SSL for NORTi

ユーザーズガイド

2024 年 5 月版

株式会社ミスポ

2024 年 5 月版で改訂された項目

ページ	改訂内容
全体	"SSL"の表記を"SSL/TLS"に、暗号化ライブラリを暗号ライブラリに改め
6	対応アルゴリズムの一覧に鍵長を追記
7	サンプルプログラムの一覧に FTPS クライアントを追加
11,17	μITRON 仕様の用語に合わせて、"リソース"を"オブジェクト"に訂正
6～8,10	「1.2 特長」、「1.3 制限事項」、「1.4 ファイル構成」、「1.5 用語」、「2.1 概要」の説明を一部見直し
12	「3.1 マクロ定義」と「3.2 コンパイル時に定義するマクロ」を「3.1 サイズ等のコンフィグレーション」と「3.2 不要コードの削除」に改め、ライブラリの再生成が必要であることを追記
13～14	SSL 通信端点の説明やコード例を見直し
49	「第 8 章 FTPS クライアント」を追加
62	「第 9 章 HTTPS サーバ／クライアント」を追加

2021 年 8 月版で改訂された項目

19,20,28	暗号アルゴリズムのリストを、暗号強度の強い順に変更

2018 年 6 月版で改訂された項目

6	鍵交換アルゴリズムに DHE(Diffie-Hellman Ephemeral)を追加
12	可変長メモリプールサイズのマクロに、DHE 使用時の SSL_VMPL_SIZ_DHE を追加
12	コンパイル時に定義するマクロに、DHE 使用有無の SSL_DHE を追加
15	ハッシュアルゴリズム/MAC のサイズの表に SHA-224、SHA-384、SHA-512 を追加
17,25	DH パラメータ生成の ssl_set_dhparam 関数を追加
20,28	DHE 用の暗号アルゴリズムの指定(TLS_DHE_RSA_WITH_???_???_???_???)を追加

2017 年 11 月版で改訂された項目

19～21	ssl_set_opt のオプション名の誤り訂正(SSL_CIPHER_LIST/ SSL_SIG_ALG_LIST → SSL_OPT_CIPHER_LIST/SSL_OPT_SIG_ALG_LIST)
19,21	SNI 対応用の SSL_OPT_SERVERNAME オプションを追加
全体	細かい表現を修正

2017 年 8 月版で改訂された項目

ページ	改定内容
11	使用するオブジェクト数が定義されているマクロの一覧表を追加
31～34	省コピーAPI ssl_get_buf, ssl_snd_buf, ssl_rcv_buf, ssl_rel_buf を追加

2017 年 5 月版で改訂された項目

5,19	ハッシュアルゴリズムに SHA-224, SHA-348, SHA-512 を追加

2014 年 10 月版で改訂された項目

5,6	対応している TLS のバージョンに"～1.2"を追記
5,13	ハッシュアルゴリズムに SHA-256 を追加
10	SSL3_0 と TLS1_0 マクロを削除
15, 17～19	ssl_set_opt API を追加
25	暗号アルゴリズム種別に 0 を指定した場合の説明を修正
25	暗号アルゴリズム種別に SHA-256 関連を追加
25	バージョンの選択方法を変更
34	SSLAP_UNSUPPORTED_EXT を追加

2014 年 5 月版で改訂された項目

5	暗号化アルゴリズムから DES を削除
5	制限事項の排他制御に関する説明を改訂
6	ライブラリのファイル名を修正
9	使用するオブジェクトの説明を改訂
10	マクロ定義の説明を改訂
11	SSL 通信端点の構造体を改訂
12	SSL 受信バッファとサイズの説明を改訂
12	SSL 送信バッファとサイズの説明を改訂
13	SSL レコードのサイズの求め方を追加
24	ssl_snd_dat API の解説を改訂
25	ssl_rcv_dat API の解説を改訂
30	ssl_err API の解説を改訂

2012 年 2 月版で改訂された項目

ページ	改訂内容
4,20,21	AES に関する記述を追加
19	引数 cipherid を ciphered と誤っていたのを修正

2011 年 3 月版で改訂された項目

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

2010 年 12 月版で改訂された項目

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

2009 年 11 月版で改訂された項目

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

2008 年 5 月版で改訂された項目

12	「5.1 PEM file」で nonsslpub.c を sslcerts.c に変更

2006 年 8 月版で改訂された項目

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

目次

第 1 章 導入

1.1 はじめに

「SSL for NORTi」は NORTi の TCP/IP スタックのトランスポート層とアプリケーション層の間で SSL(Secure Sockets Layer)と TLS(Transport Layer Security)の機能を実現します。本書では、SSL/TLS の機能と使用方法についてのみ記述していますので、TCP/IP スタックの使用方法に関しましては「NORTi Version 4 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 Version 3.0 と TLS Version 1.0～1.2 をサポートしています。互換性のため、SSL 2.0 Client Hello message も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、複数のコネクションを用いて迅速に同じサーバと接続させることができます。

SSL for NORTi が対応しているアルゴリズムは、次のとおりです。

鍵交換アルゴリズム	**RSA(鍵長 2048 ビット), DHE(公開鍵長 2048/秘密鍵長 224 ビット)**
暗号化アルゴリズム	**NULL, ARC4, TDES, AES(鍵長 128/192/256 ビット)**
ハッシュアルゴリズム	**MD5, SHA-1, SHA-2(SHA-224, SHA-256, SHA-384, SHA-512)**
証明書タイプ	**X.509 v1, X.509 v2, X.509 v3**

1.3 制限事項

・ECDHE 等の上記の一覧にないアルゴリズムや TLS1.3 には未対応です。
・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。
・暗号化/復号の処理はライブラリとして収録されており、ソースコードは付属していません。暗号ライブラリをアプリケーションから直接使用したい場合は、ご相談ください。
・同じSSL通信端点に対するssl_snd_dat、ssl_rcv_dat、ssl_errを複数のタスクから同時に発行できますが、その他のAPIは、実行終了を待ってから発行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/TLS API のヘッダ
このヘッダファイルに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/TLS の API を使用する全てアプリケーションでインクルードしてください。

nonsslp.h　　SSL/TLS 内部定義ヘッダ
このヘッダファイルには SSL/TLS モジュール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インクルードする必要はありません。

nocrypt.h　　暗号ライブラリ関数のヘッダ
このヘッダファイルには SSL/TLS モジュール内部で使用している暗号化/復号の関連の定義がされています。アプリケーションでインクルードする必要はありません。

