OpenSSLを用いたCamelliaの使い方

マルチプラットフォーム型共通鍵ブロック暗号

NTT 情報流通プラットフォーム研究所　2007/07/04 版

この資料は、共通鍵ブロック暗号“Camellia”のユーザーズガイドです。Camellia は、NTT と三菱電機が共同で開発した暗号で、ソフトウェア実装、ハードウェア実装を問わず、さまざまな環境で世界トップクラスの安全性・処理性能を実現します。その Camellia が OpenSSL に搭載されたことで、手軽に利用できるようになりました。

本ガイドは、暗号利用を検討されているシステム構築担当者やシステム開発担当者の方を対象とし、Camellia の特徴、利用方法等について説明したものです。本ガイドを参考に Camellia を実際に使ってみて下さい。使用した感想や気になった点等をご一報いただければ幸いです。

なお、詳細な情報については、文中に参照先を記載しましたので、そちらを参考にして下さい。

オープンソース Camellia とは

Camellia は、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[*1] 同等と国際的に認められた純国産暗号です。

世界トップクラスの安全性と処理性能を誇る Camellia について、以下に紹介します。

共通鍵ブロック暗号

共通鍵ブロック暗号方式とは、データを一定長のブロックに分割し暗号化するもので、公開鍵暗号方式に比べ 1000 倍以上も高速なため、メッセージやファイルの暗号化等に使われます。

ブロック長については、現在主流の 64 ビットから、安全性を高めた 128 ビットに移行しつつあります。ブロック長が 128 ビットであることは、次世代暗号の条件の一つであり、Camellia もブロック長を 128 ビットとしています。

また、あらゆる暗号技術では、鍵長が安全性を確保するうえで重要な要因とされていますが、Camellia では AES 同様、128/192/256 ビットの 3 種類の鍵長を用意しています。AES が採用されるまでの米国の標準暗号であった DES[*2] の鍵長が 56 ビットであることを考えると、十分な安全性を確保しているといえます。

世界最高水準の安全性・信頼性

1997 年に NIST(米国標準技術局)が世界中から募集し、約 5 年余の歳月をかけて評価した AES と同等の安全性・信頼性を Camellia は備えている、と多くの研究者から評価されています。日本では、CRYPTREC[*3] により、安全性に優れた暗号方式として電子政府推奨暗号に公式認定されました。その他にも、IETF[*4] からは日本で初めて TLS や IPsec、S/MIME での IETF 標準暗号に、ISO/IEC[*5] からは ISO/IEC 国際標準暗号規格として認定されています。

また、将来的な安全性期待度を示すセキュリティ・マージンは、AES を上回る 1.80～2.00 を確保しており、文字どおり「世界最高水準の安全性・信頼性を有している」といえます。

Camellia は、安全性、信頼性の他に、低消費電力の高効率小型ハードウェアをはじめ、Gbps クラスの高速ハードウェア、搭載メモリが少ない IC カード、および演算能力が低い小型 CPU 上でも、高速かつコンパクトな実装が可能である等、様々な環境で高速処理を実現します。

Camellia のオープンソース化

国際的なオープンソース・コミュニティである OpenSSL Project が提供する OpenSSLtoolkit のリビジョン 0.9.8c 以降に、Camellia が搭載されました。このことで、Camellia を利用した製品の開発が容易になり、暗号製品の相互接続性が高まることが期待できます。

その他にも、Linux、NSS、FreeBSD について、今夏頃へのリリース版搭載に向けた最終的な活動を行っています。

OpenSSL のインストール方法

はじめに、Camellia が搭載されている OpenSSL のインストール方法について説明します。

本ガイドでは、Windows 上で Camellia の動作が確認できるよう、Cygwin(Unix 擬似環境)上でのインストール方法を例示していますが、Linux や Solaris 上でも基本的には同様な方法で行うことができます。なお、Cygwin 上でビルドする場合は、gcc、+binutils、make や他のパッケージが必要となります。詳細な説明については、下記サイト、その他の関連サイトを参照して下さい。

