SSL for NORTi

User's Guide

2011 年 03 月版

MiSPO
株式会社ミスポ

目次

2006 年 8 月版で訂正された項目

ページ	更新内容
4	「1.2 特長」で RC4 を ARC4 に変更
5	「1.4 ファイル構成」で nonsslpub.c の説明を削除

2008 年 5 月版で訂正された項目

ページ	更新内容
12	「5.1 PEM file」で nonsslpub.c を sslcerts.c に変更

2009 年 11 月版で訂正された項目

ページ	更新内容
24	「ssl_sht_cep」で CLOSURE_ALERT メッセージを CLOSE_NOTIFY アラートに変更

2010 年 12 月版で訂正された項目

ページ	更新内容
4	「1.3 制限事項」に API の排他制御上の制限を追加
30	SSL クライアントでの証明書妥当性チェックの説明を追加

2011 年 03 月版で訂正された項目

ページ	更新内容
18	ssl_acp_cep API の戻り値の説明を改訂、エラー処理について説明を追加
19,20	ssl_con_cep API の戻り値の説明を改訂、エラー処理について説明を追加
22	ssl_snd_dat API の戻り値の説明を改訂
23	ssl_rcv_dat API の戻り値の説明を改訂
24	ssl_sht_cep API の戻り値の説明を改訂
25	ssl_cls_cep API の戻り値の説明を改訂、解説を改訂
26	ssl_rehandshake API の戻り値の説明を改訂
27	ssl_get_ssn および ssl_cert_clbk API の戻り値の説明を改訂
28	ssl_err API の解説を改訂

第　1　章　導入

1.1 はじめに

SSL for NORTi は NORTi TCP/IP のアプリケーション層で SSL(Secure Sockets Layer)/TLS(Transport Layer Security)の機能を実現します。本章では SSL for NORTi の使用方法について記述しています。TCP/IP の使用方法に関しましては NORTi ユーザーズガイド TCP/IP 編を参照してください。

1.2 特長

SSL for NORTi は SSL Version3.0 と TLS Version1.0 をサポートします。互換性のため、SSL 2.0 Client Hello message をサポートしています。また、SSL/TLS で確立したセッションパラメータをセッションキャッシュに保持し再開することで、多くのコネクションが迅速に同じコネクションを利用できます。SSL for NORTi では次の各アルゴリズムを使用します。

鍵交換アルゴリズム	RSA
暗号化アルゴリズム	NULL, ARC4, DES, TDES
ハッシュアルゴリズム	MD5, SHA1
証明書タイプ	X.509 v1, X.509 v2, X.509 v3

SSL for NORTi ではクライアント/サーバプログラムのサンプルを収録しています。またリトル/ビッグの両エンディアンで使用できます。どのモードを使用するかはコンパイルオプションで定義する必要があります。

1.3 制限事項

・既存のバージョンでは鍵交換の数と暗号化アルゴリズムが限定されています。
・クライアント認証は未サポートです。
・TLS1.0 の拡張機能は未サポートです。
・公開鍵証明書に署名するツールは含まれていません。
・暗/復号化の処理はライブラリとして収録されており、ソースコードは含まれていません。これらのライブラリをアプリケーションから直接使用できません。
・SSLのAPIでは排他を考慮していないため、同じSSL通信端点に対して複数のタスクから同時に操作することはできません。APIの実行終了を待ってから操作する必要があります。

1.4 ファイル構成

SSL for NORTi は次のファイルで構成されています。

ヘッダファイル

nonssl.h　SSL API ヘッダファイル(アプリケーションで include します)
このヘッダファイルはアプリケーションが使用する構造体や API のプロトタイプが定義されています。SSL の API を使用する全てアプリケーションで include してください。
nonsslp.h　SSL プロトコル仕様内部ヘッダ
このヘッダファイルは SSL 内部で使用している構造体や関数のプロトタイプが定義されています。アプリケーションで include する必要はありません。

ソースコード

nonsslrp.c　Record プロトコルのソースコード
nonsslhp.c　Handshake プロトコルのソースコード
nonsslacc.c　ChangeCipherSpec プロトコルと Alert プロトコルが含まれたソースコード
nonsslcpr.c　暗号アルゴリズムのラッパーが含まれたソースコード
nontlscpr.c　TLS1.0 プロトコル暗号関数のソースコード
nonssl3cpr.c　SSL3.0 プロトコル暗号関数のソースコード
nonsslssn.c　セッションキャッシュと再開の処理が含まれたソースコード
nonssl.c　SSL API のソースコード

ライブラリ

nsslbxxx.lib　ビッグエンディアン用 SSL ライブラリ (xxx は CPU によって異なります)
nssllxxx.lib　リトルエンディアン用 SSL ライブラリ

サンプルプログラム

nonhttps.c　HTTP クライアントで SSL を使用したサンプルプログラム
nonhttpss.c　HTTP サーバで SSL を使用したサンプルプログラム

これらのファイルは NORTi¥NETSMP¥SRC にインストールされます。

暗号化アルゴリズムライブラリ

cryptxxxb.lib　ビッグエンディアン用暗号化ライブラリ(ソフトウェア)
cryptxxxl.lib　リトルエンディアン用暗号化ライブラリ(ソフトウェア)

これらのライブラリには暗号化アルゴリズムの処理が含まれています。SSL を使用する際にはアプリケーションとリンクしてください。SH7710 を使用する場合は内蔵の暗号アクセラレータが使用可能です。crypt7710b.lib または crypt7710l.lib をリンクしてください。

1.5 用語

公開鍵暗号

2 個の鍵を使用する暗号技術の 1 分野です。公開鍵で暗号化されたメッセージは、関連する秘密鍵でのみ復号することができます。逆に、秘密鍵により署名されたメッセージは、公開鍵を使用して検証することができます。

共通鍵パラメータ

共通鍵パラメータには次の項目が含まれています。

- ユーザーの共通鍵証明書
- 共通鍵
- 取得した証明書の有効化を確認するための認証局の証明書

このオブジェクトは SSL コネクション確立時に使用されます。そのため SSL コネクションが確立される前に、設定されている必要があります。

ハンドシェイク

トランザクションのパラメータを確立するために、クライアントとサーバの間で行われる初期ネゴシエーションです。

SSL セッション

SSL セションは、クライアントとサーバとの関連付けです。セッションはハンドシェイクプロトコルによって生成されます。セッションでは、1 つの暗号セキュリティパラメータセットを定義します。このパラメータは複数のコネクションにより共有することができます。セッションは、それぞれのコネクションにおいて新しいセキュリティパラメータをネゴシエーションします。

