7. 暗号アルゴリズム

本章では、各種の暗号アルゴリズムについてサンプルプログラムを紹介します。

7.1　共通ラッパー

この章のプログラムはコマンドとして動作するように、共通のラッパーとして動作するmain関数を用意しています。common/main.cにその内容が格納されています。main.cのmain関数は一連のアーギュメントのチェックと解析を行いalgo_main関数を呼び出します。 algo_main関数はアルゴリズムサンプルごとの個別の関数です。このラッパーを使用することにより、個別のアルゴリズムの関数は、アルゴリズムのための固有の処理だけを行うことができます。

このラッパー関数を使ったコマンドは以下のアーギュメントを受け付けます。

- 第一アーギュメント：ファイル名(デフォルト入力、省略可)
- 第二アーギュメント：ファイル名(デフォルト出力、省略可)

以下のオプションアーギュメント：

- -e : 暗号化処理
- -d : 復号処理
- -k : 次のアーギュメントで１６進の鍵値を指定
- -i : 次のアーギュメントで１６進のIV値を指定
- -t : 次のアーギュメントで１６進のタグ値を指定

algo_main関数は、main.h内で以下のように定義されています。main.c内のmain関数はアーギュメントの解析内容をalgo_main関数のアーギュメントに引き継ぎます。

```
void algo_main(int mode, FILE *fp1, FILE *fp2,
                unsigned char *key, int key_sz,
                unsigned char *iv,  int iv_sz,
                unsigned char *tag, int tag_sz
               );
```

第一、第二アーギュメントで指定されたファイルはfopenしてファイルディスクリプターfp1, fp2に引き継ぎます。デフォルトのオープンモードはfp1は"rb", fp2は"wb"です。変更する場合はMakefile内でコンパイル時定義のマクロ名OPEN_MODE1, OPEN_MODE2で任意のモード文字列を定義します。

オープンに失敗した場合はmain.c内でエラーメッセージを出力し、algo_mainは呼び出しません。 アーギュメントが省略された場合はfp1, fp2にはNULLが引き渡されます。

ラッパーとしては -e, -d で示されたモード、-k, -i, -tで指定された任意の長さの１６進値をalgo_mainに引き渡します。不正な16進文字列を検出した場合はmain関数内でエラーを出力し、algo_mainは呼び出しません。指定されていないオプションアーギュメントはポインタ値にNULLが引き渡されます。オプションアーギュメントの必要性、サイズが適切かどうかは個々のalgo_mainにてチェックします。

バッファーサイズ

サンプルで使用している暗号処理APIはメモリーサイズの許す限り大きなバッファで一度に処理することができまが、小さな処理単位を繰り返して大きなサイズのデータを処理する例を示すためめにあえてバッファサイズを制限しています。各アルゴリズムのソースコードの先頭付近にある「#define BUFF_SIZE」の定義は適当に変更することができます。

7.2 ハッシュ

OpenSSL/wolfSSLでは、与えられたデータをハッシュ(メッセージダイジェスト)を求めるる為の"EVP_MD_CTX", "EVP_Digest"で始まる一連の関数を用意しています。実行するハッシュアルゴリズムはEVP_DigestInit関数で初期化する際に指定します。指定できる主なアルゴリズムを下の表にまとめます。

関数名	機能
EVP_MD_CTX_new	ハッシュ処理コンテクストを確保
EVP_MD_CTX_free	ハッシュ処理コンテクストを解放
EVP_DigestInit	ハッシュ種別を指定してコンテクストを初期化
EVP_DigestUpdate	対象メッセージを追加。繰り返して呼び出し可能
EVP_DigestFinal	ハッシュ値を求める

表：ハッシュ処理のための基本的な関数

アルゴリズム	初期化関数名
MD5	EVP_md5
Sha1	EVP_sha1
Sha224	EVP_sha224
Sha256	EVP_sha256
Sha384	EVP_sha384
Sha512	EVP_sha512
Sha512/224	EVP_sha512_224
Sha512/256	EVP_sha512_256
Sha3/224	EVP_sha3_224
Sha3/256	EVP_sha3_256
Sha3/284	EVP_sha3_384
Sha3/512	EVP_sha3_512