ソースファイル

nonssl.c　　SSL/TLS API のソース
nonsslrp.c　　Record プロトコルのソース
nonsslhp.c　　Handshake プロトコルのソース
nonsslacc.c　　ChangeCipherSpec プロトコルと Alert プロトコルのソース
nonsslcpr.c　　暗号アルゴリズムのラッパー関数のソース
nontlscpr.c　　TLS1.0～1.2 プロトコル暗号関数のソース
nonssl3cpr.c　　SSL3.0 プロトコル暗号関数のソース
nonsslssn.c　　セッションキャッシュと再開の処理のソース

SSL/TLS ライブラリ

nssl???b.lib　　ビッグエンディアン用 SSL/TLS ライブラリ
nssl???l.lib　　リトルエンディアン用 SSL/TLS ライブラリ
(???は CPU によって異なり、コンパイラによっては.lib 以外の拡張子もあります)

サンプルプログラム

nonftps.h　　FTPS クライアントのヘッダ
nonftps.c　　FTPS クライアントの API のソース
nonftpsc.c　　FTPS 対応 ftp コマンド実装例のソース
nonhttps.h　　HTTPS サーバ/クライアントのヘッダ
nonhttps.c　　HTTPS クライアント実装例のソース
nonhttpss.c　　HTTPS サーバ実装例ののソース

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

暗号ライブラリ

crypt???l.a　　リトルエンディアン用の暗号ライブラリ

crypt???b.a　　ビッグエンディアン用の暗号ライブラリ

暗号化/復号アルゴリズムの処理はこのライブラリに分離されていますので、SSL/TLS ライブラリと共にリンクしてください。ファイル名の???の部分は対応コアによって、拡張子は他には.lib などコンパイラによって異なります。

1.5 用語

公開鍵暗号

2 つの鍵を使用する暗号技術で、公開鍵で暗号化されたメッセージは、ペアとな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
- 共通鍵は SSL/TLS コネクションの確立時に使用されます。そのため、最初の接続の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/TLS セッション

SSL/TLS セッションは、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/TLS 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/TLS 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。

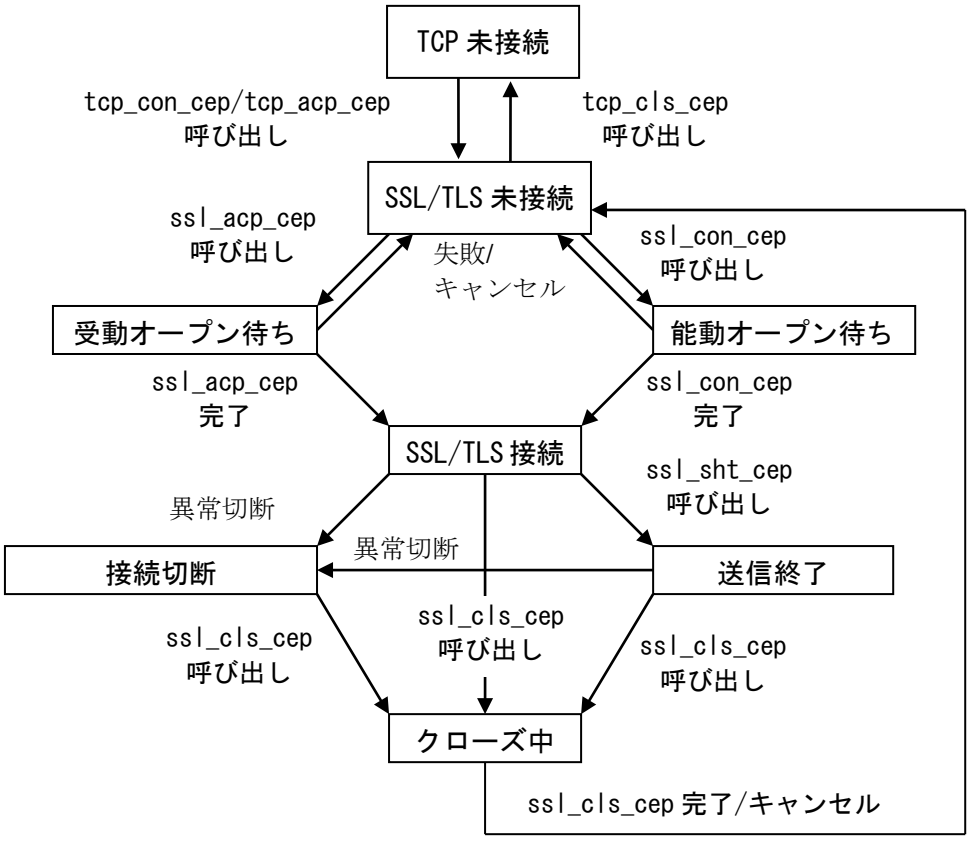
再ハンドシェイク

SSL/TLS 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

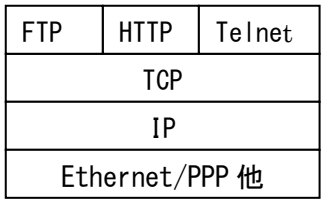


第 2 章 SSL/TLS プロトコルの構成

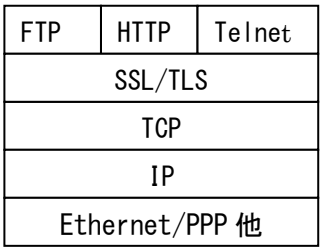
2.1 概要

SSL/TLS は TCP 層とアプリケーションとの間で動作します。NORTi の TCP/IP スタックで標準のサービスコール tcp_xxx_yyy()を、SSL for NORTi のサービスコール ssl_xxx_yyy()に置き換えることにより、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

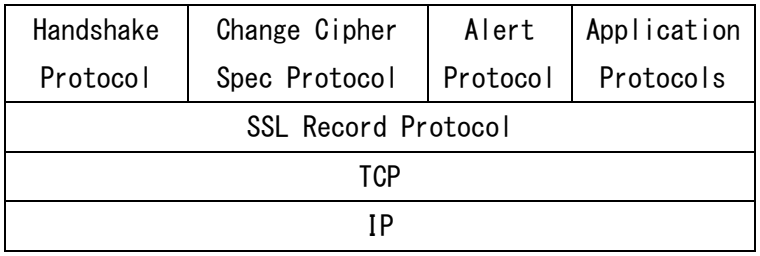


SSL/TLS 使用時の構成



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 使用するオブジェクト

SSL for NORTi では可変長メモリプール以外のカーネルのオブジェクトを使用しておらず、SSL/TLS の処理は、サービスコールを発行したタスクのコンテキストで実行されます。
SSL for NORTi で使用するオブジェクトの数は、次のマクロに定義されていますので、カーネルのコンフィグレーションで、「#define TSKID_MAX 12+(TCP_NTSK+SSL_NTSK)」ののように利用してください。