セッションの再開

SSL の完全なハンドシェイクは CPU 処理時間とやりとりに必要な往復の回数といった面で非常にコストがかかります。実行時のコストを減らすために SSL はセッション再開のメカニズムを備えています。ハンドシェイクで一番コストがかかるのはセキュリティパラメータの交換です。新しいセッションでは既存のセキュリティパラメータを使用するため、セッションの再開ではセキュリティパラメータの交換を省略します。

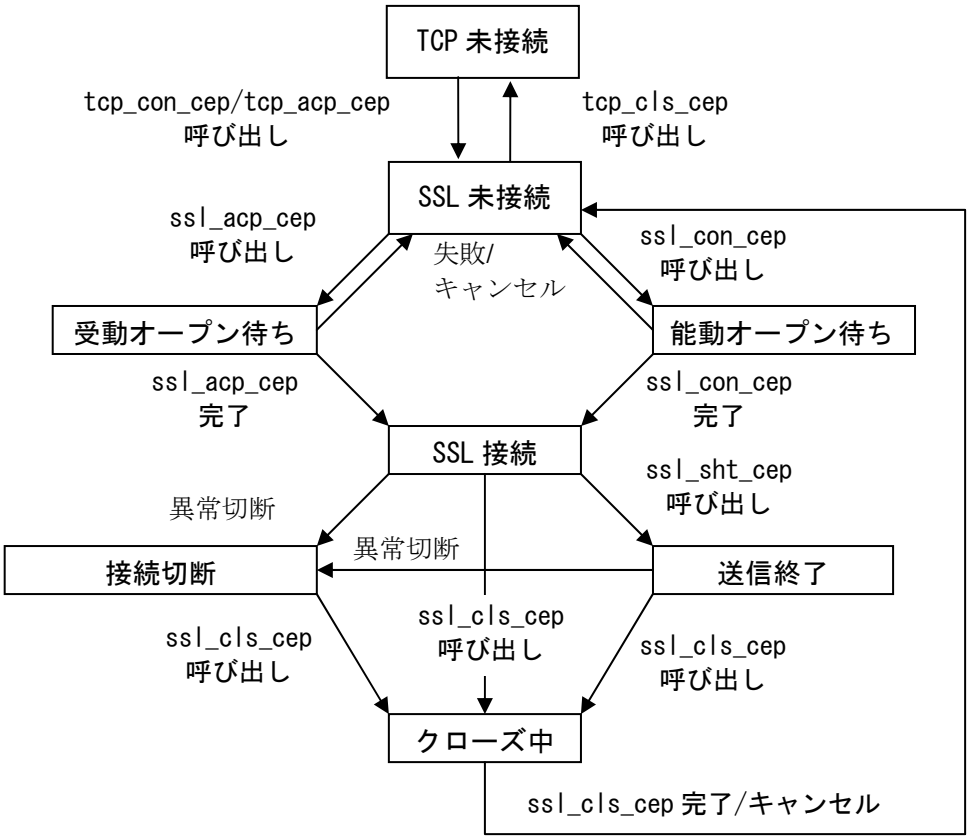
再ハンドシェイク

SSL の接続が行われるのは、最初のアプリケーションデータが書き込まれる前ですので、その接続にはどのようなセキュリティを適用すればよいのか、最初のハンドシェイクではわかりません。従って、再ハンドシェイクで、新しい情報を得る方法が有効です。

SSL 通信端点

SSL 通信端点は TCP 通信端点、共通鍵パラメータに関連付けられたオブジェクトで SSL 通信の I/F をアプリケーションに提供するオブジェクトです。SSL 通信端点はコネクション毎に分けて使用されます。

SSL 通信端点の状態

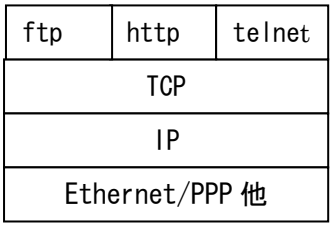


第　２　章　SSL/TLSプロトコルの構成

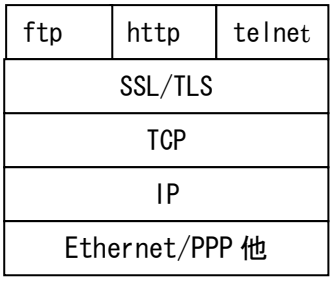
2.1 概要

SSL/TLS は TCP 層とアプリケーション間で動作します。SSL/TLS による通信が行われる前に TCP のコネクションが確立されている必要があります。通信を行う全ての API は SSL for NORTi の API を経由して TCP による通信が行われます。これによりリモートホストとの間で SSL/TLS によるセキュアな通信が実現できます。

SSL/TLS 未使用時の構成

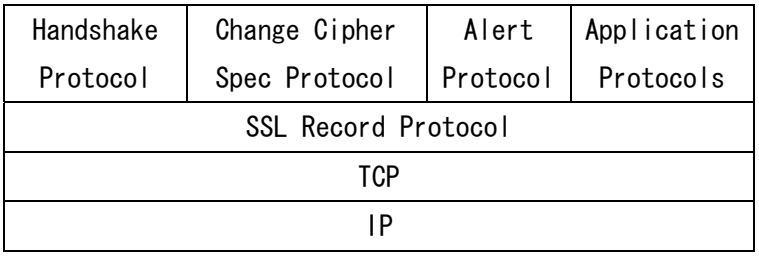


SSL/TLS 使用時の構成



2.2 階層構造

SSL for NORTi の詳細な階層構造は次のようになります。



2.3 使用するリソース

SSL for NORTi ではタスクを使用しません。SSL の処理はアプリケーションのコンテキストで実行されます。SSL for NORTi では次のリソースを使用します。

可変長メモリプール　× 1
ミューテックス　　× 1

アプリケーションタスクの優先度は IP 送受信タスクの優先度(デフォルトで 4)よりも低い優先度で使用する必要があります。

第　３　章　コンフィグレーション

3.1 マクロ定義

以下のマクロ定義は SSL for NORTi のコンフィグレーションで使用します。必要に応じて SSL for NORTi を使用する前に適切な値に変更してください。

#define SSL_SSN_MAX	(4)	Upper limit of Session Cache entries
#define SSL_VMPL_SIZ	(16 K)	可変長メモリプールのサイズ
#define SSL_VMPL_ID	(0)	内部で使用する可変長メモリプールの ID
#define SSL_MTX_ID	(0)	内部で使用するミューテックスの ID