■Cygwin project: http://cygwin.com/

OpenSSL のダウンロード

以下のサイトから、最新版の OpenSSL をダウンロードします。

■OpenSSL: http://www.openssl.org/source/

アーカイブの展開

以下のコマンドでアーカイブを展開し、展開したパスに移動します。

```
bash-3.2$ gzip -d openssl-0.9.8d.tar.gz
bash-3.2$ tar xvf openssl-0.9.8d.tar
bash-3.2$ cd <展開したディレクトリ>
```

コンパイルとインストール

次のコマンドにより、コンパイルおよびインストールを行います。なお、コマンド、パラメータ等は、環境により異なる場合がありますので、使用している環境に合わせて下さい。

```
bash-3.2$ ./config enable-camellia
[ shered library の場合は、下記のようにオプション
  を付与します。
bash-3.2$ ./config enable-camellia shared ]

bash-3.2$ make depend
bash-3.2$ make
bash-3.2$ make install
```

インストール結果の確認

インストールが正常に終了したならば、次のコマンドで Camellia が使用可能か確認します。この例では使用可能な暗号スイート*6 を表示しています。

```
$ cd /usr/local/ssl/bin
$ openssl ciphers
        :（省略）
..:DHE-RSA-CAMELLIA-256-SHA:DHE-DSS-...
```

OpenSSL による Camellia の利用方法

OpenSSL の暗号は、コマンドおよび API で利用することができます。Camellia に関しては、いずれの場合も、下表の暗号方式による暗号化・復号を行うことができます。

鍵長 ＼ 暗号ﾓｰﾄﾞ	CBC	CFB	ECB	OFB
128 ﾋﾞｯﾄ	○	○	○	○
192 ﾋﾞｯﾄ	○	○	○	○
256 ﾋﾞｯﾄ	○	○	○	○

コマンドを利用する方法

共通鍵暗号を利用する場合、主に enc コマンドを使用しますが、それぞれの暗号方式と同じ名前のコマンドを使用することもできます。

いずれの場合も、暗号方式は以下の形式で指定します。

“camellia”＋“-”＋ 鍵長 ＋“-”＋ 暗号モード

例）鍵長:128 ビット、暗号モード:ECB の場合

camellia-128-ecb

以下に、コマンドの指定例を示します。

◆enc コマンドを使用した場合の例

```
$ openssl enc -camellia-128-ecb -in plain.doc
    -out outf.bin -pass pass:･･･
```

◆暗号方式名のコマンドを使用した場合の例

```
$ openssl camellia-128-ecb -in plain.doc
    -out outf.bin -pass pass:･･･
```

鍵、初期化ベクタを生成する場合のパスワード

を指定することができます。パスワードは pass オプションで指定しますが、詳細な説明については、OpenSSL 関連の資料を参照して下さい。

また、通常は標準入力からデータを読み込み、標準出力に結果を出力しますが、標準入出力の代わりにファイルを指定することもできます。

API を利用する方法

OpenSSL で用意されている共通鍵暗号の API は暗号方式毎に用意されているため膨大ですが、EVP API を利用すると、暗号方式によらず共通のインタフェースでプログラミングすることができます。

EVP API を利用したプログラミングの例は、次節に示します。

OpenSSL によるサンプルコーディング

EVP API を利用した Camellia による暗号化・復号のサンプルコーディングの主な部分について解説します。なお、サンプルコーディングの全容については、資料末尾の Appendix をご覧下さい。

構造体の説明

サンプルコーディングで使用している構造体のうち、主なものを次に示します。

◆暗号コンテキスト

データを暗号化・復号するために必要な情報を保持しておくデータ構造体で、EVP_CIPHER_CTX_init 関数で初期化し、EVP_CIPHER_CTX_cleanup 関数でクリアする。

◆初期化ベクタ(IV：Initialization Vector)