表：EVP_DigestInitで指定できる主なハッシュアルゴリズム

1) プログラムの概要

このプログラムは引数を最大２つ取ります。第1引数は入力データを格納した入力データファイルパスを与えます。第2引数はハッシュデータを出力する先のファイルパスです。第2引数はオプションであり、指定されない場合はハッシュデータは標準出力に出力されます。与える入力データのサイズは任意です。出力されるハッシュデータはSHA256の場合は32バイトのバイナリデータとして出力されます。

処理のはじめに処理コンテクストの管理ブロックを用意します(EVP_MD_CTX_init)。次に"EVP_DigestInit"関数により用意したコンテクストに対してハッシュアルゴリズムを指定します。この例ではSHA256を指定します。上記の表に従ってこの部分を変更することで、他のハッシュ アルゴリズムの処理を行うことができます。

ハッシュ処理は"EVP_DigestUpdate"関数によって行います。メモリーサイズの制限が許す場合は入力データ全体を一括して"EVP_DigestUpdate"関数に渡すことができますが、制限がある場合は適当な大きさに区切って"EVP_DigestUpdate"関数を 複数回呼び出すこともできます。

最後に"EVP_DigestFinal"関数によりコンテクストに保存されていたハッシュ値をバッファに出力させ、その後ファイル、 あるいは標準出力に出力して終了します。

```
void algo_main( ... )
{
    EVP_MD_CTX_init(&mdCtx);

    if (EVP_DigestInit(&mdCtx, EVP_sha256()) != SSL_SUCCESS) {
        /* エラー処理 */
    }

    while (1) {
        if ((inl = fread(in, 1, BUFF_SIZE, infp)) <0) {
           /* エラー処理 */
        }
        if (EVP_DigestUpdate(&mdCtx, in, inl) != SSL_SUCCESS) {
            /* エラー処理 */
        }
        if(inl < BUFF_SIZE)
            break;
    }

    if (EVP_DigestFinal(&mdCtx, digest, &dmSz) != SSL_SUCCESS) {
        /* エラー処理 */
    }

    if (fwrite(digest, dmSz, 1, outfp) != 1) {
        /* エラー処理 */
    }

    ...
}
```

7.3 メッセージ認証コード

OpenSSL/wolfSSLでは、与えられたデータを鍵と共にメッセージ認証コードを生成する為の"HAC"で始まる次の様な一連の関数を用意しています。このセクションでは、このHMAC関数を使用したプログラム例について解説します。

機能	関数名
コンテクスト確保	HMAC_CTX_new
コンテクスト複製	HMAC_CTX_copy
MD構造体の取得	HMAC_CTX_get_md
初期設定	HMAC_Init_ex
ハッシュ更新	HMAC_Update
終了処理	HMAC_Final
コンテクスト解放	HMAC_CTX_free

1) 全体の流れ

メッセージ認証コードの生成は、入力データと鍵データを合成したうえで指定したハッシュアルゴリズムを使ってハッシュすることで行います。処理のはじめに ハッシュアルゴリズムを選択してメッセージダイジェスト構造体を取得しておきます。その後、処理コンテクストとして管理ブロック"HMAC_CTX"を確保します。次に"Init"関数により初期設定関数で確保したコンテクストに対してメッセージダイジェスト構造体、ハッシュ対象のデータと合成する鍵データを与えて初期化を実行します。

ハッシュ処理は"HMAC_Update"関数によって行います。メモリーサイズの制限が許す場合は入力データ全体を一括して"Update"関数に渡すことができますが、制限がある場合は適当な大きさに区切って"Update"関数を複数回呼び出すこともできます。最後に"Final"関数によりコンテクストに保存されていたハッシュ値(メッセージ認証コード)をバッファに出力し、ファイルに書き出します。

