte *, Byte * );

void Camellia_Encrypt( const int, const Byte *, const Byte *, Byte * );
```

```
void Camellia_Decrypt( const int, const Byte *, const Byte *, Byte * );

void Camellia_Feistel( const Byte *, const Byte *, Byte * );
void Camellia_FLlayer( Byte *, const Byte *, const Byte * );

void ByteWord( const Byte *, Word * );
void WordByte( const Word *, Byte * );
void XorBlock( const Byte *, const Byte *, Byte * );
void SwapHalf( Byte * );
void RotBlock( const Word *, const int, Word * );

const Byte SIGMA[48] = {
0xa0,0x9e,0x66,0x7f,0x3b,0xcc,0x90,0x8b,
0xb6,0x7a,0xe8,0x58,0x4c,0xaa,0x73,0xb2,
0xc6,0xef,0x37,0x2f,0xe9,0x4f,0x82,0xbe,
0x54,0xff,0x53,0xa5,0xf1,0xd3,0x6f,0x1c,
0x10,0xe5,0x27,0xfa,0xde,0x68,0x2d,0x1d,
0xb0,0x56,0x88,0xc2,0xb3,0xe6,0xc1,0xfd};

const int KSFT1[26] = {
0,64,0,64,15,79,15,79,30,94,45,109,45,124,60,124,77,13,
94,30,94,30,111,47,111,47 };
const int KIDX1[26] = {
0,0,4,4,0,0,4,4,4,4,0,0,4,0,4,4,0,0,0,0,4,4,0,0,4,4 };
const int KSFT2[34] = {
0,64,0,64,15,79,15,79,30,94,30,94,45,109,45,109,60,124,
60,124,60,124,77,13,77,13,94,30,94,30,111,47,111,47 };
const int KIDX2[34] = {
0,0,12,12,8,8,4,4,8,8,12,12,0,0,4,4,0,0,8,8,12,12,
0,0,4,4,8,8,4,4,0,0,12,12 };

const Byte SBOX[256] = {
112,130, 44,236,179, 39,192,229,228,133, 87, 53,234, 12,174, 65,
 35,239,107,147, 69, 25,165, 33,237, 14, 79, 78, 29,101,146,189,
134,184,175,143,124,235, 31,206, 62, 48,220, 95, 94,197, 11, 26,
166,225, 57,202,213, 71, 93, 61,217,  1, 90,214, 81, 86,108, 77,
139, 13,154,102,251,204,176, 45,116, 18, 43, 32,240,177,132,153,
223, 76,203,194, 52,126,118,  5,109,183,169, 49,209, 23,  4,215,
 20, 88, 58, 97,222, 27, 17, 28, 50, 15,156, 22, 83, 24,242, 34,
254, 68,207,178,195,181,122,145, 36,  8,232,168, 96,252,105, 80,
170,208,160,125,161,137, 98,151, 84, 91, 30,149,224,255,100,210,
 16,196,  0, 72,163,247,117,219,138,  3,230,218,  9, 63,221,148,
135, 92,131,  2,205, 74,144, 51,115,103,246,243,157,127,191,226,
 82,155,216, 38,200, 55,198, 59,129,150,111, 75, 19,190, 99, 46,
233,121,167,140,159,110,188,142, 41,245,249,182, 47,253,180, 89,
120,152,  6,106,231, 70,113,186,212, 37,171, 66,136,162,141,250,
114,  7,185, 85,248,238,172, 10, 54, 73, 42,104, 60, 56,241,164,
 64, 40,211,123,187,201, 67,193, 21,227,173,244,119,199,128,158};

#define SBOX1(n) SBOX[(n)]
#define SBOX2(n) (Byte)((SBOX[(n)]>>7^SBOX[(n)]<<1)&0xff)
#define SBOX3(n) (Byte)((SBOX[(n)]>>1^SBOX[(n)]<<7)&0xff)
```

```
#define SBOX4(n) SBOX[((n)<<1^(n)>>7)&0xff]

void Camellia_Ekeygen( const int n, const Byte *k, Byte *e )
{
        Byte t[64];
        Word u[20];
        int  i;

        if( n == 128 ){
                for( i=0 ; i<16; i++ ) t[i] = k[i];
                for( i=16; i<32; i++ ) t[i] = 0;
        }
        else if( n == 192 ){
                for( i=0 ; i<24; i++ ) t[i] = k[i];
                for( i=24; i<32; i++ ) t[i] = k[i-8]^0xff;
        }
        else if( n == 256 ){
                for( i=0 ; i<32; i++ ) t[i] = k[i];
        }

        XorBlock( t+0, t+16, t+32 );

        Camellia_Feistel( t+32, SIGMA+0, t+40 );
        Camellia_Feistel( t+40, SIGMA+8, t+32 );

        XorBlock( t+32, t+0, t+32 );

        Camellia_Feistel( t+32, SIGMA+16, t+40 );
        Camellia_Feistel( t+40, SIGMA+24, t+32 );