() 内の値はデフォルト値です。ID=0 は内部で自動的に割り当てます。

3.2 コンパイル時に指定するマクロ定義

以下のマクロを0で定義することで不要なコードを省くことができます。このマクロは各SSLのファイルをコンパイルする際に定義してください

SSL_SERVER	(1)	サーバ動作
SSL_CLIENT	(1)	クライアント動作
SSL3_0	(1)	SSL3.0 モジュール
TLS1_0	(1)	TLS1.0 モジュール

() 内の値はデフォルト値です。

クライアント/サーバいずれかひとつは必ず必要です。

SSL3.0/TLS1.0いずれか1つは必要です。

第　4　章　共通定義

4.1 エラーコード

SSL for NORTi のエラーコードは各サービスコールの説明をご覧ください。SSL for NORTi ではエラーの詳細は ssl_err()サービスコールで取得することができます。

4.2 SSL通信端点の構造体

SSL 通信端点は SSL 内部で使用するためコントロールブロックです。SSL コネクションと送受信バッファの領域はアプリケーション側で確保して設定してください。それ以外のメンバは SSL 内部で使用されますので、値を変更しないでください。

```
typedef struct t_ssl_cep{
    T_SSL_CON *con;         SSL コネクションへのポインタ
    VP *rbuf;               SSL 受信バッファ領域の先頭アドレス
    INT rbufsz;             SSL 受信バッファ領域のサイズ
    VP *sbuf;               SSL 送信バッファ領域の先頭アドレス
    INT sbufsz;             SSL 送信バッファ領域のサイズ
    ID tcp_cepid;           TCP の通信端点 ID
    UB *rgetp;              SSL 受信バッファの GET ポインタ
    UB *rputp;              SSL 受信バッファの PUT ポインタ
    UB *sgetp;              SSL 送信バッファの GET ポインタ
    UB *sputp;              SSL 送信バッファの PUT ポインタ
    UB perr;                プロトコルエラーコードの詳細
} T_SSL_CEP;
```

SSL 受信バッファとサイズ

SSL 通信では受信したデータ処理を行うため、一時的なバッファエリアを使用します。バッファサイズは受信する実データより大きいサイズである必要があります。SSL レコードの最大長は 16Kbyte です。もし、バッファサイズが実際に受信したデータよりも小さい場合、ssl_rcv_dat は E_NOMEM でリターンします。

```
static T_SSL_CON ssl_con;
#define SSL_IRBUF_SIZE    16384       /* SSL internal Receive Buffer Size */
static UB ssl_irbuf[SSL_IRBUF_SIZE];  /* SSL internal Receive Buffer */
static T_SSL_CEP ssl_cep = {&ssl_con, ssl_irbuf, SSL_IRBUF_SIZE, ...};
```

SSL 送信バッファとサイズ

受信処理同様、送信処理でも SSL の処理を行うために一時的なバッファエリアを使用します。バッファサイズはアプリケーションで送信するデータサイズ+SSL ヘッダサイズ(SSL_TOT_HDRSZ)以上のサイズを確保する必要があります。バッファのサイズが小さい場合、ssl_snd_dat は E_NOMEM でリターンします。

```
static T_SSL_CON ssl_con;
#define APPL_SND_BUFSZ    4096
#define SSL_ISBUF_SIZE    APPL_SND_BUFSZ + SSL_TOT_HDRSZ
static UB ssl_isbuf[SSL_ISBUF_SIZE];  /* SSL internal Send Buffer */
static T_SSL_CEP ssl_cep = {&ssl_con, ssl_irbuf, SSL_IRBUF_SIZE, ssl_isbuf,
                                    SSL_ISBUF_SIZE..};
```

第　５　章　公開鍵証明書

5.1 PEM file

SSL for NORTi では公開鍵証明書を PEM 形式のデータとして smp¥[cpu]¥[board]¥sslcerts.c の trusted_ca に保持しています。証明書のファイルから PEM 形式のテキストをここにコピーしてください。
(行の末尾に¥をつけてください)
証明書のチェーンに含まれる個々の証明書は単一の配列に設定されます。例えば以下の例には 2 つの CA 証明書はが含まれています。

```
unsigned char trusted_ca[] =
"-----BEGIN CERTIFICATE-----¥
MIICXTCCAcagAwIBAgIBADANBgkqhkiG9w0BAQUFADBqMQswCQYDVQQGEwJKUDEL¥
MAkGA1UECBMCVE8xCzAJBgNVBAcTAktXMQswCQYDVQQKEwJNaTELMAkGA1UECxMC¥
VQQIEwJUTzELMAkGA1UEBxMCS1cxCzAJBgNVBAoTAk1pMQswCQYDVQQLEwJTVzEL¥
MAkGA1UEAxMCQ04xGjAYBgkqhkiG9w0BCQEWC2VtQG1pLmNvLmpwMIGfMA0GCSqG¥
SIb3DQEBAQUAA4GNADCBiQKBgQDlqyQhritwg2C+y2Ai3RtvM8txI1cuqzDdFLYG¥
p8tOMm5LZupPTjxhnCCTafZGu3PLCVkrqGU2i7iQZfB8C+0nmyk6psSuPsFjoB9V¥
PUJXXQIDAQABoxMwETAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBBQUAA4GB¥
347jTpiQjMSMrCMwCGW0DxRRk3QxYXa3q8TMBDYlm13SNLNtiK79ZXQPACVzSczp¥
K6VeSfo6x+iKSHdk/PV3H7OyzK4R1XdEorA/QFxejjAI¥
-----END CERTIFICATE-----¥
-----BEGIN CERTIFICATE-----¥
MIICWTCCAcKgAwIBAgIBAjANBgkqhkiG9w0BAQUFADBqMQswCQYDVQQGEwJKUDEL¥
U1cxCzAJBgNVBAMTAkNOMRowGAYJKoZIhvcNAQkBFgtlbUBtaS5jby5qcDAeFw0w¥
NTA5MDcxMTEyMTVaFw0wNjA5MDcxMTEyMTVaMGwxCzAJBgNVBAYTAkpQMQswCQYD¥
MAkGA1UEAxMCTVAxHDAaBgkqhkiG9w0BCQEWDW1wQHlhaGhvby5jb20wgZ8wDQYJ¥
KoZIhvcNAQEBBQADgY0AMIGJAoGBAKcTM3FbiQ4WQ3wHJpESZyRloY64/Y/9ym4v¥
LUw7O4eCUxQpy6yraK4jEePEfdRkTksCKeshJzJn1CPLS65d/DeIz+7WmnEajJ7P¥
vigylQpa4iZrz5uu2asYqbcw+5MYdNQhZ4amaBaVlKhKSzfFECJ5JfuMoj6Bjkgr¥
vJzTeWABF3uI31tZQEIme4UdqhoUK1HC6HVSQxD7sjcp3vVEipZP/YYLbvNvgFjb¥
OvmnTqIb3MaUYEXuks0ZDQX8ck+q0FKUWj1UWK0=¥
-----END CERTIFICATE-----";
```