オブジェクト	マクロ名	値
タスク	SSL_NTSK	0
セマフォ	SSL_NSEM	0
イベントフラグ	SSL_NFLG	0
メールボックス	SSL_NMBX	0
メッセージバッファ	SSL_NMBF	0
ランデブ用ポート	SSL_NPOR	0
可変長メモリプール	SSL_NMPL	1
固定長メモリプール	SSL_NMPF	0
データキュー	SSL_NDTQ	0
ミューテックス	SSL_NMTX	0
割り込みサービスルーチン	SSL_NISR	0
周期ハンドラ	SSL_NCYC	0
アラームハンドラ	SSL_NALM	0

第 3 章 コンフィグレーション

3.1 サイズ等のコンフィグレーション

SSL for NORTi のコンフィグレーション用に以下のマクロが nonssl.h に定義されています。これらを変更する場合は、nonssl.h を編集して SSL/TLS ライブラリを再生成してください。

#define PRI_SSL	5	SSL/TLS 処理中のタスク優先度
#define SSL_SSN_MAX	4	セッションキャッシュの最大エントリ数
#define SSL_VMPL_SIZ	17K	可変長メモリプールのサイズ(DHE 未使用時)
#define SSL_VMPL_SIZ_DHE	21K	可変長メモリプールのサイズ(DHE 使用時)
#define SSL_VMPL_ID	0	内部で使用する可変長メモリプールの ID

PRI_SSL は、TCP/IP スタック内部の IP 送受信タスクの優先度(デフォルト 4)より低く(数値では大きく)してください。SSL_VMPL_ID が 0 の場合は、ID が自動的に割り当てられます。

3.2 不要コードの削除

nonssl.hには以下のマクロも定義されており、これらを0に変更することにより、使用しない機能のコードを省くことができます。

SSL_SERVER	1	サーバ動作
SSL_CLIENT	1	クライアント動作
SSL_DHE	1	DHE 使用

サーバ動作とクライアント動作の両方を0とすることはできません。

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 TCP/IP スタックと共通です。詳しくは、各サービスコールの戻り値の説明をご覧ください。直近のサービスコールのエラーコードは、ssl_err サービスコールで取得することもできます。

4.2 SSL 通信端点

SSL for NORTi では SSL/TLS 通信に必要な情報を、TCP 通信端点と関連付けた SSL 通信端点として、T_SSL_CON と T_SSL_CEP の 2 つの構造体で管理しています。
ユーザープログラムでは、これらの構造体とバッファの領域を下記の例のように、通信端点毎に確保してください。

```
#define RBUFSZ  ?????          SSL 受信バッファのサイズ（4 の倍数）
#define SBUFSZ  ?????          SSL 送信バッファのサイズ（  〃   ）
UW ssl_rbuf[RBUFSZ/4];         SSL 受信バッファ（4 バイト境界とするため UW で）
UW ssl_sbuf[SBUFSZ/4];         SSL 送信バッファ（          〃          ）
T_SSL_CON ssl_con;             SSL 接続制御ブロック
T_SSL_CEP ssl_cep = { &ssl_con, ssl_rbuf, RBUFSZ, ssl_sbuf, SBUFSZ };
```

T_SSL_CEP 構造体には多くのメンバが定義されていますが、ユーザープログラムで初期化が必要なのは上記のとおり先頭の 5 つのメンバで、次のようなコードで動的に設定することもできます。

```
ssl_cep.con    = &ssl_con;
ssl_cep.rbuf   = ssl_rbuf;
ssl_cep.rbufsz = RBUFSZ;
ssl_cep.sbuf   = ssl_sbuf;
ssl_cep.sbufsz = SBUFSZ;
```

なお、RBUFSZ、SBUFSZ、ssl_rbuf、ssl_sbuf、ssl_con、ssl_cep の各マクロ名や変数名は一例ですので、それぞれの通信端点に合わせた適切な名前にしてください。

SSL 受信バッファとサイズ

SSL/TLS 通信では受信した SSL レコードの処理を行うため、一時的にバッファを使用します。SSL レコードの最大長は 16,383 バイト(16KB－1)で、受信したレコード長がこのバッファサイズを越える場合、tcp_rcv_dat のように分割しての受信はできず、ssl_rcv_dat は E_NOMEM エラーでリターンします。

SSL 送信バッファとサイズ

SSL/TLS 通信の送信処理でも一時的にバッファを使用します。必要なバッファサイズは、SSL レコード長＋SSL_TOT_HDRSZ(ヘッダのサイズ 41 バイト)ですが、小さい場合、ssl_snd_dat はバッファに収まる分までを送信しますので、tcp_snd_dat のように繰り返し ssl_snd_dat を発行することで大きなデータも送信できます。

SSL レコードのフラグメントフィールドのサイズの求め方

アプリケーションデータに、MAC(Message Authentication Code：メッセージ認証コード)とパディングとパディング長フィールドのサイズを加えた値が、フラグメントフィールドのサイズになります。MAC のサイズは、ハッシュアルゴリズムにより決まります。パディングは、フラグメントフィールドのサイズがブロックサイズの倍数になるように付加されます。パディング長フィールドは 1 バイトです。ブロックサイズが 1 以下の場合は、パディングおよびパディング長フィールドは付加されません。

ハッシュアルゴリズム	MAC のサイズ
MD5	16
SHA-1	20
SHA-224	28
SHA-256	32
SHA-384	48
SHA-512	64

暗号化アルゴリズム	ブロックサイズ
NULL	0
ARC4	16
TDES	24
AES	16

〈例〉
アプリケーションデータのサイズ ： 16384 バイト
ハッシュアルゴリズム ： SHA-1
暗号化アルゴリズム ： AES

16384 ＋ 20 + 1 バイトが 16 の倍数になるように 11 バイトのパディングが付加され、フラグメントフィールドのサイズは 16416 バイトになる。

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

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_set_opt	オプションの設定
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_set_dmparam	DH パラメータの生成
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/TLS 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 オブジェクトの生成に失敗

［解　説］　内部で使用するデータの初期化や OS オブジェクトの生成を行います。このサービスコールは SSL の全てのサービスコール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 オブジェクトを解放します。このサービスコールを呼び出した後で、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_set_opt

［機　能］　オプションの設定

［形　式］　ER ssl_set_opt(INT optname, const VP optval, INT optlen);

optname	オプションの種類
optval	オプションの値が設定されているバッファへのポインタ
optlen	オプションの長さ

［戻り値］

E_OK	正常終了
E_PAR	パラメータエラー
E_NOSPT	未サポートの項目がある

［解　説］　ssl_ini()の直後に発行することで、optname に指定した以下のオプションの設定ができます。

SSL_OPT_CIPHER_LIST	暗号アルゴリズムのリスト（UH *型）
SSL_OPT_SIG_ALG_LIST	署名アルゴリズムのリスト（T_SSLHP_SIG_ALG *型）
SSL_OPT_SERVERNAME	サーバのドメイン名（const char *型）

