SSH スタック

ユーザーズガイド

2025 年 5 月版

MiSPO

株式会社ミスポ

目次

第 1 章 導入

1.1 はじめに

Version 4.43n 以降の「SSL for NORTi」に付属する SSH スタックは、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 Secure Shell の機能を実現します。暗号ライブラリは SSL/TLS と共通ですが機能は SSL/TLS と独立しており、SSH のみを使用する場合、SSL/TLS スタックのライブラリをリンクする必要はありません。

1.2 特長

本 SSH スタックは SSH Version 2.0 に対応しており、サポートする各アルゴリズムは次のとおりです。

鍵交換アルゴリズム	ecdh-sha2-nistp256（鍵長 256 ビット）
	ecdh-sha2-nistp384（鍵長 384 ビット）
	ecdh-sha2-nistp521（鍵長 521 ビット）
公開鍵暗号化アルゴリズム	ecdsa-sha2-nistp256（鍵長 256 ビット）
	ecdsa-sha2-nistp384（鍵長 384 ビット）
	ecdsa-sha2-nistp521（鍵長 521 ビット）
	ssh-ed25519（鍵長 256 ビット）
データ暗号化アルゴリズム	aes128-ctr（鍵長 128 ビット）
	aes192-ctr（鍵長 192 ビット）
	aes256-ctr（鍵長 256 ビット）
MAC アルゴリズム	hmac-sha2-256, hmac-sha2-512
公開鍵認証アルゴリズム	ecdsa-sha2-nistp256、ecdsa-sha2-nistp384、
	ecdsa-sha2-nistp521、ssh-ed25519

1.3 制限事項

暗号化/復号に用いられる関数のソースコードは非公開で、暗号ライブラリのみが収録されています。

1.4 ファイル構成

本 SSH スタックは、次のファイルで構成されています。

ヘッダファイル

nonssh.h　SSH スタックの API のヘッダ
このヘッダファイルには、アプリケーションで SSH スタックを利用するための関数のプロトタイプ宣言や構造体の定義がされています。SSH の機能を呼び出すアプリケーションでインクルードしてください。

nonsshp.h　SSH スタックの内部定義ヘッダ
このヘッダファイルには、SSH スタックの内部で使用している関数のプロトタイプ宣言や構造体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。

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、SSH スタックの内部で実行している暗号化/復号の処理に必要な宣言や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

nonssh.c　SSH スタックの API のソース
nonsshs.c　SSH スタックの送信処理のソース
nonsshr.c　SSH スタックの受信処理のソース
nonssch.c　SSH チャネル処理のソース
nonsshc.c　SSH 暗号化/復号処理のソース
nonsshm.c　SSH スタック内部で使われるその他の関数のソース

下記の SSH スタックのライブラリを生成するためのソースですので、これらをアプリケーションに加えてビルドする必要はありません。

ライブラリ

nssh???b.a　SSH スタックのライブラリ(ビッグエンディアン)
nssh???l.a　SSH スタックのライブラリ(リトルエンディアン)
crypt???l.a 暗号ライブラリ(リトルエンディアン)
crypt???b.a 暗号ライブラリ(ビッグエンディアン)

SSH スタックのライブラリと共に、暗号ライブラリもリンクしてください。ファイル名の???の部分は対応 CPU 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。

サンプルプログラム

nonsshcs.c　SSH クライアントの実装例のソース
nonsshd.c　SSH サーバの実装例のソース

1.5 用語

SSH コネクション

SSH コネクションは、サーバとクライアントの間で TCP/IP プロトコル上に確立される信頼できる安全な通信路です。具体的には、サーバとクライアントの間で共通に使用できる暗号アルゴリズムや認証アルゴリズムや鍵の交換方式などの交渉を行って接続し、さらにホスト認証やユーザー認証も行って SSH コネクションは確立されます。