第　６　章　サービスコール

6.1 サービスコール一覧

ssl_ini	SSL プロトコルの初期化
ssl_ext	SSL プロトコルの終了
ssl_read_certs	共通鍵パラメータの生成
ssl_free_certs	獲得した共通鍵パラメータの解放
ssl_acp_cep	接続要求待ち(受動オープン)
ssl_con_cep	接続要求(能動オープン)
ssl_snd_dat	データの送信
ssl_rcv_dat	データの受信
ssl_sht_cep	データ送信の終了
ssl_cls_cep	SSL コネクションのクローズ
ssl_rehandshake	セッションの再ハンドシェイクをクライアントに通知する
ssl_get_ssn	セッションパラメータの取得（セッション再開用）
ssl_cert_clbk	証明書受信時に呼び出されるコールバック関数の登録
ssl_err	最後に処理されたアラートメッセージを取得する

ssl_ini

［機　能］　SSL プロトコルの初期化

［形　式］　ER ssl_ini();

［戻り値］　E_OK　　正常終了
　　負の値　　OS リソースの生成に失敗

［解　説］　内部で使用するデータの初期化や OS リソースの生成を行います。このサービスコールはSSLの全てのAPIを使用する前にタスクコンテキストから呼び出してください。

ssl_ext

［機　能］　SSL プロトコルの終了

［形　式］　ER ssl_ext();

［戻り値］　E_OK　　正常終了

［解　説］　SSL プロトコルで使用している OS リソースを解放します。この API を呼び出した後で、再び SSL を使用する場合は ssl_ini を呼び出す必要があります。

ssl_read_certs

［機　能］　共通鍵パラメータの生成

［形　式］　ER ssl_read_certs(T_PUBKEY_PARAMS **certs, UB *in_cert, UB *in_privkey,
UB *priv_passwd, UB *in_trustedca);

certs	共通鍵パラメータのポインタを格納するバッファへのポインタ
in_cert	共通鍵証明書が格納されているバッファへのポインタ
in_privkey	秘密鍵が格納されているバッファへのポインタ
priv_passwd	秘密鍵のパスワードが格納されているバッファへのポインタ
in_trustedca	CA 証明書が格納されているバッファのポインタ

［戻り値］

E_OK	正常終了
E_NOMEM	共通鍵暗号のためのメモリ取得に失敗
E_PAR	パラメータエラー
E_OBJ	入力情報に問題が見つかった
E_SYS	内部処理エラー
E_NOSPT	証明書に未サポートの項目がある

［解　説］　このサービスコールは共通鍵パラメータ用のメモリを可変長メモリプールから割り当て、共通鍵パラメータの生成を行います。サーバ動作時にはユーザー自身の共通鍵証明書と秘密鍵を設定します。クライアント動作時に CA 証明書をサーバから取得した証明書とベリファイするために設定します。秘密鍵が暗号化されている場合、priv_passwd を使用して復号化されます。もし、秘密鍵が暗号化されていない場合、priv_passwd には NULL を設定してください。
クライアントで動作させる場合、in_cert、in_privkey は NULL を設定してください。もしサーバで動作させる場合、in_trustedca は NULL を設定してください。同じ共通鍵パラメータを複数のコネクションで使用できます。

［例　1］

```
/* SSL server and client functionality. Private key is DES encrypted */
static T_PUBKEY_PARAMS *certs;
TASK https_task(void)
{
                              :
    ercd = ssl_read_certs(&certs, mycert_pem, mypriv_key, pkey_passwd,
trusted_ca);
                              :
```

```
}
```

[例　2]

```
/* SSL Client only functionality */
static T_PUBKEY_PARAMS *certs;
TASK https_task(void)
{
                                :
    ercd = ssl_read_certs(&certs, NULL, NULL, NULL, trusted_ca);
                                :
}
```

ssl_free_certs

［機　能］　獲得した共通鍵パラメータの解放

［形　式］　ER ssl_free_certs(T_PUBKEY_PARAMS *certs);
　　certs　　共通鍵オブジェクトのバッファへのポインタ

［戻り値］　E_OK　　正常終了
　　その他　　内部エラー

［解　説］　このサービスコールは共通鍵オブジェクトのために確保されたメモリリソースを解放します。

ssl_acp_cep

［機　能］　接続要求待ち(受動オープン)

［形　式］　ER ssl_acp_cep(T_SSL_CEP *ssl_cep, ID tcp_cepid, T_PUBKEY_PARAMS *certs,
TMO tmout);

ssl_cep	SSL 通信端点へのポインタ
tcp_cepid	TCP 通信端点 ID
certs	ハンドシェイクで使用する共通鍵パラメータへのポインタ
tmout	タイムアウト指定

［戻り値］

E_OK	正常終了
E_PAR	不正なパラメータが指定された、予期しないメッセージの受信、または受信データの復号に失敗した
E_NOMEM	SSL ハンドシェイクを行うためのメモリが十分でない
E_OBJ	SSL 通信端点や TCP 通信端点が不正な状態
E_SYS	暗号化処理またはハンドシェイクメッセージ作成エラー
E_TMOUT	タイムアウト
その他	tcp_snd_dat、tcp_rcv_buf、または tcp_rel_buf のエラー

［解　説］　このサービスコールは SSL コネクションを生成し、SSL クライアントから ClientHelloRequest の受信を待ちます。ClientHelloRequest を受信した後、共通鍵パラメータを使用して SSL ハンドシェイクを実行します。このサービスコールを呼ぶ前に ssl_read_certs と tcp_acp_cep の呼び出しが完了している必要があります。

タイムアウトなし(tmout = TMO_FEVR)で本サービスコールを発行した場合、発行元のタスクは、SSL 接続が完了するまで待ち状態となります。 タイムアウトあり(tmout = 1～0x7fffffff)で本サービスコールを発行した場合、指定した時間が経過しても SSL 接続要求がない、または、SSL 接続が完了しなければ、E_TMOUT エラーが返ります。

本サービスコールがエラーを返す場合は、tcp_cls_cep で TCP 通信端点をクローズして、改めて tcp_acp_cep で TCP コネクションの確立を待つようにしてください。

ssl_con_cep

［機　能］　接続要求(能動オープン)