2) サンプルプログラム

以下にHMAC関数を使用してハッシュ処理を実現するサンプルプログラムを示します。

このプログラムでは次のコマンドアーギュメントを受付ます。

- 入力ファイル：指定されたファイル名のファイルを入力データとして使用します。
- 出力ファイル：指定されたファイル名のファイルに結果のデータを出力します。省略した場合、標準出力に出力します。

- "-k" の次のアーギュメントで鍵値を１６進数で指定します。最低でも1バイトの指定が必要です

このプログラムで使用するハッシュアルゴリズムは"SHA1"を使用しています。ハッシュアルゴリズムが設定されたメッセージダイジェスト構造体(EVP_MD)を後述のHMAC初期化関数に渡すことでハッシュアルゴリズムを指定できます。

メッセージダイジェスト構造体の取得はEVP_get_digestbyname関数にアルゴリズムを示す文字列を指定することで行います。下表にEVP_get_digestbyname関数に指定できるハッシュアルゴリズム文字列の例を示します。

ハッシュアルゴリズム	アルゴリズム文字列
MD5	"MD5"
BLAKE128	"BLAKE128"
BLAKE256	"BLAKE256"
SHA1	"SHA1"
SHA224	"SHA224"
SHA256	"SHA256"
SHA384	"SHA384"
SHA3_224	"SHA3_224"
SHA3_256	"SHA3_256"
SHA3_384	"SHA3_384"
SHA3_512	"SHA3_512"

入力データはBUFF_SIZEで指定したサイズの入力データ用バッファに繰り返し読み込まれ、その都度HMAC_Update関数に渡し、メッセージ認証コードの計算を更新させます。全ての入力データを与え終わったら最後にHMAC_Final関数を呼び出してメッセージ認証コードをバッファに取り出します。メッセージ認証コードのサイズはハッシュアルゴリズムによって変わりますが、SHA1の場合は20バイト出力されます。最後に、メッセージ認証コードを指定された出力ファイル（あるいは標準出力）に出力して終了します。

```
    コマンドアーギュメントの処理

    HMAC_CTX* hctx;
    const EVP_MD *md;

    /* ハッシュアルゴリズムの選択 */
    md = EVP_get_digestbyname("SHA1");

    /* コンテキストの確保 */
```

```
    hctx = HMAC_CTX_new();

    /* コンテクストの初期化 */
    HMAC_Init_ex(hctx, key, key_sz, md, NULL);

    while (1) {
        inl = fread(in, 1, BUFF_SIZE, infp);
        HMAC_Update(hctx, (const unsigned char*)in, inl);
        if(inl < BUFF_SIZE)
            break;
    }

    /* メッセージ認証コードの取得 */
    HMAC_Final(hctx, hmac, &len);

    fwrite(hmac, len, 1, outfp);

    HMAC_CTX_free(hctx);
```

7.4 共通鍵暗号

OpenSSL/wolfSSLでは、共通鍵暗号の処理のために"EVP"で始まる一連の関数が用意されています。このセクションでは、このEVP関数の一般規則とそれを使用した共通鍵暗号のプログラム例について解説します。

1) 全体の流れ

処理のはじめに"CTX_new"関数により処理コンテクストを管理するための管理ブロックを確保します。次に"Init"関数により初期設定関数で確保したコンテクストに対して鍵、IVなどのパラメータを設定します。

暗号化復号処理は"Update"関数によって行います。メモリー上の入力バッファに対して処理が行われ、出力バッファに出力されます。メモリーサイズの制限が許す場合は入力データ全体を一括して"Update"関数に渡すことができますが、制限がある場合は適当な大きさに区切って"Update"関数を複数回呼び出すこともできます。その際ブロック型暗号のブロックサイズを気にすることなく、適当な処理サイズを指定することができます。最後に"Final"関数により半端なデータに対するパディングを処理を行います。

最後に終了後管理ブロックを解放します。

2) サンプルプログラム

以下にEVP関数を使用して共通鍵暗号処理を実現するサンプルプログラムを示します。"CIPHER" 定数の定義を変更することで各種の暗号アルゴリズム、利用モードを処理することができます（指定できる暗号スイートについては"6) 暗号アルゴリズム、利用モード"を参照）。