動作に矛盾が生じる場合もあるので、ssl_set_opt()よりも前に他のサービスコールは発行しないでください。また、複数の ssl_set_opt()を発行する場合は、SSL_OPT_CIPHER_LIST を一番最初に指定してください。

SSL_OPT_CIPHER_LIST では、使用する暗号アルゴリズムのリストを設定します。クライアント動作では、ssl_con_cep()の cipherid に 0 を指定した場合だけ影響し、このリストの順にサーバに要求を出します。サーバ動作の場合は、クライアントからの要求がこのリストに含まれるアルゴリズムを選択します。この場合、このリストの順序は影響せず、クライアントからの要求の順序が優先されます。

optval で指定する配列には、以下のマクロを列記してください。

```
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_RC4_128_SHA
```

```
    TLS_RSA_WITH_RC4_128_MD5
    TLS_RSA_WITH_NULL_SHA256
    TLS_RSA_WITH_NULL_SHA
    TLS_RSA_WITH_NULL_MD5
    TLS_NULL_WITH_NULL_NULL(終端)
```

<使用例>

```
static const UH cipherlist[] = {
    TLS_RSA_WITH_AES_256_CBC_SHA256,
    TLS_RSA_WITH_AES_128_CBC_SHA256,
    TLS_NULL_WITH_NULL_NULL              /* 終端 */
};

func()
{
    ssl_set_opt(SSL_OPT_CIPHER_LIST, cipherlist, sizeof cipherlist);
}
```

SSL_OPT_CIPHER_LIST の設定を省略した場合は、下記のデフォルトのリストが適用され、使用するメモリプールのサイズが増える DHE に関するアルゴリズム(TLS_DHE_RSA_WITH_???_???_???_???)は含まれていません。

```
const UH cipherlist[] = {
    TLS_RSA_WITH_AES_256_CBC_SHA256,
    TLS_RSA_WITH_AES_128_CBC_SHA256,
    TLS_RSA_WITH_AES_256_CBC_SHA,
    TLS_RSA_WITH_AES_128_CBC_SHA,
    TLS_RSA_WITH_3DES_EDE_CBC_SHA,
    TLS_RSA_WITH_RC4_128_SHA,
    TLS_RSA_WITH_RC4_128_MD5,
    TLS_RSA_WITH_NULL_SHA256,
    TLS_RSA_WITH_NULL_SHA,
    TLS_RSA_WITH_NULL_MD5,
    TLS_NULL_WITH_NULL_NULL
};
```

SSL_OPT_SIG_ALG_LIST では、署名に使用するアルゴリズムのリストを設定します。このリストは TLS1.2 の場合だけ使用されます。クライアント動作では、このリストの順にサーバに要求します。サーバ動作の場合は、クライアントからの要求がこのリストに含まれるアルゴリズムのペアを選択します。この場合、このリストの順序は影響せず、クライアントからの要求の順序が優先されます。

optval で指定する配列には、以下の構造体メンバのマクロを列記してください。

```
typedef struct t_sslhp_sig_alg {
    UB hash_alg;     ハッシュのアルゴリズム
                     HASH_ALG_SHA512
```

```
                    HASH_ALG_SHA384
                    HASH_ALG_SHA256
                    HASH_ALG_SHA224
                    HASH_ALG_SHA1
                    HASH_ALG_MD5
                    HASH_ALG_NONE(終端)
    UB sig_alg;     署名のアルゴリズム
                    SIG_ALG_RSA
                    SIG_ALG_ANON(終端)
} T_SSLHP_SIG_ALG;
```

<使用例>

```
static const T_SSLHP_SIG_ALG sigalglist[] = {
    { HASH_ALG_SHA256, SIG_ALG_RSA  },
    { HASH_ALG_NONE,   SIG_ALG_ANON }   /* 終端 */
};

func()
{
    ssl_set_opt(SSL_OPT_SIG_ALG_LIST,sigalglist,sizeof sigalglist);
}
```

SSL_OPT_SIG_ALG_LIST の設定を省略した場合は、このデフォルトのリストが適用されます。

```
const T_SSLHP_SIG_ALG sigalglist[] = {
    { HASH_ALG_SHA512, SIG_ALG_RSA  },
    { HASH_ALG_SHA384, SIG_ALG_RSA  },
    { HASH_ALG_SHA256, SIG_ALG_RSA  },
    { HASH_ALG_SHA224, SIG_ALG_RSA  },
    { HASH_ALG_SHA1,   SIG_ALG_RSA  },
    { HASH_ALG_NONE,   SIG_ALG_ANON }
};
```

SSL_OPT_SERVERNAME は、SNI(Server Name Indication)対応のサーバにクライアントとして接続するためのオプションです。SNI では 1 台のサーバに複数のドメイン毎の証明書が設定されていますが、本オプションにより、その一つが選択されます。optval にはドメイン名の文字列へのポインタを指定してください。文字列の終端の NUL 文字('¥0')で長さは判断されますので、optlen の指定は不要です(0 を推奨)。指定を取り消し、サーバのドメイン名が無指定(SNI 非対応)のデフォルト状態に戻すには、optval に NULL を指定してください。

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確保し、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したメモリ領域を解放します。

ssl_set_dhparam

［機　能］　DH パラメータの生成

［形　式］　ER ssl_set_dhparam(T_PUBKEY_PARAMS *certs, UB *in_dhparam);
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_dhparam　DH パラメータが格納されているバッファへのポインタ

［戻り値］　E_OK　　正常終了
E_NOMEM　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　DH パラメータに未サポートの項目がある

［解　説］　このサービスコールは、DH パラメータ用の領域を可変長メモリプールから追加で確保し、そこに DH パラメータを設定します。先に ssl_read_certs()を発行して取得した certs を、このサービスコールに指定してください。ここで確保した領域は、ssl_free_certs()の発行により、ssl_read_certs()で確保した領域と一緒に解放されます。
なお、このサービスコールはサーバ動作時専用ですので、クライアント動作時には発行しないでください。クライアント動作時は、DH パラメータを通信相手のサーバから取得します。

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/TLS 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/TLS 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/TLS 接続要求がない、または、SSL/TLS 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）
cipherid	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/TLS 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。