［形　式］　ER ssl_con_cep(T_SSL_CEP *ssl_cep, ID tcp_cepid, T_PUBKEY_PARAMS *certs,
T_SSN_PARAMS *ssn_param, UH cipherid, UB ver, TMO tmout);

ssl_cep	SSL 通信端点へのポインタ
tcp_cepid	TCP 通信端点 ID
certs	ハンドシェイクで使用する共通鍵パラメータへのポインタ
ssn_param	セッションパラメータへのポインタ（セッション再開時のみ）
ciphered	暗号アルゴリズム種別
ver	バージョンの選択(TLS または SSL)
tmout	タイムアウト指定

［戻り値］

E_OK	正常終了
E_PAR	不正なパラメータが指定された、予期しないメッセージの受信、または受信データの復号に失敗した
E_NOMEM	SSL ハンドシェイクを行うためのメモリが十分でない
E_OBJ	SSL 通信端点や TCP 通信端点が不正な状態
E_SYS	暗号化処理またはハンドシェイクメッセージ作成エラー
E_TMOUT	タイムアウト
その他	tcp_snd_dat、tcp_rcv_buf、または tcp_rel_buf のエラー

［解　説］　このサービスコールは SSL コネクションを生成し、SSL のハンドシェイクを開始するために ClientHelloRequest をサーバに送信します。ハンドシェイクでは共通鍵と証明書の取得や暗号化アルゴリズム、バージョンをサーバへ通知します。ハンドシェイクが完了した場合、またはエラーの場合にサービスコールから戻ります。このサービスコールを呼ぶ前に ssl_read_certs と tcp_con_cep の呼び出しが完了している必要があります。セッションの再開の時、ssn_param には前回接続時の情報が入っている必要があります。ssn_param が NULL の場合、SSL ハンドシェイクは新しいセッションを確立します。

暗号アルゴリズム種別には以下の値を設定できます。

0　　（サーバ側が選択します）

TLS_RSA_WITH_NULL_MD5

TLS_RSA_WITH_NULL_SHA

TLS_RSA_WITH_RC4_128_MD5

TLS_RSA_WITH_RC4_128_SHA

TLS_RSA_WITH_3DES_EDE_CBC_SHA

バージョンの選択には SSL_CON(SSL3.0)または TLS_CON(TLS_1.0)を設定できます。サーバ側が選択したバージョンをサポートしていない場合、SSL3.0 または TLS1.0 が自動的に選択されます。セッション再開時は暗号アルゴリズム種別とバージョンの選択は無視されます。

タイムアウトなし(tmout = TMO_FEVR)で本サービスコールを発行した場合、発行元のタスクは、SSL 接続が完了するまで待ち状態となります。 タイムアウトあり(tmout = 1～0x7fffffff)で本サービスコールを発行した場合、指定した時間が経過しても SSL 接続要求がない、または、SSL 接続が完了しなければ、E_TMOUT エラーが返ります。

本サービスコールがエラーを返す場合は、tcp_cls_cep で TCP 通信端点をクローズして、改めて tcp_con_cep で TCP コネクションを確立するようにしてください。

[例　1]

```
/* SSL connection,  Any one of supported Ciphers, Negotiate new session */
TASK https_task(void)
{
                              :
    ercd = ssl_con_cep(&https_cli_cep, cepid, certs, NULL, 0, SSL_CON,
8000/MSEC);
                              :
}
```

[例　2]

```
/* TLS connection,  TLS_RSA_WITH_3DES_EDE_CBC_SHA, Negotiate new session */
TASK https_task(void)
{
                              :
    ercd = ssl_con_cep(&https_cli_cep, cepid, certs, NULL,
 TLS_RSA_WITH_3DES_EDE_CBC_SHA, TLS_CON, 8000/MSEC);
                              :
}
```

[例　3]

```
/* Session Resumption */
static T_SSN_PARAMS ssn_params;
TASK https_task1(void)
{
                              :
    /* Get Session Parameters from any existing SSL connection */
```

```
    ercd = ssl_get_ssn(&https_exst_cep, &ssn_params);
                                :
}

TASK https_task2(void)
{
                                :
/* Session is resumed for a new SSL connection */
ercd = ssl_con_cep(&c_https_cli_cep, cepid, certs, &ssn_params,
 0, 0, 8000/MSEC);
                                :
}
```

ssl_snd_dat

［機　能］　データの送信

［形　式］　ER ssl_snd_dat(T_SSL_CEP *ssl_cep, UB *buf, UW len, TMO tmout);

ssl_cep	SSL 通信端点へのポインタ
buf	送信データへのポインタ
len	送信したいデータの長さ
tmout	タイムアウト指定

［戻り値］

正の値	正常終了(送信バッファに入れたデータの長さ)
E_NOMEM	データ送信を行うためのメモリが不十分
E_OBJ	SSL 通信端点や TCP 通信端点が不正な状態
E_SYS	暗号化または内部処理エラー
E_TMOUT	タイムアウト
その他	tcp_snd_dat のエラー

［解　説］　このサービスコールでは送信パケットにMAC(Message Authentication Code : メッセージ認証コード)とレコードプロトコルヘッダを追加し、データを暗号化してtcp_snd_datでデータを送信バッファにコピーします。
送信バッファに空きが無い場合、空きが生じるまで、発行元のタスクは待ち状態となります。タイムアウトなし(tmout = TMO_FEVR)で本サービスコールを発行した場合、発行元のタスクは、送信バッファへのコピーが完了するまで待ち状態となります。 タイムアウトあり(tmout = 1～0x7fffffff)で本サービスコールを発行した場合、指定した時間が経過しても、送信バッファに空きが生じなければ、E_TMOUTエラーが返ります。送信したいデータの長さと戻り値は必ずしも同じになりません。指定したデータサイズよりも空いているバッファサイズが小さい場合は空いているバッファサイズ分をコピーしてサービスコールから戻ります。

ssl_rcv_dat

［機　能］　データの受信

［形　式］　ER ssl_rcv_dat(T_SSL_CEP *ssl_cep, UB *buf, UW len, TMO tmout);

ssl_cep	SSL通信端点へのポインタ
buf	受信データを入れる領域へのポインタ
len	受信したいデータの長さ
tmout	タイムアウト指定

［戻り値］

正の値	正常終了(取り出したデータの長さ)
0	データ終結(接続が正常切断された)
E_NOMEM	データを受信するためのメモリが足りない
E_OBJ	SSL 通信端点や TCP 通信端点が不正な状態
E_SYS	内部処理エラー
E_PAR	予期しないメッセージの受信、または受信データの復号が失敗
E_TMOUT	タイムアウト
その他	tcp_rcv_buf または tcp_rel_buf のエラー