動作可能なサンプルコードはExamples/2.Chrypto/sym/aes-cbc.c を参照してください。このプログラムでは次のコマンドアーギュメントを受付ます。

- 入力ファイル：指定されたファイル名のファイルを入力データとして使用します。
- 出力ファイル：指定されたファイル名のファイルに結果のデータを出力します。省略した場合、標準出力に出力します。
- "-e"は暗号化、"-d"は複合を指定します。指定のない場合は暗号化処理をします。
- "-k" の次のアーギュメントで鍵値を１６進数で指定します。
- "-i" の次のアーギュメントでIV値を１６進数で指定します。

```
#define CIPHER EVP_aes_128|CBC()

    コマンドアーギュメントの処理

    evp = EVP_CIPHER_CTX_new(); /* コンテクスト確保 */
    EVP_CipherInit(evp, CIPHER, key, iv, mode);  /* アルゴリズム、鍵、IV、モードの設定 */


    while (1) {
        inl = fread(in, 1, BUFF_SIZE, stdin));
        if(inl < BUFF_SIZE) break;
        EVP_CipherUpdate(evp, out, &outl, in, inl); /*  暗号、復号 */
        fwrite(out, 1, outl, outfp);
    }
    EVP_CipherFinal(evp, out, &outl); /* パディング処理 */
    fwrite(out, 1, outl, outfp);

    EVP_CIPHER_CTX_free(evp);
```

3) 認証付き暗号(AEAD)

AES-GCMなど認証付き暗号の場合は認証タグを取扱う必要があります。下のプログラムで示すように、暗号化の際は、"Final"の後に復号の際に使用する認証タグを得ておきます。復号の際は、"Final"の前にそのタグを設定します。"Final"処理の返却値が成功であることで認証タグの検証が成功したことを確認します。

動作可能なサンプルコードはExamples/2.Chrypto/sym/aes-cbc.c を参照してください。このプログラムでは次のコマンドアーギュメントを受付ます。

- 入力ファイル：指定されたファイル名のファイルを入力データとして使用します。

- 出力ファイル：指定されたファイル名のファイルに結果のデータを出力します。省略した場合、標準出力に出力します。
- "-e"は暗号化、"-d"は複合を指定します。指定のない場合は暗号化処理をします。
- "-k" の次のアーギュメントで鍵値を１６進数で指定します。
- "-i" の次のアーギュメントでIV値を１６進数で指定します。
- "-t" の次のアーギュメントでタグ値を１６進数で指定します。

```
   コマンドアーギュメントの処理

    evp = EVP_CIPHER_CTX_new(); /* コンテクスト確保 */
    EVP_CipherInit(evp, CIPHER, key, iv, mode);  /* アルゴリズム、鍵、IV、モードの設定 */

    while (1) {
        inl = fread(in, 1, BUFF_SIZE, stdin));
        if(inl < BUFF_SIZE) break;
        EVP_CipherUpdate(evp, out, &outl, in, inl); /*  暗号、復号 */
        fwrite(out, 1, outl, outfp);
    }

    if(mode == DEC) /* 復号処理ならば認証用タグを設定 */
        EVP_CIPHER_CTX_ctrl(evp, EVP_CTRL_AEAD_SET_TAG, 16, tagIn;

    if(EVP_CipherFinal(evp, out, &outl) != SSL_SUCCESS) /* パディング処理 */
        エラー処理
    else
        fwrite(out, 1, outl, outfp);

    if(mode == ENC) /* 暗号処理ならばタグを得る */
        EVP_CIPHER_CTX_ctrl(evp EVP_CTRL_AEAD_GET_TAG, 16, tagOut);

    EVP_CIPHER_CTX_free(evp);
```