        ByteWord( t+0,  u+0 );
        ByteWord( t+32, u+4 );

        if( n == 128 ){
                for( i=0; i<26; i+=2 ){
                        RotBlock( u+KIDX1[i+0], KSFT1[i+0], u+16 );
                        RotBlock( u+KIDX1[i+1], KSFT1[i+1], u+18 );
                        WordByte( u+16, e+i*8 );
                }
        }
        else{
                XorBlock( t+32, t+16, t+48 );

                Camellia_Feistel( t+48, SIGMA+32, t+56 );
                Camellia_Feistel( t+56, SIGMA+40, t+48 );

                ByteWord( t+16, u+8  );
                ByteWord( t+48, u+12 );

                for( i=0; i<34; i+=2 ){
                        RotBlock( u+KIDX2[i+0], KSFT2[i+0], u+16 );
                        RotBlock( u+KIDX2[i+1], KSFT2[i+1], u+18 );
```

```
                        WordByte( u+16, e+(i<<3) );
                }
        }
}

void Camellia_Encrypt( const int n, const Byte *p, const Byte *e, Byte *c )
{//n=鍵長   p=平文   e=拡張鍵   c=暗号文
        int i;

        XorBlock( p, e+0, c );

        for( i=0; i<3; i++ ){
                Camellia_Feistel( c+0, e+16+(i<<4), c+8 );
                Camellia_Feistel( c+8, e+24+(i<<4), c+0 );
        }

        Camellia_FLlayer( c, e+64, e+72 );

        for( i=0; i<3; i++ ){
                Camellia_Feistel( c+0, e+80+(i<<4), c+8 );
                Camellia_Feistel( c+8, e+88+(i<<4), c+0 );
        }

        Camellia_FLlayer( c, e+128, e+136 );

        for( i=0; i<3; i++ ){
                Camellia_Feistel( c+0, e+144+(i<<4), c+8 );
                Camellia_Feistel( c+8, e+152+(i<<4), c+0 );
        }

        if( n == 128 ){
                SwapHalf( c );
                XorBlock( c, e+192, c );
        }
        else{
                Camellia_FLlayer( c, e+192, e+200 );

                for( i=0; i<3; i++ ){
                        Camellia_Feistel( c+0, e+208+(i<<4), c+8 );
                        Camellia_Feistel( c+8, e+216+(i<<4), c+0 );
                }

                SwapHalf( c );
                XorBlock( c, e+256, c );
        }
}

void Camellia_Decrypt( const int n, const Byte *c, const Byte *e, Byte *p )
{
        int i;

        if( n == 128 ){
```

```
                XorBlock( c, e+192, p );
        }
        else{
                XorBlock( c, e+256, p );

                for( i=2; i>=0; i-- ){
                        Camellia_Feistel( p+0, e+216+(i<<4), p+8 );
                        Camellia_Feistel( p+8, e+208+(i<<4), p+0 );
                }

                Camellia_FLlayer( p, e+200, e+192 );
        }

        for( i=2; i>=0; i-- ){
                Camellia_Feistel( p+0, e+152+(i<<4), p+8 );
                Camellia_Feistel( p+8, e+144+(i<<4), p+0 );
        }

        Camellia_FLlayer( p, e+136, e+128 );

        for( i=2; i>=0; i-- ){
                Camellia_Feistel( p+0, e+88+(i<<4), p+8 );
                Camellia_Feistel( p+8, e+80+(i<<4), p+0 );
        }

        Camellia_FLlayer( p, e+72, e+64 );

        for( i=2; i>=0; i-- ){
                Camellia_Feistel( p+0, e+24+(i<<4), p+8 );
                Camellia_Feistel( p+8, e+16+(i<<4), p+0 );
        }

        SwapHalf( p );
        XorBlock( p, e+0, p );
}

void Camellia_Feistel( const Byte *x, const Byte *k, Byte *y )
{
        Byte t[8];

        t[0] = SBOX1(x[0]^k[0]);
        t[1] = SBOX2(x[1]^k[1]);
        t[2] = SBOX3(x[2]^k[2]);
        t[3] = SBOX4(x[3]^k[3]);
        t[4] = SBOX2(x[4]^k[4]);
        t[5] = SBOX3(x[5]^k[5]);
        t[6] = SBOX4(x[6]^k[6]);
        t[7] = SBOX1(x[7]^k[7]);

        y[0] ^= t[0]^t[2]^t[3]^t[5]^t[6]^t[7];
        y[1] ^= t[0]^t[1]^t[3]^t[4]^t[6]^t[7];
        y[2] ^= t[0]^t[1]^t[2]^t[4]^t[5]^t[7];
```

```
        y[3] ^= t[1]^t[2]^t[3]^t[4]^t[5]^t[6];
        y[4] ^= t[0]^t[1]^t[5]^t[6]^t[7];
        y[5] ^= t[1]^t[2]^t[4]^t[6]^t[7];
        y[6] ^= t[2]^t[3]^t[4]^t[5]^t[7];
        y[7] ^= t[0]^t[3]^t[4]^t[5]^t[6];
}

void Camellia_FLlayer( Byte *x, const Byte *kl, const Byte *kr )
{
        Word t[4],u[4],v[4];

        ByteWord( x, t );
        ByteWord( kl, u );
        ByteWord( kr, v );

        t[1] ^= (t[0]&u[0])<<1^(t[0]&u[0])>>31;
        t[0] ^= t[1]|u[1];
        t[2] ^= t[3]|v[1];
        t[3] ^= (t[2]&v[0])<<1^(t[2]&v[0])>>31;