［解　説］　ssl_cep によって指定された SSL 通信端点からデータを読み出します。受信バッファに入ったデータを、buf で指し示される領域へコピーした時点で、このサービスコールからリターンします。受信バッファに入っているデータ長が受信しようとしたデータ長 len よりも短い場合、受信バッファが空になるまでデータを取り出し、取り出したデータの長さを戻り値として返します。 受信バッファが空の場合には、データを受信するまで、このサービスコール発行元のタスクは待ち状態となります。

タイムアウトなし(tmout = TMO_FEVR)で本サービスコールを発行した場合、発行元のタスクは、データのコピーが完了するまで待ち状態となります。 タイムアウトあり(tmout = 1～0x7fffffff)で本サービスコールを発行した場合、指定した時間が経過しても、データを受信しなければ、E_TMOUT エラーが返ります。

相手側から接続が正常切断され、受信バッファにデータがなくなると、本サービスコールから 0 が返ります。

ssl_sht_cep

［機　能］　データ送信の終了

［形　式］　ER ssl_sht_cep(T_SSL_CEP *ssl_cep, TMO tmout);

ssl_cep	SSL 通信端点へのポインタ
tmout	タイムアウト指定

［戻り値］

E_OK	正常終了
E_NOMEM	アラートメッセージを送信するためのメモリが足りない
E_OBJ	SSL 通信端点や TCP 通信端点が不正な状態
E_SYS	暗号化または内部処理エラー
E_TMOUT	タイムアウト
E_PAR	パラメータエラー(tmout に TMO_POL または TMO_NBLK を指定した)
E_CLS	SSL 通信端点が未接続状態
その他	tcp_snd_dat のエラー

［解　説］　このサービスコールは送信バッファからデータが送信された後で、リモートホストに CLOSE_NOTIFY アラートを送り、データ送信が終了したことを通知します。

タイムアウトなし(tmout = TMO_FEVR)で本サービスコールを発行した場合、発行元のタスクは、アラートメッセージの送信バッファへのコピーが完了するまで待ち状態となります。 タイムアウトあり(tmout = 1～0x7fffffff)で本サービスコールを発行した場合、指定した時間が経過しても、コピーが完了しなければ、E_TMOUT エラーが返ります。

ssl_cls_cep

［機　能］　SSL コネクションのクローズ

［形　式］　ER ssl_cls_cep(T_SSL_CEP *ssl_cep, TMO tmout);
ssl_cep　　SSL 通信端点へのポインタ
tmout　　タイムアウト指定

［戻り値］　E_OK　　正常終了
E_TMOUT　　タイムアウト
E_PAR　　パラメータエラー(tmout に TMO_POL または TMO_NBLK を指定した)
E_CLS　　SSL 通信端点が未接続状態

［解　説］　このサービスコールではCLOSE_NOTIFYアラートを送受信します。送信データがバッファにある場合は、データが送信されてからCLOSE_NOTIFYを送信します。CLOSE_NOTIFYを送信した後、リモートホストからのCLOSE_NOTIFYを待ちます。リモートホストのデータ送信が完了していない場合はデータ送信が完了しCLOSE_NOTIFYを受信するまで待ちます。そのときに受信したデータは破棄されます。このサービスコールによってセッションキャッシュが更新され、SSLコネクションで取得されたメモリブロックが解放されます。
タイムアウトなし(tmout = TMO_FEVR)で本サービスコールを発行した場合、発行元のタスクは、切断が完了するまで待ち状態となります。 タイムアウトあり(tmout = 1～0x7fffffff)で本サービスコールを発行した場合、指定した時間が経過しても、切断が完了しなければ、E_TMOUT エラーが返ります。
SSL 接続中にサービスコール(ssl_snd_dat、ssl_rcv_dat、ssl_sht_cep、または ssl_rehandshake)でエラーが起こった場合、または相手から SSL コネクションが切断された場合は、このサービスコールにより SSL コネクションのクローズ処理を行う必要があります。SSL コネクションをクローズした後、tcp_cls_cep で TCP 通信端点をクローズしてください。

ssl_rehandshake

［機　能］　セッションの再ハンドシェイクをクライアントに通知する

［形　式］　ER ssl_rehandshake(T_SSL_CEP *ssl_cep, TMO tmout);

ssl_cep	SSL 通信端点へのポインタ
tmout	タイムアウト指定

［戻り値］

E_OK	正常終了
E_NOMEM	HelloRequest メッセージ の処理を行うためのメモリが足りない
E_OBJ	SSL 通信端点が未接続
E_SYS	暗号化または内部処理エラー
E_TMOUT	タイムアウト
E_PAR	パラメータエラー(tmout に TMO_POL または TMO_NBLK を指定した)
その他	tcp_snd_dat のエラー

［解　説］　このサービスコールはサーバ動作時のみ有効です。SSL サーバは新しいコネクションを確立するためにクライアントにセッションの再ネゴシエーションを通知します。クライアントがセッションの再ネゴシエーションを望まない場合、このメッセージはクライアントによって無視される可能性があります。
タイムアウトなし(tmout = TMO_FEVR)で本サービスコールを発行した場合、発行元のタスクは、HelloRequest の送信バッファへのコピーが完了するまで待ち状態となります。 タイムアウトあり(tmout = 1～0x7fffffff)で本サービスコールを発行した場合、指定した時間が経過しても、コピーが完了しなければ、E_TMOUT エラーが返ります。

ssl_get_ssn

［機　能］　セッションパラメータの取得（セッション再開用）

［形　式］　ER ssl_get_ssn(T_SSL_CEP *ssl_cep, T_SSN_PARAMS *ssn_params);
ssl_cep　　SSL 通信端点へのポインタ
ssn_params　　SSL セッションパラメータへのポインタ

［戻り値］　E_OK　　正常終了
E_PAR　　ssl_cep または ssn_params が NULL
E_OBJ　　ハンドシェイクが未完了で、セッションが不正な状態

［解　説］　このサービスコールはセッションの再開時に使用するセッションパラメータを取得します。

ssl_cert_clbk

［機　能］　証明書受信時に呼び出されるコールバック関数の登録

［形　式］　ER ssl_cert_clbk(T_SSL_CEP *ssl_cep, ER (*cert_validator)(T_X509 *t));
ssl_cep　　SSL 通信端点へのポインタ
cert_validator　証明書受信時に呼び出されるコールバック関数のポインタ