4) EVP関数の命名規則

EVP関数では、共通鍵の暗号または復号処理の方向がプログラミング時に静的に決定している場合のための関数と実行時に動的に決めることができる関数の二つの系列の関数が用意されています。静的な場合は関数名に"Encrypt"または"Decrypt"の命名が含まれていて、処理の方向を表します。動的な場合は関数名には"Cipher"の命名がされ、EVP_CipherInitの初期設定時に処理の方向を指定します。次の表に、これらの共通鍵処理のための関数名をまとめます。

機能	暗号化	復号	動的指定
コンテクスト確保	EVP_CIPHER_CTX_new	EVP_CIPHER_CTX_new	EVP_CIPHER_CTX_new

機能	暗号化	復号	動的指定
初期設定	EVP_EncryptInit	EVP_DecryptInit	EVP_CipherInit
暗号/復号	EVP_EncryptUpdate	EVP_DecryptUpdate	EVP_CipherUpdate
終了処理	EVP_EncryptFinal	EVP_DecryptFinal	EVP_CipherFinal
コンテクスト解放	EVP_CIPHER_CTX_free	EVP_CIPHER_CTX_free	EVP_CIPHER_CTX_free

5) パディング処理

EVP関数では、ブロック型暗号のためのパディング処理を自動的に行います。パディングスキームはPKCSです。このため、暗号化処理の場合は処理結果は入力データのサイズに比べてブロックサイズの整数倍にアラインされる分だけ大きくなる点に注意が必要です。入力データがブロックサイズの整数倍の場合にもパディング用に 1 ブロック分の出力データが付加されます。一方、復号の際はパディングの内容が解消され、復号された本来の出力データのみとなります。パディングを含んだ暗号、復号処理の出力データサイズは"Final"関数のアーギュメントに返却されます。

パディングスキームにはPKCS#7に規定されるスキームが使用されます　(3.4 共通鍵暗号 4)パディングスキーム参照)。

6) 暗号アルゴリズム、利用モード

EVPでは各種の暗号アルゴリズム、利用モードなどの処理パラメータの設定を"Init"関数で行うことで、処理を統一的に取り扱うことができます。以下に"Init"にて指定できる主な暗号スイートをまとめます。

シンボル	アルゴリズム	ブロック長	鍵長	利用モード
EVP_aes_xxx_cbc	AES	128	xxx: 128, 192, 256	CBC
EVP_aes_xxx_cfb1	AES	128	xxx: 128, 192, 256	CFB1
EVP_aes_xxx_cfb8	AES	128	xxx: 128, 192, 256	CFB8
EVP_aes_xxx_cfb128	AES	128	xxx: 128, 192, 256	CFB128
EVP_aes_xxx_ofb	AES	128	xxx: 128, 192, 256	OFB
EVP_aes_xxx_xts	AES	128	xxx: 128, 256	XTS
EVP_aes_xxx_gcm	AES	128	xxx: 128, 192, 256	GCM
EVP_aes_xxx_ecb	AES	128	xxx: 128, 192, 256	ECB
EVP_aes_xxx_ctr	AES	128	xxx: 128, 192, 256	CTR
EVP_des_cbc	DES	64	56	CBC
EVP_des_ecb	DES	64	56	ECB
EVP_des_ede3_cbc	DES-EDE3	64	168	CBC

シンボル	アルゴリズム	ブロック長	鍵長	利用モード
EVP_des_ede3_ecb	DES-EDE3	64	168	ECB
EVP_idea_cbc	IDEA	64	128	CBC
EVP_rc4	RC4			