        WordByte( t, x );
}

void ByteWord( const Byte *x, Word *y )
{
        int i;
        for( i=0; i<4; i++ ){
                y[i] = ((Word)x[(i<<2)+0]<<24) + ((Word)x[(i<<2)+1]<<16)
                     + ((Word)x[(i<<2)+2]<<8 ) + ((Word)x[(i<<2)+3]<<0 );
        }
}

void WordByte( const Word *x, Byte *y )
{
        int i;
        for( i=0; i<4; i++ ){
                y[(i<<2)+0] = (Byte)(x[i]>>24&0xff);
                y[(i<<2)+1] = (Byte)(x[i]>>16&0xff);
                y[(i<<2)+2] = (Byte)(x[i]>> 8&0xff);
                y[(i<<2)+3] = (Byte)(x[i]>> 0&0xff);
        }
}

void RotBlock( const Word *x, const int n, Word *y )
{
        int r;
        if( r = (n & 31) ){
                y[0] = x[((n>>5)+0)&3]<<r^x[((n>>5)+1)&3]>>(32-r);
                y[1] = x[((n>>5)+1)&3]<<r^x[((n>>5)+2)&3]>>(32-r);
        }
        else{
                y[0] = x[((n>>5)+0)&3];
```

```
                y[1] = x[((n>>5)+1)&3];
        }
}

void SwapHalf( Byte *x )
{
        Byte t;
        int  i;
        for( i=0; i<8; i++ ){
                t = x[i];
                x[i] = x[8+i];
                x[8+i] = t;
        }
}

void XorBlock( const Byte *x, const Byte *y, Byte *z )
{
        int i;
        for( i=0; i<16; i++ ) z[i] = x[i] ^ y[i];
}
```

ECcml.h

```
/*****************************************************
 * ECcml.h
 * Misty 暗号関数
 *****************************************************/

// 戻り値
#define NOERROR                  0                      // エラーなし
#define NOTENOUGHMEMORY  -1                     // メモリー不足
#define ACCESSERROR              -2                     // アクセスエラー
#define MATHERROR                -3                     // 数学的な誤り
#define KEYLENGTHERROR   -4                     // 鍵長不正
#define OTHERERROR               -5                     // その他のエラー

// 定数
#define MINKEYLENGTH     32                     // 最低鍵長（ビット）


typedef unsigned char uchar;
typedef unsigned short ushort;
typedef unsigned char Byte;
typedef unsigned long Word;

/*********************/
/*** RSA公開鍵暗号***/
/*********************/

// 暗号化　復号化
```

```
void Camellia_Ekeygen( const int, const Byte *, Byte * );
void Camellia_Encrypt( const int, const Byte *, const Byte *, Byte * );
```

ECcml.cpp

```
////////////////////////////////////////////////////////////////////////
//  ECcml.cpp : コンソールアプリケーション用のエントリポイントの定義
////////////////////////////////////////////////////////////////////////

#include "stdafx.h"

#include <iostream>
#include <string.h>
#include <stdlib.h>
#include "ECcml.h"
#include <stdio.h>

using namespace std;

FILE *stream;
FILE *stream1;
FILE *stream2;

extern ushort EXTKEY[4][8];

// 暗号文のHEX表示用
char toChar( int c )
{
        if( c >= 0 && c <= 9 )                  // 0～ならば
                return char( c + 0x30 ); // ASCIIに変換して返す
        else if( c >= 10 && c <= 15 )     // 10～ならば
                return char( c + 0x37 ); // A～FのASCIIを返す
        else
                return ' ';
}

// メイン
int main(                                                       // 正常終了時:0、異常時:1
                int argc,                                       // 引数の数
                char** argv )                           // 引数へのポインタ
{
        int i, len;
        int block;
        unsigned char klen[8], key[32], key2[64];
        unsigned char exkey[256];
        int numclosed;
        char j,k;
        char* bufp;
        char* bufc;
        int cnt;
```

```
        long mesLength; // 平文長（バイト）
        int c;

        printf( "Start!¥n" );

          // 引数チェック
        if( argc != 4 ){                                // 使い方の誤り
                printf( "引数の数が不正ですから異常終了します。¥n" );
                exit(1);
        }

 /* 鍵ファイルを開く*/
 if( (stream  = fopen( argv[1], "rb" )) == NULL )
    printf( "Can not open Key file.¥n" );

/* 平文を読み出すファイルを開く*/
 if( (stream1  = fopen( argv[2], "rb" )) == NULL )
    printf( "Can not open Plane Text file.¥n");

 /* 暗号文を書き込むファイルを開く*/
 if( (stream2 = fopen( argv[3], "wb" )) == NULL )
    printf( "Can not open for Encrypted file.¥n" );


        fscanf(stream,"%s",klen);
        fscanf(stream,"%s",key2);
        len = atoi((char*)klen);

        for(i=0; i<len/8; i++){
                j = *(key2+2*i);
                k = *(key2+2*i+1);
                if(j>=0x30 && j<=0x39) j = j-0x30;
                else{
                        if(j>=0x41 && j<=0x46) j = j-0x41+0x0A;
                }
                if(k>=0x30 && k<=0x39) k = k-0x30;
                else{
                        if(k>=0x41 && k<=0x46) k = k-0x41+0x0A;
                }
                key[i] = j*0x10 + k;
        }
        key[len/8] = NULL;