［戻り値］　E_OK　　正常終了
E_PAR　　ssl_cep が NULL、または ssl_cep で指定した SSL 通信端点が未初期化

［解　説］　このサービスコールはサーバから証明書が発行されたときに呼び出されるコールバック関数を登録します。ユーザーアプリケーションはこのコールバック関数を使用して証明書が適切かどうかを確認することができます。証明書が適切な場合、コールバックから 0 でリターンしてください。証明書に問題がある場合はコールバックからマイナス値でリターンしてください。その場合、コネクションは切断されます。証明書妥当性チェックについての詳細な内容は「第７章 SSL クライアント 証明書妥当性チェック」を参照してください。

ssl_err

［機　能］　最後に処理されたアラートメッセージを取得する

［形　式］　ER ssl_err(T_SSL_CEP *ssl_cep);
ssl_cep　　SSL 通信端点へのポインタ

［戻り値］　アラート

［解　説］　アラートプロトコルメッセージのアラートディスクリプションが返ります。SSL 通信でエラーが起こる場合は、相手にアラートメッセージが送信され、SSL コネクションが切断されます。このサービスコールにより、受信または送信したアラートメッセージを取得することができます。アラートメッセージを送受信していない状態でサービスコールがエラーを返している場合、パラメータエラーなど、SSL 内部でエラーが起きていることが考えられます。

アラート記述	送信したアラート	受信したアラート
No Alert has been sent or received	255	255
SSLAP_CLOSE_NOTIFY	0	128
SSLAP_UNEXPECT_MSG	10	138
SSLAP_BAD_REC_MAC	20	148
SSLAP_DECRYPT_FAIL	21	149
SSLAP_REC_OVERFLOW	22	150
SSLAP_DCOMPRS_FAIL	30	158
SSLAP_HANDSK_FAIL	40	168
SSLAP_NO_CERT	41	169
SSLAP_BAD_CERT	42	170
SSLAP_UNSUPORT_CERT	43	171
SSLAP_CERT_REVOKED	44	172
SSLAP_CERT_EXPIRED	45	173
SSLAP_CERT_UNKNOWN	46	174
SSLAP_ILLEGAL_PARAM	47	175
SSLAP_UNKKONW_CA	48	176
SSLAP_ACCESS_DENIED	49	177

SSLAP_DECODE_ERR	50	178
SSLAP_DECRYPT_ERR	51	179
SSLAP_EXPORT_RESTRICT	60	188
SSLAP_PROT_VERSION	70	198
SSLAP_INSUFICIENT_SEC	71	199
SSLAP_INTERNAL_ERR	80	208
SSLAP_USR_CANCELED	90	218
SSLAP_NO_RENEGOTIATE	100	328

第　7　章　SSLクライアント 証明書妥当性チェック

SSL クライアントは、ハンドシェイク時にサーバから受信した DER(Distinguished Encoding Rule)形式の証明書を構文解析して、検証を行ないます。そして、検証結果とともに、DER データバッファに保存されている以下の情報のポインタをコールバック関数に渡します。

- ✧ X509 のバージョン(Version)
- ✧ シリアルナンバー(Serial Number)
- ✧ 証明書の発行者(Issuer)
- ✧ 証明される対象(Subject)
- ✧ 有効期限(Validity period)

コールバック関数により、検証結果と上記の情報を確認して、コネクションを確立するかどうか決めることができます。この際、コールバック関数のリターン値によりコネクションの確立が決まります。「0」をリターンすれば、コネクションを確立します。負数(戻り<0)をリターンすれば、コネクションを切断します。コールバックを登録しない場合は証明書の検証結果に関わらず、コネクションを確立します。

7.1 T_X509 構造体

検証結果と証明書情報は、以下の T_X509 構造体へのポインタとしてコールバック関数に渡されます。サーバから証明書チェーンを受信する場合は、(T_X509->next)ポインタでリストが生成されます。リストの最初の証明書はサーバの証明書で、リストの次の証明書は「署名した証明書」となります。

```
/* X509 証明書情報*/
typedef struct t_x509 {
    T_PKINFO pkinfo;                /* 公開鍵と公開鍵に関する情報 */
    UB subj_hash[SHA1_DIGEST_LEN]; /* 証明される対象の情報のハッシュ */
    struct t_x509 *next;            /* 次の証明書へのポインタ */
    UB *version;                    /* X509 のバージョンへのポインタ */
    UB *s_no;                       /* シリアルナンバーへのポインタ */
    UW sno_len;                     /* シリアルナンバーの長さ */
    UW sig_algo;                    /* CA が利用する暗号化アルゴリズム */
    UB *issuer;                     /* 証明書の発行者情報へのポインタ */
    UB *validity;                   /* 有効期限へのポインタ */
```

```
    UB *subject;                    /* 証明される対象へのポインタ */
    UB cert_hash[SHA1_DIGEST_LEN]; /* 証明書のハッシュ */
    UB isuer_hash[SHA1_DIGEST_LEN];/* 証明書の発行者情報のハッシュ */
    UB *sig;                        /* デジタル署名へのポインタ */
    UW siglen;                      /* デジタル署名の長さ */
    BOOL valid;                     /* 有効性フラグ */
}T_X509;
```

7.2 証明書検証と検証結果のチェック

SSL クライアントはサーバから受信した証明書を認証局(CA)証明書によって検証し、証明書が有効の場合は T_X509->valid を(1)に設定し、無効の場合は(-1)に設定します。サーバから証明書チェーンを受信する場合には、各証明書をチェーンの次の「署名した証明書」によって検証し、チェーンの最後の証明書を CA 証明書によって検証します。

コールバック関数で、各証明書の妥当性を証明書の valid フラグによりそれぞれ確認できます。

例えば、以下のコールバック関数では、検証結果を参照して証明書が適切な場合、「0」を返してコネクションを確立します。証明書に問題がある場合、負数(-1)を返してコネクションを切断します。

```
static ER ssl_cer_valid(T_X509 *cert)  /* Callback function */
{
    while(cert != NULL){
        if(cert->valid < 0){
            puts("Received Certificate does not pass the Validation Checks");
            return -1;
        }
        cert = cert->next;
    }

    /* Accept the connection by returning Zero */
    return 0;
}
```

7.3 証明書情報の取得

T_X509 構造体の version、s_no、issuer、validity、subject、sig ポインタはそれぞれ証明書データでの「X509 のバージョン」、「シリアルナンバー」、「証明書の発行者」、「有効期限」、「証明される対象」、「デジタル署名」コンポーネントの位置を表します。「X509 のバージョン」の長さは 1 バイトです。sno_len、siglen はそれぞれ s_no、sig のデータの長さとなります。なお、issuer、validity、subject は DER 形式のため、x509_parse_dname 関数、x509_parse_validity 関数により解析する必要があります。

ただし、コールバック関数からリターンすると、ハンドシェイク時にサーバから受信した証明書データのバッファは解放されるため、上記のポインタはコールバック関数内でしか使用できません。コールバック関数の外で使用する場合は、このコンポーネントを別にコピーして使用してください。