暗号アルゴリズムの種別(cipherid)には以下の値を設定でき、無指定(0)の場合は、デフォルトのリスト、または、ssl_set_opt()で設定したリストが使われます。

TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_NULL_SHA256
TLS_RSA_WITH_NULL_SHA
TLS_RSA_WITH_NULL_MD5

バージョンの選択には SSLv3 または TLSv10,TLSv11,TLSv12 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、古いバージョン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/TLS 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/TLS 接続要求がない、または、SSL/TLS 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, SSLv3,
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, TLSv12, 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);
                                :

}
```

[例　4]

```
/* TLS1.0 connection, TLS_RSA_WITH_AES_256_CBC_SHA, Negotiate new session
*/
TASK https_task(void)
{
                                :
    ercd = ssl_con_cep(&https_cli_cep, cepid, certs, NULL,
                          TLS_RSA_WITH_AES_256_CBC_SHA, TLSv10, 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。E_TMOUTを返した場合は、次回のssl_snd_datで、保存してあったSSLレコードの続きから送信します。この場合、bufとlenには前回指定したデータを渡してください。他のデータを渡しても無視します。
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_snd_dat、ssl_rcv_dat、ssl_errのみ複数のタスクから同時にコールできます。

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_snd_dat、ssl_rcv_dat、ssl_err のみ複数のタスクから同時にコールできます。

ssl_get_buf

［機　能］　送信用バッファの取得

［形　式］　ER ssl_get_buf(T_SSL_CEP *ssl_cep, UB **p_buf, TMO tmout);

ssl_cep	SSL通信端点へのポインタ
p_buf	バッファアドレス格納先へのポインタ
tmout	タイムアウト指定

［戻り値］

正の値	正常終了(獲得したバッファのサイズ)
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］　このサービルコールでは、SSL 送信用バッファのアドレスとサイズを取得できます。このアドレスに送信したいデータを書き込んで ssl_snd_buf()を発行することで、余分なバッファを用意せずにデータを送信できます。ssl_get_buf()を発行した後は、必ず ssl_snd_buf()を発行してください。ssl_get_buf()の発行を取り消したい場合は、ssl_snd_buf()の len に 0 を指定して発行してください。ssl_snd_buf()でデータを送信した後に、再度データを送信したい場合は、必ず、ssl_get_buf()でバッファのアドレスを取得し直してください。送信バッファにデータが残っていた場合は、送信バッファが空になるまで待ち状態になります。そのため、ssl_get_buf で得られるサイズは、常に一度に送信可能な最大サイズが得られます。

［補　足］　ssl_get_buf()を発行すると、送信バッファがアラートメッセージの送信で使われないようにロックされます。他のタスクから並行して、ssl_rcv_dat()や ssl_rcv_buf()を発行できますが、ここでエラーが発生した場合、ssl_snd_buf()で送信バッファのロックが解除されるまで、アラートメッセージの送信が保留されます。保留されたアラートメッセージの送信がタイムアウトした場合は、アラートメッセージは送信されなくなります。

ssl_snd_buf

［機　能］　送信バッファに書き込んだデータの送信

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

ssl_cep	SSL通信端点へのポインタ
len	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
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］　このサービルコールでは ssl_snd_dat()と同様に、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。タイムアウトに関する動作は、ssl_snd_dat()と同じです。詳細は、ssl_snd_dat()の解説を参照ください。E_TMOUT を返した場合は、再度同じパラメータで ssl_snd_buf()を発行してください。その場合は、保存してあった SSL レコードの続きから送信します。ssl_snd_dat()と異なり、len で指定したサイズが必ず送信されますが、len には、ssl_get_buf()で得られたサイズより大きい値は指定できません。

ssl_rcv_buf

［機　能］　受信したデータの入ったバッファの取得

［形　式］　ER ssl_rcv_buf(T_SSL_CEP *ssl_cep, UB **p_buf, TMO tmout);

ssl_cep	SSL通信端点へのポインタ
p_buf	受信データ先頭アドレス格納先へのポインタ
tmout	タイムアウト指定

［戻り値］

正の値	正常終了(受信データの長さ)
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不十分
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］　このサービスコールでは受信バッファに残っているデータのアドレスとサイズを取得します。ssl_rcv_dat()と異なるのは、ssl_rel_buf()を発行するまで、データが受信バッファに残り続ける点です。その他の動作の詳細は、ssl_rcv_dat()の解説を参照ください。

ssl_rel_buf

［機　能］　受信バッファのデータの解放

［形　式］　ER ssl_rel_buf(T_SSL_CEP *ssl_cep, UW len);
ssl_cep　SSL通信端点へのポインタ
len　　データの長さ

［戻り値］　E_OK　　正常終了
E_PAR　　len で指定されたサイズのデータが受信バッファに残っていない

［解　説］　このサービスコールでは指定されたサイズ分だけ受信バッファのデータを捨てます。ssl_rcv_buf()の発行を複数回に分けて発行できます。ssl_rcv_buf()で得られたサイズを全て捨てた場合は、受信バッファが空になり、次回のssl_rcv_buf()でデータの受信処理に入ります。本サービスコールで待ち状態になることはありません。

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/TLS 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/TLS 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/TLS 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/TLS 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/TLS コネクションのクローズ処理を行う必要があります。SSL/TLS 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/TLS 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/TLS 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。
ssl_snd_dat、ssl_rcv_dat、ssl_err のみ複数のタスクから同時にコールできます。

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
SSLAP_UNSUPPORTED_EXT	110	338