        Camellia_Ekeygen( len, key, exkey );

// 平文
        fseek(stream1, 0, SEEK_END);
        long filelen = ftell(stream1);
        fseek(stream1,0,0);
        int head = sizeof(long);
        mesLength = filelen + head;
```

```
        //暗号化
        if(mesLength <= 1024){
                block = mesLength/16 + ((mesLength%16)?1:0);
                bufp = (char*)new(char[block*16 + 1]);
                bufc = (char*)new(char[block*16 + 1]);
                if(bufp == NULL){
                        printf("メモリ不足です。異常終了します。¥r¥n");
                        return(-1);
                        }
                (*(long*)(bufp)) = filelen;

        // 平文
                fseek(stream1, 0, 0);
                i = head;
                do{
                        c = fgetc(stream1);
                        bufp[i]=c;
                        i=i+1;
                }while(c!=EOF);
                bufp[i-1]=NULL;

    // 暗号化実行
        for(i=0;i<block;i++){
            Camellia_Encrypt( len,  (unsigned char*)(bufp+i*16), (unsigned char*)exkey, (unsigned
char*)(bufc+i*16));
                  //n=鍵長   bufp=平文   exkey=拡張鍵   ct=暗号文
        }

        // 暗号文を書き込む
                for( cnt = 0; cnt<block*16; cnt++ ){
                        fwrite( &(bufc[cnt]),sizeof(char),1, stream2);
                }
        }
        else{
                block = mesLength/16 + ((mesLength%16)?1:0);
                bufp = (char*)new(char[1024 + 2]);
                bufc = (char*)new(char[1024 + 2]);
                if(bufp == NULL){
                        printf("メモリ不足¥r¥n");
                        return(-1);
                        }
                (*(long*)(bufp)) = filelen;

                long rBlen = block;
                int r = 0;
                do{
                // 平文

                        if(r == 0){
                                i = head;
                                fseek(stream1, r*1024, 0);
                        }
```

```
                        if(r > 0 ){
                                i = 0;
                                fseek(stream1, r*1024-4, 0);
                        }
                        do{
                                c = fgetc(stream1);
                                bufp[i]=c;
                                i=i+1;
                        }while((c!=EOF) && (i<=1024));
                        bufp[i-1]=NULL;

                        if(rBlen >= 1024/16){ block = 1024/16; }
                        if(rBlen < 1024/16){ block = rBlen;}

                // 暗号化実行
        for(i=0;i<block;i++){
            Camellia_Encrypt(len,  (unsigned char*)(bufp+i*16), (unsigned char*)exkey, (unsigned
char*)(bufc+i*16));
                    //n=鍵長   bufp=平文   exkey=拡張鍵   ct=暗号文
        }

        // 暗号文を書き込む
                        for( cnt = 0; cnt<block*16; cnt++ ){
                                fwrite( &(bufc[cnt]),sizeof(char),1, stream2);
                        }
                        r += 1;
                        rBlen -= 1024/16;
                }while(rBlen>0);
        }

        if( fclose( stream1 ) )
        printf( "Can not close Plane Text file.¥n" );
        /* ストリームを閉じる*/
        if( fclose( stream2 ) )
                printf( "Can not close Encrypted file.¥n" );
   /* 他のすべてのファイルを閉じる*/
        numclosed = _fcloseall( );

        delete[] bufp;
        delete[] bufc;

        printf( "End!¥n" );

        return 0;
}
```

Stdafx.cpp

```
////////////////////////////////////////////////////////////////////
//  stdafx.cpp : 標準インクルードファイルを含むソースファイル
//               ECMisty.pch 生成されるプリコンパイル済ヘッダー
```

```
//                stdafx.obj 生成されるプリコンパイル済タイプ情報
////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"

// TODO: STDAFX.H に含まれていて、このファイルに記述されていない
// ヘッダーファイルを追加してください。
```

以上、VC++2005 でソフトを作成するために必要なファイルです。

3.　CmlDC のソースコード

Camellia.cpp

```
/*****************************************************
 *                                                   *
 *      Camellia Block Encryption Algorithm          *
 *        in ANSI-C Language : Camellia.c   *
 *                                                   *
 *        Version M1.02 September 24 2001   *
 *  Copyright Mitsubishi Electric Corp 2000-2001  *
 *                                                   *
 *****************************************************/
#include "stdafx.h"

typedef unsigned char Byte;
typedef unsigned long Word;

void Camellia_Ekeygen( const int, const Byte *, Byte * );

void Camellia_Encrypt( const int, const Byte *, const Byte *, Byte * );
void Camellia_Decrypt( const int, const Byte *, const Byte *, Byte * );

void Camellia_Feistel( const Byte *, const Byte *, Byte * );
void Camellia_FLlayer( Byte *, const Byte *, const Byte * );

void ByteWord( const Byte *, Word * );
void WordByte( const Word *, Byte * );
void XorBlock( const Byte *, const Byte *, Byte * );
void SwapHalf( Byte * );
void RotBlock( const Word *, const int, Word * );

const Byte SIGMA[48] = {
0xa0,0x9e,0x66,0x7f,0x3b,0xcc,0x90,0x8b,
0xb6,0x7a,0xe8,0x58,0x4c,0xaa,0x73,0xb2,
0xc6,0xef,0x37,0x2f,0xe9,0x4f,0x82,0xbe,
0x54,0xff,0x53,0xa5,0xf1,0xd3,0x6f,0x1c,
```