平文の最初のブロックと XOR をとるために用意するデータブロック(CBC、CFB 等の暗号モードで使用)。

IV の値は、秘密にする必要はないが無作為な値にする必要がある。EVP_CipherInit 関数を用いて、暗号コンテキストに設定する。

関数の説明

サンプルコーディングで使用している関数のうち、主なものを以下に示します。この他の関数については、Appendix をご覧下さい。

◆EVP_CipherInit

暗号化・復号を行うために、アルゴリズム、鍵、IV 等を暗号コンテキストに設定する。

◆EVP_CipherUpdate

共通鍵アルゴリズムを使用して、データの暗号化・復号を行う。

◆EVP_CipherFinal

暗号化・復号を行った結果データの最終ブロックを取り出す。

◆EVP_CIPHER_CTX_init

暗号コンテキストを初期化する。本関数で初期化した暗号コンテキストは、EVP_CIPHER_CTX_cleanup 関数でクリアする。

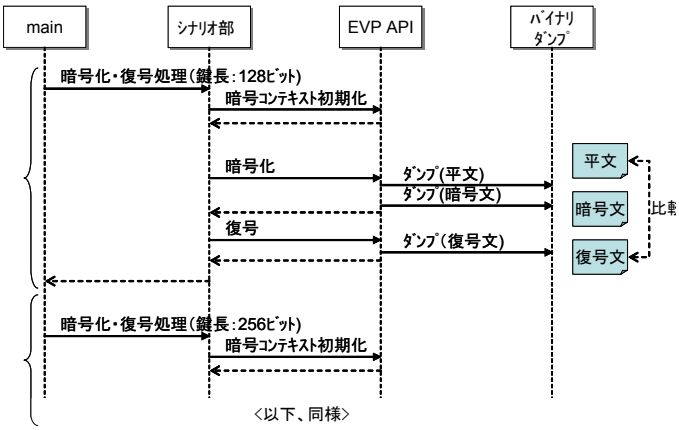
◆RAND_poll

擬似乱数生成器にシード(初期設定で使用する予測不能な秘密のデータ)を追加する。動作環境によっては RAND_seed 関数を使用する必要がある。

サンプルプログラムの仕様

◆処理内容

サンプルプログラムの処理の流れを次に示します。



このサンプルプログラムでは、平文を暗号化した後、暗号文を復号します。それぞれのデータをダンプすることにより、平文が暗号化されること、および平文と復号文が一致することを確認します。

◆処理条件

サンプルプログラムの処理条件は以下のとおりです。

API 種別	EVP API
鍵長	128 ビット(1 回目)、256 ビット(2 回目)
暗号モード	CBC モード
パディング	有り

サンプルコーディングの説明

◆暗号コンテキストの定義と初期化

はじめに、暗号化または復号に必要な暗号コンテキスト(暗号化用、復号用)自体を、EVP_CIPHER_ CTX_init 関数を用いて初期化します。

```
EVP_CIPHER_CTX enc, dec;
        :
EVP_CIPHER_CTX_init(&ctx) ;
```

暗号コンテキストの定義
enc：暗号化用
dec：復号用

暗号コンテキスト自体の初期化

◆暗号化または復号を行うための初期設定

次に、暗号化または復号を行うための情報を、EVP_ CipherInit 関数を用いて暗号コンテキストに設定します。

cipher	使用する暗号の種類
key	暗号化/復号に使用する鍵
iv	初期化ベクタ
kind	暗号化/復号の種別

暗号コンテキストに、各種情報を設定

```
ret = EVP_CipherInit(&ctx, cipher, key, iv, kind);
if(ret == 1) {
        :
```

処理結果を判定

サンプルコーディングでは、それぞれのパラメータを個別に設定するため、複数回にわたって EVP_CipherInit 関数を呼び出していますが、一度に設定することも可能です。