7) その他のAPI

以下に共通鍵暗号の処理に関連する主なEVP関数をまとめます。

関数名	機能
EVP_CIPHER_CTX_iv_length, EVP_CIPHER_iv_length	IVサイズを取得
EVP_CIPHER_CTX_key_length, EVP_CIPHER_key_length	鍵サイズを取得
EVP_CIPHER_CTX_mode, EVP_CIPHER_mode	暗号、復号のモードを取得
EVP_CIPHER_CTX_block_size, EVP_CIPHER_block_size	ブロックサイズを取得
EVP_CIPHER_CTX_flags, EVP_CIPHER_flags	フラグを取得
EVP_CIPHER_CTX_cipher	アルゴリズムを取得
EVP_CIPHER_CTX_set_key_length	鍵サイズを設定
EVP_CIPHER_CTX_set_iv	IVサイズを設定
EVP_CIPHER_CTX_set_padding	パディングを設定
EVP_CIPHER_CTX_set_flags	フラグを設定
EVP_CIPHER_CTX_clear_flags	フラグをクリア
EVP_CIPHER_CTX_reset	コンテクストをリセット (後方互換：EVP_CIPHER_CTX_FREEで不要に)
EVP_CIPHER_CTX_cleanup	コンテクストをクリーンアップ (後方互換：EVP_CIPHER_CTX_FREEで不要に)

7.5 公開鍵暗号

7.5.1 RSA鍵ペア生成

1) 概要

このサンプルプログラムは一対のRSA秘密鍵と公開鍵を生成します。RSA_generate_keyにて内部形式(RSA構造体)で鍵を生成します。これを、i2d_RSAPrivateKey, i2d_RSAPublicKeyにて、DER形式のプライベート鍵、公開鍵に変換し、それぞれのファイルに出力します。

2) コマンド形式

サンプルプログラムでは以下のアーギュメントを指定します。

- 第一アーギュメント：プライベート鍵のファイル名
- 第二アーギュメント：公開鍵のファイル名

3) コマンド例

```
$ ./genrsa pri.der pub.der
```

4) ソースコードの概要

```
    rsa = RSA_generate_key(RSA_SIZE, RSA_E, NULL, NULL);
    pri_sz = i2d_RSAPrivateKey(rsa, &pri);
    pub_sz = i2d_RSAPublicKey(rsa, &pub);
    fwrite(pub, 1, pub_sz, fpPub);
    fwrite(pri, 1, pri_sz, fpPri);
```

7.5.2 RSA暗号化、復号

```
暗号化
fread(stdin);
EVP_PKEY_CTX_set_rsa_padding();
EVP_PKEY_encrypt();
fwrite(sdiout);

復号
fread(stdin);
EVP_PKEY_CTX_set_rsa_padding();
EVP_PKEY_decrypt();
fwrite(sdiout);
```

7.5.3 RSA署名/検証

1) 全体の流れ

署名:

処理のはじめに"EVP_MD_CTX_new"関数により処理コンテクストを管理するための管理ブロックを確保します。次に"EVP_DigestSignInit"関数により初期設定関数で確保したコンテクストに対して鍵、ハッシュアルゴリズム種別などのパラメータを設定します。

"EVP_DigestSignUpdate"関数によって対象メッセージのダイジェストを求めます。メモリーサイズの制限が許す場合は対象メッセージ全体を一括して"EVP_DigestSignUpdate"関数に渡すことができますが、制限がある場合は適当な大きさに区切って"EVP_DigestSignUpdate"関数を複数回呼び出すこともできます。メッセージをすべて読み込んだら、"EVP_DigestSignFinal"関数により求めたダイジェスト値と署名鍵から署名値を求めます。

最後に終了後管理ブロックを解放します。

検証:

処理のはじめに"EVP_MD_CTX_new"関数により処理コンテクストを管理するための管理ブロックを確保します。次に"EVP_DigestVerifyInit"関数により初期設定関数で確保したコンテクストに対して鍵、ハッシュアルゴリズム種別などのパラメータを設定します。

"EVP_DigestVerifyUpdate"関数によって対象メッセージのダイジェストを求めます。メモリーサイズの制限が許す場合は対象メッセージ全体を一括して"EVP_DigestVerifyUpdate"関数に渡すことができますが、制限がある場合は適当な大きさに区切って"EVP_DigestVerifyUpdate"関数を複数回呼び出すこともできます。メッセージをすべて読み込んだら、"EVP_DigestVerifyFinal"関数により署名値を検証します。