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。

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;
}
```

第 8 章 FTPS クライアント

8.1 はじめに

サンプルプログラムとして、FTPS (File Transfer Protocol over SSL)のクライアントが SSL for NORTi Version 4.43j 以降で付属しています。FTPS クライアントは、SSL/TLS で FTPS サーバに接続してファイルの読み出しや書き込みを行いますが、本プログラムは、FTPS サーバにあるファイルを任意のデータ長で読み書きできる API と、それらを組み合わせた ftp コマンドの実装例から構成されます。

8.2 ファイル構成

nonftps.h	FTPS クライアントのヘッダ
nonftps.c	FTPS クライアントの API のソース
nonftpsc.c	FTPS 対応 ftp コマンド実装例のソース

FTPS クライアントの API を発行するユーザープログラムで nonftps.h をインクルードし、nonftps.c をプロジェクトに加えてビルドしてください。
Telnet サーバも組み込んで、Linux や Windows のコマンドプロンプトのように ftp コマンドを実行する場合は、nonftpsc.c もビルドに加えてください

8.3 使用するオブジェクト

TCP 受付口 ----- FTPS クライアント数×1
TCP 通信端点 -- FTPS クライアント数×2

FTPS クライアントの API は、複数のクライアントをサポートしています。TCP/IP スタックのコンフィグレーションでは、TCP 受付口 ID と TCP 通信端点 ID の上限値に上記の値を加算してください。

8.4 排他制御について

同一の FTPS クライアントに対する API は 1 つのタスクだけから発行するようにし、複数の FTPS クライアントを設ける場合でも、各 FTPS クライアントを扱うタスクの優先度は同じにしてください。同一の FTPS クライアントに対する API が複数のタスクで重なって発行されることを考慮した排他制御は省略されており、別々の FTPS クライアントであっても、優先度の異なるタスクから API が発行されることを考慮した排他制御は省略されていますので。
また、FTPS クライアントの待ち状態の発生する API を発行しているタスクに対して、待ちを解除する wup_tsk や rel_wai システムコールは発行しないでください。

8.5 コンフィグレーション

nonftps.h の次のマクロに、変更可能なバッファのサイズ等が定義されています。これらを変更する場合は、nonftps.h を直接書き換えるのではなく、nonftps.c と nonftps.h をインクルードして T_FTPS 構造体を定義しているソースのコンパイラオプションでマクロ定義することをお勧めします。()の値はデフォルト値です。

FTPS_SBUFSZC -- コマンドポート用の TCP 送信バッファのサイズ (2048)
FTPS_RBUFSZC -- コマンドポート用の TCP 受信バッファのサイズ (2048)
FTPS_SBUFSZD -- データポート用の TCP 送信バッファのサイズ (4096)
FTPS_RBUFSZD -- データポート用の TCP 受信バッファのサイズ (4096)
FTPS_SRECSZD -- データポート用の SSL 送信バッファのサイズ (4096＋2089)
FTPS_RRECSZD -- データポート用の SSL 受信バッファのサイズ (16384＋2048)
FTPS_TMOUT ----- 通信のタイムアウト値 (60000/MSEC)

コマンドポート用の TCP 送受信バッファのサイズ

FTP のコマンドポート用の TCP 通信端点の送信バッファサイズを FTPS_SBUFSZC で、受信バッファサイズを FTPS_RBUFSZC で変更できます。コマンドのやり取り自体で大きなパケットが送受信されることはありませんが、接続時のハンドシェークではまとまったサイズのパケットが送受信されることがあります。メモリを削減したい場合は、最小 256 バイトを目安に、2 のべき乗の値(256, 512, 1024)を指定してください。デフォルト値より大きな値にしても、ほとんど効果はありません。

データポート用の TCP 送受信バッファのサイズ

FTP のデータポート用の TCP 通信端点の送信バッファサイズを FTPS_SBUFSZD、受信バッファサイズを FTPS_RBUFSZD で変更できます。大きなファイルを送受信しない場合は小さくしても影響はありません。また、大きくすることで、RTT の大きな相手との送受信速度を改善できる可能性があります。このマクロにも、2 のべき乗の値を指定してください。

データポート用の SSL 送信バッファのサイズ

FTP のデータポート用の SSL/TLS レコードの送信バッファサイズを FTPS_SRECSZD で変更できます。大きなファイルを送信しない場合は、小さくしても影響はありせんが、2089 は必ず加えてください(目安は 1024+2089 程度まで)。大きくすることで効率よく SSL/TLS レコードを生成できるようになりますが、効果があるのは、16384＋2048 までです。このマクロは、2 のべき乗とする必要はありません。

データポート用の SSL 受信バッファのサイズ

FTP のデータポート用の SSL/TLS レコードの受信バッファサイズを FTPS_RRECSZD で

変更できます。ただし、ファイルのデータの受信だけではなく、SSL/TLS のネゴシエーションでも使用しており、サーバから送られてくるサイズ不定のレコード全体を受信できる必要がありますので、大幅な削減は難しいです。相手が大きなレコードを送信しないことが明確である場合を除いて、デフォルトのままとしてください。また、それより大きな値を指定しても効果はありません。このマクロも、2 のべき乗となっている必要はありません。

通信のタイムアウト値

FTPS クライアント内部で発行している TCP/IP や SSL/TLS の API のタイムアウト値を、FTPS_TMOUT で変更できます。デフォルト値は、性能の高くない CPU 向けに大きな値としてあります。

8.6 FTPS クライアントの API

ftps_ini

［機　能］　FTPS クライアントの初期化

［形　式］　ER ftps_ini(T_FTPS *ftps);
　　ftps　　FTPS クライアント管理ブロックへのポインタ

［戻り値］　E_OK　　正常終了

［解　説］　FTPS クライアント用の変数領域(管理ブロック)を、T_FTPS 構造体としてユーザープログラム側に定義し、その先頭アドレスを ftps に指定して最初に一度だけ発行してください。T_FTPS 構造体の名前は任意です。

［ 例 ］　T_FTPS ftps;
　　ftps_ini(&ftps);

ftpsn_ini

［機　能］　複数の FTPS クライアントの初期化

［形　式］　ER ftpsn_ini(T_FTPS *ftps, int n);
ftps　　FTPS クライアント管理ブロック配列へのポインタ
n　　FTPS クライアント数（1～）

［戻り値］　E_OK　　正常終了
E_PAR　　FTPS クライアント数が 1 未満

［解　説］　FTPS クライアントを複数設けて、別々のまたは同じ FTPS サーバに同時に複数の接続をすることができます。FTPS クライアントの数分の変数領域(管理ブロック)を T_FTPS 構造体の配列としてユーザープログラム側に定義し、その先頭アドレスを ftps に、配列の要素数を n に指定して、最初に一度だけ発行してください。
本 API で初期化する場合には、ftps_ini()は使用しないでください。
n に 1 を指定した場合には、ftps_ini()と同じになります。

［　例　］　T_FTPS ftps[2];
ftpsn_ini(ftps, 2);

ftps_select

［機　能］　FTPS クライアントの選択

［形　式］　ER ftps_select(int i);
　　i　　FTPS クライアント番号（0～）