◆暗号化(または復号)の実行

次に、EVP_CipherUpdate 関数を用いて平文(Plain_data)を暗号化します。パラメータは以下のとおりです。

enc_data	暗号文格納域
enc_len	暗号文の長さ
plain_data	平文格納域
—	平文の長さ

```
/* 暗号化する　*/
ret = EVP_CipherUpdate(&ctx, enc_data, &enc_len,
            plain_data, sizeof(plain_data));
if(ret != 1) {
            :
/* 暗号化最終ブロックを処理する　*/
ret = EVP_CipherFinal(&ctx, enc_dara + enc_len,
            &tmplen);
if(ret != 1) {
            :
```

平文(plain_data)を暗号化する

最終ブロックに対する処理

最終ブロックにおいて、ブロックに満たないデータは、次の EVP_CipherUpdate が呼ばれるか、EVP_CipherFinal が呼ばれない限り暗号化は行なわれません。

図中の“最終ブロックによる処理”では、EVP_Cipher-Final により最終ブロックを暗号化しています。

以上が、EVP API を利用して暗号化する場合のおおまかな流れと、主要部分のコーディングです。

なお、OpenSSL の各 API の使用方法については、“OpenSSL/doc”ディレクトリ配下のファイルに記述されていますので参照して下さい。

Camellia を利用した製品の例

Camellia を利用した製品には、Web でのコンテンツ配信をセキュア化する製品、メールを暗号化する製品、送受信データを暗号化する製品等があります。以下にその一部を紹介します。

(1) CipherCraft【NTT ソフトウェア株式会社】

Camellia や RSA 等、複数の暗号アルゴリズムを実装した暗号ライブラリで、CipherCraft/Mail(メール暗号化)、CipherCraft/VPN(SSL-VPN)、Cipher Craft/File(ファイル暗号化)という製品群の総称名

(2) FILE LOCK II【NTT アドバンステクノロジ株式会社】

ファイルやフォルダを暗号化するソフト

(3) SmartLeakProduct【NTT アドバンステクノロジ株式会社】

多彩な暗号化機能を備えた内部情報漏洩対策システム

(4) SKIP9 認証 BOX サーバ【大井電気株式会社】

クライアント・サーバ間双方向認証システム

(5) CWAT【株式会社インテリジェントウェイブ】

内部情報漏洩対策システム

(6) その他製品多数

Camellia に関する情報の入手方法

Camellia に関する最新情報や紹介記事等は、次のサイトから入手することができます。

■サイト名：NTT の暗号要素技術 > Camellia

http://info.isl.ntt.co.jp/crypt/camellia/index.html

■Camellia に関する情報

- ●Camellia に関するニュースリリース・関連記事
- ●Camellia の紹介
- ●標準化情報(Camellia を認定した標準化団体等)
- ●Camellia を採用したセキュリティ製品の紹介
- ●Camellia 仕様書等の技術情報

■本件問い合わせ先：camellia@lab.ntt.co.jp

---

*1　AES(Advanced Encryption Standard)：米国政府の標準暗号化方式。

*2　DES(Data Encryption Standard)：1977 年に、NBS によって米国の標準暗号に採用された秘密鍵暗号化アルゴリズム。2005 年に規格廃止。

*3　CRYPTREC(Cryptography Research and Evaluation Comittees)：総務省及び経済産業省が共同で主管している暗号技術評価委員会。電子政府推奨暗号の選定や監視、暗号技術の評価等を実施。