```
0x10,0xe5,0x27,0xfa,0xde,0x68,0x2d,0x1d,
0xb0,0x56,0x88,0xc2,0xb3,0xe6,0xc1,0xfd};

const int KSFT1[26] = {
0,64,0,64,15,79,15,79,30,94,45,109,45,124,60,124,77,13,
94,30,94,30,111,47,111,47 };
const int KIDX1[26] = {
0,0,4,4,0,0,4,4,4,4,0,0,4,0,4,4,0,0,0,0,4,4,0,0,4,4 };
const int KSFT2[34] = {
0,64,0,64,15,79,15,79,30,94,30,94,45,109,45,109,60,124,
60,124,60,124,77,13,77,13,94,30,94,30,111,47,111,47 };
const int KIDX2[34] = {
0,0,12,12,8,8,4,4,8,8,12,12,0,0,4,4,0,0,8,8,12,12,
0,0,4,4,8,8,4,4,0,0,12,12 };

const Byte SBOX[256] = {
112,130, 44,236,179, 39,192,229,228,133, 87, 53,234, 12,174, 65,
 35,239,107,147, 69, 25,165, 33,237, 14, 79, 78, 29,101,146,189,
134,184,175,143,124,235, 31,206, 62, 48,220, 95, 94,197, 11, 26,
166,225, 57,202,213, 71, 93, 61,217,  1, 90,214, 81, 86,108, 77,
139, 13,154,102,251,204,176, 45,116, 18, 43, 32,240,177,132,153,
223, 76,203,194, 52,126,118,  5,109,183,169, 49,209, 23,  4,215,
 20, 88, 58, 97,222, 27, 17, 28, 50, 15,156, 22, 83, 24,242, 34,
254, 68,207,178,195,181,122,145, 36,  8,232,168, 96,252,105, 80,
170,208,160,125,161,137, 98,151, 84, 91, 30,149,224,255,100,210,
 16,196,  0, 72,163,247,117,219,138,  3,230,218,  9, 63,221,148,
135, 92,131,  2,205, 74,144, 51,115,103,246,243,157,127,191,226,
 82,155,216, 38,200, 55,198, 59,129,150,111, 75, 19,190, 99, 46,
233,121,167,140,159,110,188,142, 41,245,249,182, 47,253,180, 89,
120,152,  6,106,231, 70,113,186,212, 37,171, 66,136,162,141,250,
114,  7,185, 85,248,238,172, 10, 54, 73, 42,104, 60, 56,241,164,
 64, 40,211,123,187,201, 67,193, 21,227,173,244,119,199,128,158};

#define SBOX1(n) SBOX[(n)]
#define SBOX2(n) (Byte)((SBOX[(n)]>>7^SBOX[(n)]<<1)&0xff)
#define SBOX3(n) (Byte)((SBOX[(n)]>>1^SBOX[(n)]<<7)&0xff)
#define SBOX4(n) SBOX[((n)<<1^(n)>>7)&0xff]

void Camellia_Ekeygen( const int n, const Byte *k, Byte *e )
{
        Byte t[64];
        Word u[20];
        int  i;

        if( n == 128 ){
                for( i=0 ; i<16; i++ ) t[i] = k[i];
                for( i=16; i<32; i++ ) t[i] = 0;
        }
        else if( n == 192 ){
                for( i=0 ; i<24; i++ ) t[i] = k[i];
                for( i=24; i<32; i++ ) t[i] = k[i-8]^0xff;
        }
```

```
        else if( n == 256 ){
                for( i=0 ; i<32; i++ ) t[i] = k[i];
        }

        XorBlock( t+0, t+16, t+32 );

        Camellia_Feistel( t+32, SIGMA+0, t+40 );
        Camellia_Feistel( t+40, SIGMA+8, t+32 );

        XorBlock( t+32, t+0, t+32 );

        Camellia_Feistel( t+32, SIGMA+16, t+40 );
        Camellia_Feistel( t+40, SIGMA+24, t+32 );

        ByteWord( t+0,  u+0 );
        ByteWord( t+32, u+4 );

        if( n == 128 ){
                for( i=0; i<26; i+=2 ){
                        RotBlock( u+KIDX1[i+0], KSFT1[i+0], u+16 );
                        RotBlock( u+KIDX1[i+1], KSFT1[i+1], u+18 );
                        WordByte( u+16, e+i*8 );
                }
        }
        else{
                XorBlock( t+32, t+16, t+48 );

                Camellia_Feistel( t+48, SIGMA+32, t+56 );
                Camellia_Feistel( t+56, SIGMA+40, t+48 );