［戻り値］　E_OK　　正常終了
　　E_PAR　　FTPS クライアント番号が範囲外

［解　説］　ftpsn_ini()で複数クライアントを初期化した場合は、本 API で、続く API の対象クライアントを選択してください。ftps_ini()で初期化した場合は、本 API を発行する必要はありません。
FTPS クライアント毎にユーザータスクを設けて、その各タスクの先頭で本 API が発行されることを想定していますが、例のように ftps_select()を適宜発行することで、1 つのユーザータスクで、複数のクライアントを切り替えながら使用することもできます。

［ 例 ］　ftps_select(0);
　　ftps_option(…);
　　ftps_select(1);
　　ftps_option(…);

ftps_option

［機　能］　FTPS オプションの設定

［形　式］　ER ftps_option(int optname, optval);
optname　オプションの種別
optval　オプションの値（種別によって省略）

［戻り値］　E_OK　　正常終了
E_NOSPT 未サポートのオプション
E_OBJ　すでにログインしている、または、FTPS クライアントが未選択

［解　説］　次の 3 種の optname と optval の組み合わせで、機能を 1 つずつ設定できます。

optname	optval
OPT_FTPS_NIF	ネットワークインタフェース（T_NIF *型）
OPT_FTPS_PORTNO	ポート番号（int 型）
OPT_FTPS_CALLBACK	コールバックルーチン（FTPS_CALLBACK 型）

複数のネットワークインタフェース(NIF)があってデフォルト以外の NIF で FTPS クライアントで使用する場合、OPT_FTPS_NIF オプションでそれを指定してください。
ウェルノウンの 21 番以外の FTPS サーバのポート番号を使用する場合は、OPT_FTPS_PORTNO オプションでそれを指定してください。
FTPS クライアントの処理中に発生したイベントを検知したい場合には、OPT_FTPS_CALLBACK オプションでコールバックルーチンを登録できます。

次の 6 種の optname の指定(optval は省略)で、モードを 1 つずつ設定できます。

optname	モード
OPT_FTPS_PASSIVE	パッシブモード（デフォルト）
OPT_FTPS_ACTIVE	アクティブモード
OPT_FTPS_EXPLICIT	Explicit モード（デフォルト）
OPT_FTPS_IMPLICIT	Implicit モード
OPT_FTPS_VERBOSE	応答メッセージ抑制
OPT_FTPS_DEBUG	デバッグモード

本 API は、FTPS サーバにログイン中には実行できませんが、ログアウト後に再度ログインする前には、オプションを変えての再発行もきます。

---

ftps_connect

［機　能］　FTPS サーバへの接続

［形　式］　ER ftps_connect(UW ipaddr, const char *user, const char *pass);
ipaddr　接続先の IP アドレス
user　　ユーザーID 文字列
pass　　パスワード文字列

［戻り値］　E_OK　　正常終了
E_PAR　ユーザーID、または、パスワードが不正
E_NOMEM メモリ不足
E_CLS　ログイン失敗で FTPS サーバとの接続を切断
E_OBJ　すでにログインしている、または、FTPS クライアントが未選択
その他　内部で発行している TCP/IP API 等のエラー

［解　説］　FTPS サーバに接続し、ログインまでを実行します。ログインに失敗した場合は、FTPS サーバとの接続を切断します。

ftps_cmd

［機　能］　FTP コマンドの実行

［形　式］　ER ftps_cmd(const char *command);
command　FTP コマンド文字列

［戻り値］　E_OK　　正常終了
E_PAR　コマンドが不正(FTPS サーバとの接続は継続)
E_CLS　通信エラーで FTPS サーバとの接続を切断
E_OBJ　FTPS クライアントが未選択

［解　説］　FTP コマンド"dir", "ls", "cd", "rm", "get", "put", "bye", "quit","ascii", "bin", "passive"を文字列で指定して直接実行できます。
コマンドにパラメータの文字列を付加する場合は、1 つ以上のスペースで区切ってください。
本 API を実行する前に、前もって ftps_connect()で FTPS サーバと接続しておく必要があります。
"get"、または、"dir"コマンドを実行した後は、続けて、ftps_read()でデータを読み出してください。
"put"コマンドを実行した後は、続けて、ftps_write()でデータを書き込んでください。

ftps_open

［機　能］　FTPS サーバのファイルオープン

［形　式］　ER ftps_open(const char *path, const char *mode);
　　path　　ファイル名
　　mode　　モード（"r": 読み出し、"w": 書き込み）

［戻り値］　E_OK　　正常終了
　　E_PAR　　ファイル名やモードが不正(FTPS サーバとの接続は継続)
　　E_CLS　　通信エラーで FTPS サーバとの接続を切断
　　E_OBJ　　FTPS クライアントが未選択

［解　説］　path で指定されたファイルを開きます。ftps_cmd()による"get"や"put"コマンドの代わりに利用できます。
　　mode に"r"を指定すると、続けて ftps_read()でデータを読み出せます。
　　mode に"w"を指定すると、続けて ftps_write()でデータを書き込めます。

ftps_read

［機　能］　FTPS サーバのファイルからのデータ読み出し

［形　式］　ER ftps_read(void *buf, int size);
　　buf　　読み出したデータを格納するバッファへのポインタ
　　size　　バッファのサイズ

［戻り値］　0 以上　正常終了(取得したデータ長)
　　E_OBJ　ファイル未オープン、または、FTPS クライアントが未選択
　　E_CLS　通信エラーで FTPS サーバとの接続を切断

［解　説］　FTPS サーバにあるファイルのデータを読み出します。
　　前もって、ftps_open()の"r"モードでファイルをオープンしておくか、ftps_cmd()で"dir"、または、"get"コマンドが実行されている必要があります。戻値が size の値未満の場合は、全てのデータの読み出しが終了したことを示します。

ftps_write

［機　能］　FTPS サーバのファイルへのデータ書き込み

［形　式］　ER ftps_write(const void *buf, int size);
　　buf　　書き込むデータが格納されているバッファへのポインタ
　　size　　データの長さ

［戻り値］　E_OK　　正常終了
　　E_CLS　通信エラーで FTPS サーバとの接続を切断
　　E_OBJ　ファイル未オープン、または、FTPS クライアントが未選択

［解　説］　FTPS サーバにあるファイルへデータを書き込みます。前もって、ftps_open()の"w"モードでファイルをオープンしておくか、ftps_cmd()で"put"コマンドが実行されている必要があります。

ftps_close

［機　能］　FTPS サーバのファイルクローズ

［形　式］　ER ftps_close(void);

［戻り値］　E_OK　　正常終了
　　E_CLS　通信エラーで FTPS サーバとの接続を切断
　　E_OBJ　FTPS クライアントが未選択

［解　説］　データの読み出し、または書き込みを終了します。
　　一連の ftps_read()や ftps_write()の後に、必ず発行してください。

ftps_exit

［機　能］　サーバとの接続終了

［形　式］　ER ftps_exit(void);

［戻り値］　E_OK　　正常終了
　　E_CLS　通信エラーで FTPS サーバとの接続を切断
　　E_OBJ　FTPS クライアントが未選択