*4　IETF(Internet Engineering Task Force)：インターネットに関する技術仕様を策定している機関。

*5　ISO/IEC：　国際標準化機構/国際電気標準会議。

*6　暗号スイート：SSL コネクションが認証、鍵交換、ストリーム暗号化を行うために使う、アルゴリズムの組合せ。

Appendix 1：関数一覧

サンプルプログラムで使用している関数の一覧*1を以下に示します。

No.	関数名	機能概要
1	ENV_CipherInit	共通鍵アルゴリズムを使用して暗号化・復号を行うために、アルゴリズム、鍵、IV(初期化ベクタ)などを暗号コンテキストに設定する。
2	ENV_CipherUpdate	共通鍵アルゴリズムを使用して、データの暗号化・復号を行う。
3	ENV_CipherFinal	暗号化・復号を行った結果データの最終ブロックを読み込む。
4	ENV_CIPHER_CTX_init	暗号コンテキストを初期化する。
5	ENV_CIPHER_CTX_cleanup	暗号コンテキストをクリアする。
6	ENV_CIPHER_CTX_block_size	暗号文を格納するブロックのサイズ(バイト長)を取得する。
7	ENV_CIPHER_CTX_iv_length	IVのサイズ(バイト長)を取得する。
8	ENV_CIPHER_CTX_set_padding	暗号コンテキストにパディングの有無を設定する。
9	OpenSSL_add_all_algorithms	全アルゴリズムのEVPオブジェクトと文字列を関連付けた内部テーブルを読み込む。
10	CRYPTO_cleanup_ex_data	cryptoライブラリの内部領域を解放する。
11	RAND_poll	PRNG(擬似乱数生成器)にシード*2を追加する。
12	RAND_seed	動作環境などによって、PRNGに十分なシードが与えられないとき、本関数でPRNGにシードを追加する。
13	RAND_cleanup	PRNGをクリアする。
14	RAND_bytes	擬似乱数を生成する。
15	ERR_load_crypto_strings	暗号モジュール用のエラーメッセージを読み込む。
16	ERR_free_strings	読み込んだエラーメッセージ領域を解放する。
17	ERR_remove_state	スレッドがもつエラーキューを解放する。