                ByteWord( t+16, u+8  );
                ByteWord( t+48, u+12 );

                for( i=0; i<34; i+=2 ){
                        RotBlock( u+KIDX2[i+0], KSFT2[i+0], u+16 );
                        RotBlock( u+KIDX2[i+1], KSFT2[i+1], u+18 );
                        WordByte( u+16, e+(i<<3) );
                }
        }
}

void Camellia_Encrypt( const int n, const Byte *p, const Byte *e, Byte *c )
{//n=鍵長   p=平文   e=拡張鍵   c=暗号文
        int i;

        XorBlock( p, e+0, c );

        for( i=0; i<3; i++ ){
                Camellia_Feistel( c+0, e+16+(i<<4), c+8 );
                Camellia_Feistel( c+8, e+24+(i<<4), c+0 );
        }
```

```
        Camellia_FLlayer( c, e+64, e+72 );

        for( i=0; i<3; i++ ){
                Camellia_Feistel( c+0, e+80+(i<<4), c+8 );
                Camellia_Feistel( c+8, e+88+(i<<4), c+0 );
        }

        Camellia_FLlayer( c, e+128, e+136 );

        for( i=0; i<3; i++ ){
                Camellia_Feistel( c+0, e+144+(i<<4), c+8 );
                Camellia_Feistel( c+8, e+152+(i<<4), c+0 );
        }

        if( n == 128 ){
                SwapHalf( c );
                XorBlock( c, e+192, c );
        }
        else{
                Camellia_FLlayer( c, e+192, e+200 );

                for( i=0; i<3; i++ ){
                        Camellia_Feistel( c+0, e+208+(i<<4), c+8 );
                        Camellia_Feistel( c+8, e+216+(i<<4), c+0 );
                }

                SwapHalf( c );
                XorBlock( c, e+256, c );
        }
}

void Camellia_Decrypt( const int n, const Byte *c, const Byte *e, Byte *p )
{
        int i;

        if( n == 128 ){
                XorBlock( c, e+192, p );
        }
        else{
                XorBlock( c, e+256, p );

                for( i=2; i>=0; i-- ){
                        Camellia_Feistel( p+0, e+216+(i<<4), p+8 );
                        Camellia_Feistel( p+8, e+208+(i<<4), p+0 );
                }

                Camellia_FLlayer( p, e+200, e+192 );
        }

        for( i=2; i>=0; i-- ){
                Camellia_Feistel( p+0, e+152+(i<<4), p+8 );
                Camellia_Feistel( p+8, e+144+(i<<4), p+0 );
```

```
        }

        Camellia_FLlayer( p, e+136, e+128 );

        for( i=2; i>=0; i-- ){
                Camellia_Feistel( p+0, e+88+(i<<4), p+8 );
                Camellia_Feistel( p+8, e+80+(i<<4), p+0 );
        }

        Camellia_FLlayer( p, e+72, e+64 );

        for( i=2; i>=0; i-- ){
                Camellia_Feistel( p+0, e+24+(i<<4), p+8 );
                Camellia_Feistel( p+8, e+16+(i<<4), p+0 );
        }

        SwapHalf( p );
        XorBlock( p, e+0, p );
}

void Camellia_Feistel( const Byte *x, const Byte *k, Byte *y )
{
        Byte t[8];

        t[0] = SBOX1(x[0]^k[0]);
        t[1] = SBOX2(x[1]^k[1]);
        t[2] = SBOX3(x[2]^k[2]);
        t[3] = SBOX4(x[3]^k[3]);
        t[4] = SBOX2(x[4]^k[4]);
        t[5] = SBOX3(x[5]^k[5]);
        t[6] = SBOX4(x[6]^k[6]);
        t[7] = SBOX1(x[7]^k[7]);

        y[0] ^= t[0]^t[2]^t[3]^t[5]^t[6]^t[7];
        y[1] ^= t[0]^t[1]^t[3]^t[4]^t[6]^t[7];
        y[2] ^= t[0]^t[1]^t[2]^t[4]^t[5]^t[7];
        y[3] ^= t[1]^t[2]^t[3]^t[4]^t[5]^t[6];
        y[4] ^= t[0]^t[1]^t[5]^t[6]^t[7];
        y[5] ^= t[1]^t[2]^t[4]^t[6]^t[7];
        y[6] ^= t[2]^t[3]^t[4]^t[5]^t[7];
        y[7] ^= t[0]^t[3]^t[4]^t[5]^t[6];
}

void Camellia_FLlayer( Byte *x, const Byte *kl, const Byte *kr )
{
        Word t[4],u[4],v[4];

        ByteWord( x, t );
        ByteWord( kl, u );
        ByteWord( kr, v );

        t[1] ^= (t[0]&u[0])<<1^(t[0]&u[0])>>31;
```

```
        t[0] ^= t[1]|u[1];
        t[2] ^= t[3]|v[1];
        t[3] ^= (t[2]&v[0])<<1^(t[2]&v[0])>>31;