7.3.1 API

x509_parse_dname	証明書の発行者と対象の情報を解析
x509_parse_validity	証明書の有効期限を解析

x509_parse_dname

［機　能］　X509 証明書の発行者と証明される対象のコンポーネントを解析します。

［形　式］　ER x509_parse_dname(UB *cert_der, T_DNAME *dname);

cert_der	DER 形式のデータのバッファポインタ
dname	T_DNAME 構造体のポインタ

［戻り値］

正の値	正常終了(処理された DER データの長さ)
E_OBJ	無効な DER データ

［解　説］　X509 証明書の発行者(issuer)と証明される対象(subject)を以下の形式で取得します。

```
/* Distinguished Name */
typedef struct t_dname {
    UB *org;        /* Pointer to Organization Name */
    UB *orgu;       /* Pointer to Organization Unit Name */
    UB *cn;         /* Pointer to Common Name (URL) */
    UW orglen;      /* Length of Org Name */
    UW orgulen;     /* Length of Org Unit Name */
    UW cnlen;       /* Length of Common Name (URL) */
}T_DNAME;
```

orglen、orgulen、cnlen はそれぞれ org、orgu、cn のデータの長さとなります。org、orgu、cn コンポーネントはハンドシェイク時にサーバから受信した証明書データのバッファ内を指すので、SSL コールバック関数の外では無効になります。このコンポーネントをコールバック関数の外で使用する場合は、コピーして使用してください。なお、x509_parse_dname は org、orgu、cn 以外の Country Name、State Name などのデータの取得には対応していません。

x509_parse_validity

［機　能］　X509 証明書の有効期限のコンポーネントを解析します。

［形　式］　ER x509_parse_validity(UB *cert_der, T_VTIME *validity);

cert_der	DER 形式のデータのバッファポインタ
validity	T_VTIME 構造体のポインタ

［戻り値］

正の値	正常終了(処理された DER データの長さ)
E_OBJ	無効な DER データ

［解　説］　T_X509 の証明書の有効期限(validity period)を以下の形式で取得します。

```
/* Validity Period */
typedef struct t_vtime {
    UB *start;      /* 有効期間の開始日時へのポインタ */
    UB *end;        /* 有効期間の終了日時へのポインタ */
    UW slen;        /* 有効期間の開始日時データの長さ */
    UW elen;        /* 有効期間の終了日時データの長さ */
} T_VTIME;
```

slen、elen はそれぞれ start、end のデータの長さとなります。start、end コンポーネントはハンドシェイク時にサーバから受信した証明書データのバッファ内を指すので、SSL コールバック関数の外では無効になります。このコンポーネントをコールバック関数の外で使用する場合は、コピーして使用してください。なお、start と end は「YYMMDDHHMMSSZ」(where ‘Z’ is the capital letter Z)形式の UTC Time データとなりますので、適当に変換して使用してください。

7.4 証明書情報の取得例

以下の print_x509_info 関数は、サーバ証明書の情報を取得してコンソールに表示しています。この例では証明書の有効期限や対象情報による有効性の確認はしていないので、必要によってアプリケーションで確認してください。

```
static ER ssl_cer_valid(T_X509 *cert)   /* Callback function */
{
    /* The first certificate of the chain is Server Certificate */
    T_X509 *server_cert = cert;

    while(cert != NULL){
        if(cert->valid < 0){
            puts("Received Certificate does not pass the Validation Checks");
            return -1;
        }
        cert = cert->next;
    }

    /* Print the Server Certificate information */
    print_x509_info(server_cert);

    /* Accept the connection by returning Zero */
    return 0;
}

ER print_x509_info(T_X509 *x509)
{
    static const UB ver = 0x01;
    T_DNAME name;
    T_VTIME validity;
    ER ercd;

    memset(&name, 0, sizeof(T_DNAME));
    memset(&validity, 0, sizeof(T_VTIME));

    print("¥r¥nVersion: ");
    if (x509->version != NULL)
        print_digit(x509->version, 1);
    else
        print("0x01"); /* No version field for Version 1 Certificates */

    print("¥r¥nSerial Number: ");
    print_digit(x509->s_no, x509->sno_len);

    ercd = x509_parse_dname(x509->issuer, &name);
    if(ercd < 0)
        return ercd;
    print("¥r¥nIssuer Details: ");
```

```
    print("¥r¥nOrganization Name: ");
    print_buf(name.org, name.orglen);
    print("¥r¥nOrganization Unit Name: ");
    print_buf(name.orgu, name.orgulen);
    print("¥r¥nCommon Name(URL): ");
    print_buf(name.cn, name.cnlen);
    print("¥r¥n");

    ercd = x509_parse_dname(x509->subject, &name);
    if(ercd < 0)
        return ercd;
    print("¥r¥nSubject Details: ");
    print("¥r¥nOrganization Name: ");
    print_buf(name.org, name.orglen);
    print("¥r¥nOrganization Unit Name: ");
    print_buf(name.orgu, name.orgulen);
    print("¥r¥nCommon Name(URL): ");
    print_buf(name.cn, name.cnlen);
    print("¥r¥n");

    print("¥r¥nValidity Period Details: ");
    ercd = x509_parse_validity(x509->validity, &validity);
    if(ercd < 0)
        return ercd;
    print("¥r¥nStart(YYMMDDHHMMSSZ): ");
    print_buf(validity.start, validity.slen);
    print("¥r¥nEnd(YYMMDDHHMMSSZ): ");
    print_buf(validity.end, validity.elen);
    print("¥r¥n");
    return E_OK;
}

void print_buf(UB *buf, int len)
{
    char tmp_buf[32];

    if ((buf == NULL) || (len == 0))
        return;

    memcpy(tmp_buf, buf, len);
    tmp_buf[len] = '¥0';
    print(tmp_buf);
}

void print_digit(UB *buf, UW len)
{
    static const char hexc[] = "0123456789ABCDEF";
    char tmp_buf[3];
    int i;
```

```
    if ((buf == NULL) || (len == 0))
        return;

    tmp_buf[2] = '¥0';
    print("0x");
    for (i = 0; i < len; i++) {
        tmp_buf[0] = hexc[(buf[i] >> 4) & 0x0f];
        tmp_buf[1] = hexc[buf[i] & 0x0f];
        print(tmp_buf);
    }
    return;
}
```

SSL for NORTi ユーザーズガイド

株式会社ミスポ http://www.mispo.co.jp
〒213-0002 神奈川県川崎市高津区二子 5-1-1 高津パークプラザ 3F
一般的なお問い合せ sales@mispo.co.jp
技術サポートご依頼 norti@mispo.co.jp