*1：メモリ確保・解放関数については省略しました。

*2：シードとは、PRNGの初期状態の設定に使用する予測不能な秘密のデータのこと。

Appendix 2：サンプルコーディング

◆ 以下は、EVP API を利用し、下記条件で暗号化および復号を行った場合のサンプルコーディングです。
【 API 種別：EVP API　鍵長：128 ビット(1 回目)、256 ビット(2 回目)　暗号モード：CBC モード　パディング：有り　】

```
/*
* Camellia EVP API を利用した共通鍵暗号化・復号の
* サンプルプログラム
 *
 */
#include <stdio.h>
#include <string.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/rand.h>
#include <openssl/crypto.h>

#if defined(_MSC_VER) && defined(_WIN32)
#include <openssl/applink.c>
#endif
#define PLAIN_DATA_SIZE 32
#define NO_PAD 0
#define PAD 1
#define UNDEF -1
#define NO_IV 0
#define SET_IV 1
#define ENCRYPT 1
#define DECRYPT 0

/*
 * ―<binary data 表示>―
 *  暗号文等のバイナリデータを表示する。
 */
void binary_print_data(const unsigned char* data,
                                    int len, char* msg)
{
      int i;
      printf("%s:¥n",msg);
      for (i=0; i<len; i++) {
          printf("0x%02x, ", data[i]);
          if (((i+1) % 8) == 0) {
              printf("¥n");
          }
      }
      printf("¥n");
}

/*
 * ―<暗号化・復号処理>―
 *  パラメータで指定された暗号アルゴリズムを用いて
 *  平文を一括で暗号化、および暗号文を一括で復号する。
 * <引数>
 *     alg: 共通鍵暗号化アルゴリズム
 * keylen: 鍵長(byte ｻｲｽﾞ)
 *  setiv: IV(IV の設定必要有), NO_IV(IV の設定必要無)
 *    pad: PAD(ﾊﾟﾃﾞｨﾝｸﾞ有), NO_PAD(ﾊﾟﾃﾞｨﾝｸﾞ無)
 *
 *<戻り値>
 *      1:正常終了
 *      0:異常終了
 */
int sample_scenario(const EVP_CIPHER *cipher, int keylen,
int setiv, int pad)
{
      int ret = 0;
      EVP_CIPHER_CTX enc,dec;
      unsigned char *key = NULL;
      unsigned char *iv = NULL;
      int ivlen = 0;
      const unsigned char plain_data
      [PLAIN_DATA_SIZE] = {
          0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,
          0x09,0x00,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
          0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,
          0x99,0x00,0xaa,0xbb,0xcc,0xdd,0xee,0xff};
      unsigned char *enc_data = NULL;
      unsigned char *dec_data = NULL;
      int enclen;
      int declen;
      int tmplen;

      /************ ここから暗号化処理 **************/
      EVP_CIPHER_CTX_init(&enc);

      ret = EVP_CipherInit(&enc, cipher, NULL, NULL,
ENCRYPT);
      /** 暗号化に使用する EVP_CIPHER_CTX を初期化 */
      if(ret != 1) {
          printf("EVP_CipherInit failed!¥n");
          goto cleanup;
      }

      /** 主鍵の生成(ここでは乱数を利用) */
      key = (unsigned char*)OPENSSL_malloc(keylen);
          RAND_bytes(key,keylen);
      binary_print_data(key,keylen,"key");

      /**
       * IV 生成(ここでは乱数を利用)
       *   ECB モードには IV を使用しないため設定する
         *   必要はない。
       */
      if(setiv == SET_IV)
      {
```

```
        ivlen = EVP_CIPHER_CTX_iv_length(&enc);
        iv = (unsigned char*)OPENSSL_malloc(ivlen);
        RAND_bytes(iv,ivlen);
        binary_print_data(iv,ivlen,"iv");
    }

    /**
     * 主鍵・IV を EVP_CIPHER_CTX に設定
     */
    ret = EVP_CipherInit(&enc,NULL, key, iv, ENCRYPT);
    if(ret != 1) {
        printf("EVP_CipherInit failed!¥n");
    }

    /**
     * ﾊﾟﾃﾞｨﾝｸﾞ設定
     * OFB,CFB モードはパディングと無関係であるため
       *設定自体が必要ない。
     */
    if(pad != UNDEF)
    {
        ret = EVP_CIPHER_CTX_set_padding(&enc, pad);
        if(ret != 1) {
    printf("EVP_CIPHER_CTX_set_paddingfailed!¥n");
            goto cleanup;
        }
    }

    /** 暗号化結果出力バッファ初期化 */
    if(pad == PAD) {
        printf("padding:ON¥n");
        tmplen = EVP_CIPHER_CTX_block_size(&enc);
    tmplen = ((PLAIN_DATA_SIZE +  tmplen)/tmplen)