        WordByte( t, x );
}

void ByteWord( const Byte *x, Word *y )
{
        int i;
        for( i=0; i<4; i++ ){
                y[i] = ((Word)x[(i<<2)+0]<<24) + ((Word)x[(i<<2)+1]<<16)
                     + ((Word)x[(i<<2)+2]<<8 ) + ((Word)x[(i<<2)+3]<<0 );
        }
}

void WordByte( const Word *x, Byte *y )
{
        int i;
        for( i=0; i<4; i++ ){
                y[(i<<2)+0] = (Byte)(x[i]>>24&0xff);
                y[(i<<2)+1] = (Byte)(x[i]>>16&0xff);
                y[(i<<2)+2] = (Byte)(x[i]>> 8&0xff);
                y[(i<<2)+3] = (Byte)(x[i]>> 0&0xff);
        }
}

void RotBlock( const Word *x, const int n, Word *y )
{
        int r;
        if( r = (n & 31) ){
                y[0] = x[((n>>5)+0)&3]<<r^x[((n>>5)+1)&3]>>(32-r);
                y[1] = x[((n>>5)+1)&3]<<r^x[((n>>5)+2)&3]>>(32-r);
        }
        else{
                y[0] = x[((n>>5)+0)&3];
                y[1] = x[((n>>5)+1)&3];
        }
}

void SwapHalf( Byte *x )
{
        Byte t;
        int  i;
        for( i=0; i<8; i++ ){
                t = x[i];
                x[i] = x[8+i];
                x[8+i] = t;
        }
}

void XorBlock( const Byte *x, const Byte *y, Byte *z )
```

```
{
        int i;
        for( i=0; i<16; i++ ) z[i] = x[i] ^ y[i];
}
```

DCcml.h

```
/************************************************
 * DCcml.h
 * Camellia 暗号関数
 ************************************************/

// 戻り値
#define NOERROR                  0                      // エラーなし
#define NOTENOUGHMEMORY  -1                     // メモリー不足
#define ACCESSERROR              -2                     // アクセスエラー
#define MATHERROR                -3                     // 数学的な誤り
#define KEYLENGTHERROR   -4                     // 鍵長不正
#define OTHERERROR               -5                     // その他のエラー

// 定数
#define MINKEYLENGTH     32                     // 最低鍵長（ビット）

typedef unsigned char uchar;
typedef unsigned short ushort;
typedef unsigned char Byte;
typedef unsigned long Word;

/**********************/
/*** RSA公開鍵暗号***/
/**********************/

// 暗号化　復号化
void Camellia_Ekeygen( const int, const Byte *, Byte * );
void Camellia_Decrypt( const int, const Byte *, const Byte *, Byte * );
```

DCcml.cpp

```
////////////////////////////////////////////////////////////////////////
//  DCcml.cpp : コンソールアプリケーション用のエントリポイントの定義
////////////////////////////////////////////////////////////////////////

#include "stdafx.h"

#include <iostream>
#include <string.h>
#include <stdlib.h>
#include "DCcml.h"
#include <stdio.h>
```

```
using namespace std;

FILE *stream;
FILE *stream1;
FILE *stream2;

extern ushort EXTKEY[4][8];

// 暗号文のHEX表示用
char toChar( int c )
{
        if( c >= 0 && c <= 9 )                  // 0～ならば
                return char( c + 0x30 ); // ASCIIに変換して返す
        else if( c >= 10 && c <= 15 )     // 10～ならば
                return char( c + 0x37 ); // A～FのASCIIを返す
        else
                return ' ';
}

// メイン
int main(                                                       // 正常終了時:0、異常時:1
                int argc,                                       // 引数の数
                char** argv )                           // 引数へのポインタ
{
        int i, len;
        int block;
        char j,k;
        unsigned char klen[8], key[32], key2[64];
        unsigned char exkey[256];
        int numclosed;
        int cnt;
        int lenp;
        int c;
        char* bufp;
        unsigned char* bufc;

        printf( "Start!¥n" );

        // 引数チェック
        if( argc != 4 ){                        // 使い方の誤り
                exit(1);
        }

   /* 鍵ファイルを開く*/
   if( (stream  = fopen( argv[1], "rb" )) == NULL )
      printf( "Can not open Key file.¥n" );

  /* 平文を書き込むファイルを開く*/
   if( (stream1  = fopen( argv[3], "wb" )) == NULL )
      printf( "Can not open file for Plane Text.¥n");

   /* 暗号文を読み出すファイルを開く*/
```

```
if( (stream2 = fopen( argv[2], "rb" )) == NULL )
   printf( "Can not open Encrypted file.¥n" );


      fseek(stream,0,0);
      fscanf(stream,"%s",klen);
      fscanf(stream,"%s",key2);
      len = atoi((char*)klen);

      for(i=0; i<len/8; i++){
               j = *(key2+2*i);
               k = *(key2+2*i+1);
               if(j>=0x30 && j<=0x39) j = j-0x30;
               else{
                        if(j>=0x41 && j<=0x46) j = j-0x41+0x0A;
               }
               if(k>=0x30 && k<=0x39) k = k-0x30;
               else{
                        if(k>=0x41 && k<=0x46) k = k-0x41+0x0A;
               }
               key[i] = j*0x10 + k;
      }
      key[len/8] = NULL;

      Camellia_Ekeygen( len, key, exkey );