最後に終了後管理ブロックを解放します。

2) サンプルプログラム

以下にEVP関数を使用したRSA署名と検証のサンプルプログラムを示します。このプログラムでは次のコマンドアーギュメントを受付ます。

署名：rsasig

コマンドアーギュメント：

- 入力ファイル：DER形式の署名鍵ファイル
- 出力ファイル：署名値を出力します。省略した場合、標準出力に出力します。
- 標準入力：　署名対象メッセージを入力します

```
    /* 署名鍵の読み込み */
    key_sz = fread(in, 1, size, infp);
    pkey = d2i_PrivateKey(EVP_PKEY_RSA, NULL, &inp, key_sz);

    /* 管理ブロックの準備 */
    md = EVP_MD_CTX_new();
    EVP_DigestSignInit(md, NULL, HASH, NULL, pkey);

    /* メッセージを読み込みダイジェストを求める */
    for (; size > 0; size -= BUFF_SIZE) {
        inl = fread(msg, 1, BUFF_SIZE, stdin);
        EVP_DigestSignUpdate(md, msg, inl);
    }

    /* 署名生成 */
    EVP_DigestSignFinal(md, sig, &sig_sz);
    fwrite(sig, 1, sig_sz, outfp) != sig_sz);
```

検証：rsaver

コマンドアーギュメント：

- 入力ファイル1：DER形式の検証鍵ファイル
- 入力ファイル2：署名値が格納されたファイル
- 標準入力：　署名対象メッセージを入力します

```
検証
/* 検証鍵と署名の読み込み */
key_sz = fread(pubkey, 1, KEY_SIZE, infp);
sig_sz = fread(sig, 1, SIG_SIZE, fp2);
pkey = d2i_PublicKey(EVP_PKEY_RSA, NULL, &p, key_sz);

/* 管理ブロックの確保、設定 */
md = EVP_MD_CTX_new());
EVP_DigestVerifyInit(md, NULL, HASH, NULL, pkey) != SSL_SUCCESS) {
    fprintf(stderr, "EVP_DigestVerifyInit\n");
    goto cleanup;
}

/* メッセージを読み込みダイジェストを求める */
for (; size > 0; size -= BUFF_SIZE) {
    inl = fread(msg, 1, BUFF_SIZE, stdin)) < 0);
    EVP_DigestVerifyUpdate(md, msg, inl);

/* 署名の検証 */
EVP_DigestVerifyFinal(md, sig, sig_sz) == SSL_SUCCESS)
    printf("Signature Verified\n");
else
    printf("Invalid Signature\n");
```

7.7 CSR

1. 作成と署名

   name = X509_NAME_new(); 509_NAME_add_entry_by_txt(name, "commonName", MBSTRING_UTF8, (byte*)"wolfssl.com", 11, 0, 1); X509_NAME_add_entry_by_txt(name, "emailAddress", MBSTRING_UTF8, (byte*)"support@wolfssl.com", 19, -1, 1);

   d2i_PrivateKey(EVP_PKEY_RSA, NULL, &rsaPriv, (long)sizeof_client_key_der_2048); pub = d2i_PUBKEY(NULL, &rsaPub, (long)sizeof_client_keypub_der_2048); eq = X509_REQ_new();

   X509_REQ_set_subject_name(req, name); X509_REQ_set_pubkey(req, pub); X509_REQ_sign(req, priv, EVP_sha256()); i2d_X509_REQ(req, &der), 643); XFREE(der, NULL, DYNAMIC_TYPE_OPENSSL); der = NULL;

   mctx = EVP_MD_CTX_new(); EVP_DigestSignInit(mctx, &pkctx, EVP_sha256(), NULL, priv); X509_REQ_sign_ctx(req, mctx);

EVP_MD_CTX_free(mctx); X509_REQ_free(NULL); X509_REQ_free(req); EVP_PKEY_free(pub); EVP_PKEY_free(priv);

2） 検証

```
bio = BIO_new_file(csrFile, "rb");
d2i_X509_REQ_bio(bio, &req);
pub_key = X509_REQ_get_pubkey(req);
X509_REQ_verify(req, pub_key);

X509_free(req);
BIO_free(bio);
EVP_PKEY_free(pub_key);
```