* tmplen;
    }else{
        printf("padding:OFF¥n");
        tmplen = PLAIN_DATA_SIZE;
    }
    enc_data = (unsigned char*)OPENSSL_malloc(tmplen);
    memset(enc_data, 0, tmplen);
    tmplen = 0;

    /** 暗号化 */
    ret = EVP_CipherUpdate(&enc, enc_data, &enclen,
    plain_data, sizeof(plain_data));
    if(ret != 1) {
        printf("EVP_CipherUpdate failed!¥n");
        goto cleanup;
    }

    /** 暗号化最終ブロック処理 */
    ret = EVP_CipherFinal(&enc, enc_data + enclen,
&tmplen);
    if(ret != 1) {
        printf("EVP_CipherFinal failed!¥n");
        goto cleanup;
    }
    enclen += tmplen;
    binary_print_data(plain_data, PLAIN_DATA_SIZE,
 "plain_data");
    binary_print_data(enc_data, enclen, "enc_data");

    /********** ここから復号処理 **********/
    /** 復号に使用する EVP_CIPHER_CTX 構造体初期化 */
    EVP_CIPHER_CTX_init(&dec);

    /** EVP_CIPHE_CTX 構造体設定 */
    ret = EVP_CipherInit(&dec,cipher,key,iv, DECRYPT);
    if(ret != 1) {
        printf("EVP_CipherInit failed!¥n");
        goto cleanup;
    }

    /** パディング設定 */
    ret = EVP_CIPHER_CTX_set_padding(&dec, pad);
    if(ret != 1) {
    printf("EVP_CIPHER_CTX_set_padding  failed!¥n");
        goto cleanup;
    }

    /** 復号結果出力バッファ初期化 */
    dec_data = OPENSSL_malloc(enclen);
    memset(dec_data, 0, enclen);

    /** 復号処理 */
    ret = EVP_CipherUpdate(&dec, dec_data, &declen,
enc_data, enclen);
    if(ret != 1) {
        printf("EVP_CipherUpdate failed!¥n");
        goto cleanup;
    }

    /** 最終ブロック処理 */
    ret = EVP_CipherFinal(&dec, dec_data + declen,
&tmplen);
    if(ret != 1) {
        printf("EVP_CipherFinal failed!¥n");
        goto cleanup;
    }
    declen += tmplen;
    binary_print_data(dec_data, declen, "dec_data");

    /** 復号結果チェック*/
    if(!(declen == PLAIN_DATA_SIZE  &&
        memcmp(plain_data,dec_data,declen) == 0))
    {
        printf("Decryption is failed!¥n");
        ret = 0;
        goto cleanup;
    }

cleanup:
    /** EVP_CIPHER_CTX クリーンアップ */
    EVP_CIPHER_CTX_cleanup(&enc);
```

```
	EVP_CIPHER_CTX_cleanup(&dec);

	/** 暗号化・復号結果出力バッファ解放 */
	if(key)OPENSSL_free(key);
	if(iv)OPENSSL_free(iv);
	if(enc_data)OPENSSL_free(enc_data);
	if(dec_data)OPENSSL_free(dec_data);

	/** ERR 情報クリア */
	ERR_clear_error();
	return ret;
}

/*
 * ―〈サンプル プログラム〉―
 *  EVP_Cipher 系の API を利用して
 *  各暗号アルゴリズムの動作を確認する。
 *
 * 〈引数〉
 *    alg: 共通鍵暗号化アルゴリズム
 * keylen: 鍵長(byte ｻｲｽﾞ)
 *  setiv: IV(IV の設定必要有), NO_IV(IV の設定必要無)
 *    pad: PAD(ﾊﾟﾃﾞｨﾝｸﾞ有), NO_PAD(ﾊﾟﾃﾞｨﾝｸﾞ無)
 *
 * 〈戻り値〉
 *      1:正常終了
 *      0:異常終了
 *
 */
int main()
{
	/** アルゴリズムテーブル読み込み */
	OpenSSL_add_all_algorithms();

	/** エラーメッセージ読み込み */
	ERR_load_crypto_strings();

	/** 擬似乱数生成器初期化*/
	RAND_poll();
	while (RAND_status() == 0) {
	    int rnd = rand();
	    RAND_seed(&rnd, sizeof(rnd));
	}

	/******* 暗号化・復号処理 *******/
	/* Camellia(CBC ﾓｰﾄﾞ, 鍵長:128bit,ﾊﾟﾃﾞｨﾝｸﾞ無) */
	printf("―――  Camellia(mod:CBC,  key:128bit,
pad:off) ―――¥n");
	sample_scenario(EVP_camellia_128_cbc(),16,
SET_IV, NO_PAD);

	/* Camellia(CBC ﾓｰﾄﾞ, 鍵長:256bit, ﾊﾟﾃﾞｨﾝｸﾞ有 */
	printf("―――  Camellia(mod:CBC,  key:256bit,
pad:on) ―――¥n");
	sample_scenario(EVP_camellia_256_cbc(), 32,
SET_IV, PAD);

	/** 乱数解放 */
	RAND_cleanup();

	/** エラーメッセージ解放 */
	ERR_remove_state(0);
	ERR_free_strings();

	/**
	 * 全アルゴリズムの EVP 構造体と文字列を関連付けた
	 * データ削除
	 */
	EVP_cleanup();

	/** crypto ライブラリの内部領域解放 */
	CRYPTO_cleanup_all_ex_data();

	return 0;
}
```