　　その他　内部で発行している TCP/IP API 等のエラー

［解　説］　FTPS サーバとの接続を切断して FTPS クライアントを終了します。
　　オープン中のファイルがあれば先にクローズ(実行中のコマンドがあれば中断)します。

コールバック

［機　能］　イベント通知用のコールバックルーチン

［形　式］　void callback(int event, VP parblk, int len);
event　イベントコード(TEV_MESSAGE)
parblk　パラメータ
len　　パラメータの長さ

［戻り値］　なし

［解　説］　FTPS クライアントの処理中に発生したイベントを受け取ります。サポートされているイベントは、応答メッセージ通知(TEV_MESSAGE)のみです。
上記の形式で callback と表現されているのは、ftps_option()でコールバックルーチンに登録されるユーザー作成の関数で、名前は任意です。
コールバックルーチン内部では、本クライアントの API や TCP/IP API や待ち状態の発生するシステムコールを発行できません。

ftps_command

［機　能］　FTPS 対応 ftp コマンド処理

［形　式］　int ftps_command(int argc, char *argv[]);
argc　　コマンド/パラメーターの数
argv　　コマンド/パラメーターが格納されているアドレス

［戻り値］　正常終了
負　　内部で発行している FTPS クライアント API のエラー

［解　説］　nonftpsc.c に、FTPS クライアントの API の使用例として実装されており、Linux や Windows のコマンドプロンプトの ftp コマンドのように、Telnet 上で FTPS に対応したファイル転送のコマンドを実行します。
コマンドの形式は次のとおりです。

ftp [-v] [-d] [-i] [<nif>] <ipaddr>
-v　　応答メッセージ抑制
-d　　デバッグモード有効
-i　　Implicit モード(990 番ポートを使用)
<nif>　　ネットワークインタフェース名 (例 : eth0, eth1)
<ipaddr> FTPS サーバの IP アドレス (例 : 192.168.1.3)

第 9 章 HTTPS サーバ／クライアント

9.1 はじめに

サンプルプログラムとして、HTTPS (Hypertext Transfer Protocol Secure)のサーバとクライアントが SSL for NORTi に付属しています。

9.2 ファイル構成

nonhttps.h	HTTPS サーバ/クライアントのヘッダ
nonhttpss.c	HTTPS サーバ実装例のソース
nonhttps.c	HTTPS クライアント実装例のソース

HTTPS サーバを使用する場合は、その API を発行するユーザープログラムで nonhttps.h をインクルードし、nonhttpss.c をプロジェクトに加えてビルドしてください。
HTTPS クライアントを使用する場合は、その API を発行するユーザープログラムで nonhttps.h をインクルードし、nonhttps.c をプロジェクトに加えてビルドしてください。

9.3 使用するオブジェクト

HTTPS サーバ

タスク ---------- 1
TCP 受付口 ----- 1
TCP 通信端点 -- 1

HTTPS クライアント

TCP 通信端点 -- 1

TCP/IP スタックのコンフィグレーションでは、タスク ID と TCP 受付口 ID と TCP 通信端点 ID の上限値に上記の値を加算してください。なお、HTTPS クライアントの API は、FTPS とは異なり複数のクライアントをサポートしていません。

9.4 コンフィグレーション

HTTPS サーバもクライアントも一定サイズのメモリをバッファとして使用します。これらの領域は HTTPS サーバ/クライアントのモジュール内部に構造体変数として確保されますが、ユーザープログラム側に確保することもできます。具体的には、サーバでは nonhttpss.c に HTTPSS_XBUF マクロを定義した上で、「T_HTTPSS_BUF httpssbuf;」をユーザープログラムに宣言してください。クライアントでは nonhttps.c に HTTPS_XBUF マクロを定義した上で、「T_HTTPS_BUF httpsbuf;」をユーザープログラムに宣言してください。

9.5 HTTPS サーバの API

httpss_ini

［機　能］　HTTPS サーバを起動

［形　式］　int httpss_ini(ID tskid, ID cepid, ID repid)

taskid　　生成するタスク ID

cepid　　生成する TCP 通信端点 ID

repid　　生成する TCP 受付口 ID

［戻り値］　E_OK　　正常終了

その他　　内部で発行している TCP/IP API 等のエラー

［解　説］　HTTPS サーバに必要な各種オブジェクトを生成し、タスクを起動します。HTTPS サーバを運用するのに必要な処理は、本 API をコールするだけです。引数の各 ID に 0 を指定できます。

9.6 HTTPS クライアントの API

https_ini

［機　能］　HTTPS クライアントモジュールを初期化

［形　式］　ER https_ini(void)

［戻り値］　E_OK　　正常終了

［解　説］　HTTPS クライアントモジュールを初期化します。現バージョンでは、プロキシサーバ対応用に追加した内部変数を初期化するだけです。最初に 1 回だけ発行してください。

https_set_opt

［機　能］　HTTPS クライアントのオプション設定

［形　式］　ER https_set_opt(INT optname, const VP optval, INT optlen)
optname　オプションの種別
optval　オプションの値（種別によって省略）
optlen　optval の長さ

［戻り値］　E_OK　　正常終了

［解　説］　次の 2 種の optname と optval の組み合わせで、機能を 1 つずつ設定できます。

optname	optval
SET_PROXY_ADDR	プロキシサーバの IP アドレス設定(UW 型)
SET_PROXY_PORT	プロキシサーバのポート番号設定(UH 型)

SET_PROXY_ADDR で 0 以外の値を設定することで、プロキシサーバを介してアクセスできるようになります。デフォルトのポート番号は 3128 で、それを SET_PROXY_PORT で変更できます。
接続のたびに異なる値に設定することもでき、設定を解除する場合は、SET_PROXY_ADDR で 0 を指定してください。

［ 例 ］

```
UW ipaddr;
UH portno;

https_ini();
ipaddr = ascii_to_ipaddr("192.168.0.102");
portno = 8080;
https_set_opt(SET_PROXY_ADDR, &ipaddr, sizeof ipaddr);
https_set_opt(SET_PROXY_PORT, &portno, sizeof portno);
```

https_command

［機　能］　HTTPS クライアントのコマンド処理

［形　式］　int https_command(int argc, char *argv[])
argc　　コマンド/パラメーターの数
argv　　コマンド/パラメーターが格納されているアドレス

［戻り値］　0　　正常終了

［解　説］　コマンドパラメータを解析し、HTTPS サーバと接続し、応答をコンソールに表示します。コマンドの形式は次のとおりです。

https [-i nif] <ipaddr>
-i　　ネットワークインタフェース名を指定（例：eth0, eth1）
<ipaddr> HTTPS サーバの IP アドレス（例：192.168.1.3）

SSL for NORTi ユーザーズガイド

株式会社ミスポ http://www.mispo.co.jp/
〒222-0033 神奈川県横浜市港北区新横浜 3-20-8 BENEX S-3 12F
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