// 暗号文
      fseek(stream2, 0, SEEK_END);
      long filelen = ftell(stream2);
      fseek(stream2,0,0);
      int head = sizeof(long);
      long mesLength = filelen;

      if(mesLength <= 1024){
               block = mesLength/16 + ((mesLength%16)?1:0);
               bufc = (unsigned char*)new(char[block*16 + 2]);
               bufp = (char*)new(char[block*16 + 2]);
               if(bufp == NULL){
                        printf("メモリ不足¥r¥n");
                        return(-1);
                        }

      // 暗文
               fseek(stream2, 0, 0);
               i = 0;
               do{
                        c = fgetc(stream2);
                        bufc[i]=c;
                        i=i+1;
               }while(c!=EOF);
               bufc[i-1]=NULL;
```

```
    // 復号化実行
                for(i=0;i<block;i++){
                        Camellia_Decrypt( len,  (unsigned char*)(bufc+i*16), exkey, (unsigned
char*)(bufp+i*16));
                   //n=鍵長   bufc=暗号文   exkey=拡張鍵   bufp=平文
                }

        // 復号文を書き込む
                lenp = *((long*)(bufp));
                bufp[lenp+head] = NULL;
                for( cnt = 0; cnt<lenp; cnt++ ){
                        fwrite( &(bufp[cnt+head]),sizeof(char),1, stream1);
                }
        }
        else{
                block = mesLength/16 + ((mesLength%16)?1:0);
                bufc = (unsigned char*)new(char[1024 + 2]);
                bufp = (char*)new(char[1024 + 2]);
                if(bufp == NULL){
                        printf("メモリ不足¥r¥n");
                        return(-1);
                        }
                long rBlen = block;
                int r = 0;
                do{
                        // 暗文
                        fseek(stream2, r*1024, 0);
                        i = 0;
                        do{
                                c = fgetc(stream2);
                                bufc[i]=c;
                                i=i+1;
                        }while((c!=EOF) && (i<=1024));
                        bufc[i-1]=NULL;

                        if(rBlen >= 1024/16){ block = 1024/16; }
                        if(rBlen < 1024/16){ block = rBlen;}

                // 復号化実行
                        for(i=0;i<block;i++){
                                Camellia_Decrypt( len, (unsigned char*)(bufc+i*16), exkey, (unsigned
char*)(bufp+i*16) );
                        }

                // 復号文を書き込む
                        if(r==0){
                                lenp = (*(long*)(bufp));
                                for( cnt = 0; cnt<(1024-head); cnt++ ){
                                        fwrite( &(bufp[cnt+head]),sizeof(char),1, stream1);
                                }
                                lenp -= 1024-head;
                        }
```

```
                        else{
                                if(rBlen >= 1024/16){
                                        for( cnt = 0; cnt<1024; cnt++ ){
                                                fwrite( &(bufp[cnt]),sizeof(char),1, stream1);
                                        }
                                        lenp -= 1024;
                                }
                                else{
                                        for( cnt = 0; cnt<lenp; cnt++ ){
                                                fwrite( &(bufp[cnt]),sizeof(char),1, stream1);
                                        }
                                }
                        }
                        r += 1;
                        rBlen -= 1024/16;
                }while(rBlen>0);
        }

        if( fclose( stream1 ) )
        printf("平文用ファイルは閉じられませんでした。¥n");
        if( fclose( stream2 ) )
                printf("暗号文のファイルは閉じられませんでした。¥n");
   /* 他のすべてのファイルを閉じる*/
        numclosed = _fcloseall( );

        delete[] bufp;
        delete[] bufc;

        printf( "End!¥n" );

        return 0;
}
```

Stdafx.cpp

```
////////////////////////////////////////////////////////////////////
//  stdafx.cpp : 標準インクルードファイルを含むソースファイル
//               DCMisty.pch 生成されるプリコンパイル済ヘッダー
//               stdafx.obj 生成されるプリコンパイル済タイプ情報
////////////////////////////////////////////////////////////////////
#include "stdafx.h"
// TODO: STDAFX.H に含まれていて、このファイルに記述されていない
// ヘッダーファイルを追加してください。
```

以上、VC++2005 でソフトを作成するために必要なファイルです。

なお、stdafx.h は VC++2005 が自動的に作るファイルです。内容は以下のようなものです。

Stdafx.h

```
////////////////////////////////////////////////////////////////////
// stdafx.h : 標準のシステムインクルードファイル、
//            または参照回数が多く、かつあまり変更されない
//            プロジェクト専用のインクルードファイルを記述します。
////////////////////////////////////////////////////////////////////

#if !defined(AFX_STDAFX_H__0970D4A0_4770_4806_B6B8_C06A5B282D03__INCLUDED_)
#define AFX_STDAFX_H__0970D4A0_4770_4806_B6B8_C06A5B282D03__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define WIN32_LEAN_AND_MEAN // Windows ヘッダーから殆ど使用されないスタッフを除外します

#include <stdio.h>

// TODO: プログラムで必要なヘッダー参照を追加してください。

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ は前行の直前に追加の宣言を挿入します。

#endif // !defined(AFX_STDAFX_H__0970D4A0_4770_4806_B6B8_C06A5B282D03__INCLUDED_)
```

おわり。　　2012.04.08　　宇山靖政