以上が、EVP API を利用したサンプルコーディングです。

補足 1. Makefile のサンプル

本サンプルプログラムは、以下の Makefile でコンパイル、リンクしました。

HOME、パス等は、それぞれの環境に合わせて修正して下さい。

```
PROG_NAME = sample
HOME      = /home/test
OPENSSLINC = $(HOME)/openssl-0.9.8e/include
LIBS      = $(HOME)/openssl-0.9.8e
CFLAGS = -I$(OPENSSLINC) -c -g -DDEBUG
.c.o:
        ${CC} ${CFLAGS} $<
OBJ     = ¥
        sample.o
${PROG_NAME}: ${OBJ}
        ${CC} -o $@ ${OBJ} -L$(LIBS) -lcrypto -L$(LIBS)
-lssl
clean:
        -rm *.o
```

補足 2. サンプルプログラムの実行結果

本ガイドのサンプルプログラムの実行結果を以下に示します。

```
bash-3.2$ ./sample.exe
------ Camellia(mod:CBC, key:128bit, pad:off) ---
key:
0xf6, 0xf0, 0x13, 0xa2, 0x68, 0xc0, 0xa4, 0xee,
0x2a, 0x67, 0xd7, 0x2b, 0xc8, 0xb7, 0xf9, 0x9a,

iv:
0x6a, 0xf2, 0xe4, 0x59, 0x7a, 0xec, 0x3a, 0xd2,
0x3a, 0x0c, 0xb8, 0x56, 0x04, 0x92, 0xb7, 0xe4,

padding:OFF
plain_data:
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
0x09, 0x00, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
0x99, 0x00, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff,

enc_data:
0x58, 0x51, 0x3e, 0x65, 0x54, 0x39, 0xe1, 0xd5,
0x30, 0x18, 0x80, 0x78, 0x98, 0x65, 0xd7, 0x33,
0xb1, 0xe5, 0x77, 0x84, 0x01, 0xf7, 0xf5, 0x88,
0x81, 0xde, 0x1b, 0x0b, 0x70, 0x51, 0x82, 0x19,

dec_data:
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
0x09, 0x00, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
0x99, 0x00, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff,

------ Camellia(mod:CBC, key:256bit, pad:on) ---
key:
0x5a, 0x7a, 0x23, 0x8a, 0xe2, 0x23, 0xa4, 0xe3,
0xe8, 0xb6, 0xb0, 0x0c, 0x98, 0xa6, 0xd4, 0x84,
0x4c, 0x27, 0xd0, 0xec, 0x9e, 0xed, 0x36, 0xad,
0xc3, 0x39, 0x3b, 0x01, 0xca, 0xa5, 0x72, 0x49,

iv:
0xa0, 0xa7, 0xa3, 0xb3, 0x85, 0x68, 0xc4, 0x04,
0x72, 0x11, 0x93, 0x25, 0xf0, 0x94, 0xae, 0x18,

padding:ON
plain_data:
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
0x09, 0x00, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
0x99, 0x00, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff,

enc_data:
0xde, 0x94, 0x57, 0x70, 0xb7, 0xfc, 0xa2, 0x94,
0x85, 0xa5, 0xaf, 0x1d, 0xe4, 0xf1, 0xe2, 0xb7,
0x5d, 0x73, 0xcc, 0x48, 0x58, 0xbc, 0xf4, 0xb3,
0xb4, 0xe5, 0x6b, 0x17, 0x91, 0x47, 0x17, 0xa5,
0x79, 0xa7, 0xd3, 0x88, 0x5e, 0xfa, 0x5d, 0x6c,
0xe0, 0x45, 0x23, 0xbb, 0x5d, 0xbe, 0x58, 0x7c,

dec_data:
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
0x09, 0x00, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
0x99, 0x00, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff,

bash-3.2$
```

以上が、サンプルプログラムの実行結果です。