SSH チャネル

確立した SSH コネクションは多重化され、1 つの SSH コネクション上に複数の SSH チャネルを開いてデータの送受信を行えます。SSH チャネルの種類には、トンネリングの X11 セッションや TCP/IP ポート転送もありますが、本 SSH スタックは通常のセッションにのみ対応しています。

第 2 章 コンフィグレーション

2.1 コンフィグレーション用マクロ

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

SSH_PRI	5	SSH スタックの API を実行中のタスク優先度
SSH_MPLSZ	12*1024	SSH 内部で使用する可変長メモリプールのサイズ
SSH_MPLID	0	SSH 内部で使用する可変長メモリプールの ID
SSH_CHS	2	SSH コネクション上に同時に開ける SSH チンネル数
SSH_DTMO	200/MSEC	SSH 受信データ終了判断のタイムアウト値
SSH_KEX_MSGSZ	1800	相手から受信する鍵交換メッセージ(KEXINIT)の最大サイズ
TCP_PORT_SSH	22	SSH サーバのポート番号

SSH_PRI の値に、SSH スタックの API を呼び出したタスクの優先度を一時的に上げることで、API 間の排他制御を行っています。SSH_PRI 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。
SSH_MPLID が 0 の場合、可変長メモリプールの ID が自動的に割り当てられます。
SSH_DTMO は ssh_rcv_pkt()での受信パケットの区切りの判断に使われるタイムアウト値、TCP_PORT_SSH は SSH サーバのデフォルトのポート番号で、いずれも、ssh_set_opt()で変更することもできます。

2.2 不要コードの削除

コンパイラオプションのマクロ定義で下記を指定して SSH スタックのライブラリを再生成することにより、使用しない機能のコードを省くことができます。

SSH_SERVER_ONLY	SSH サーバ向けのコードのみを有効
SSH_CLIENT_ONLY	SSH クライアント向けのコードのみを有効

2.3 使用するオブジェクト

SSH スタックの処理は、その API を呼び出したタスクのコンテキストで実行され、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いません。
SSH スタックで使用するオブジェクトの数が、nonssh.h に次のマクロで定義されていますので、「#define MPLID_MAX 1+(TCP_NMPL+SSH_NMPL)」のようにカーネルのコンフィグレーションで利用できます。

オブジェクト	マクロ名	値
タスク	SSH_NTSK	0
セマフォ	SSH_NSEM	0
イベントフラグ	SSH_NFLG	0
メールボックス	SSH_NMBX	0
メッセージバッファ	SSH_NMBF	0
ランデブ用ポート	SSH_NPOR	0
可変長メモリプール	SSH_NMPL	1
固定長メモリプール	SSH_NMPF	0
データキュー	SSH_NDTQ	0
ミューテックス	SSH_NMTX	0
割り込みサービスルーチン	SSH_NISR	0
周期ハンドラ	SSH_NCYC	0
アラームハンドラ	SSH_NALM	0

第 3 章 共通定義

3.1 エラーコード

SSH スタックの API からは、カーネルや TCP/IP スタックで共通のエラーコード以外に、下記の SSH 独自のエラーコードが返ります。

EV_SSHNEG	-97	SSH サーバとクライアント間の暗号アルゴリズムの交渉に失敗
EV_SSHKEY	-98	SSH サーバから受信した公開ホスト鍵の検証に失敗
EV_SSHMSG	-99	接続相手から受信したメッセージを解析できない
EV_SSHMAC	-100	接続相手から受信した MAC(メッセージ認証コード)が不正
EV_SSHUNEX	-101	接続相手から予期しないメッセージを受信
EV_SSHCHOPN	-102	SSH サーバからチャネル開始の要求を拒否された
EV_SSHCHREQ	-103	SSH サーバからチャネル固有の要求を拒否された
EV_SSHAUTH	-104	パスワード認証や公開鍵認証方式のユーザー認証に失敗
EV_SSHDECPUB	-105	公開鍵認証方式のユーザー認証の公開鍵を解読できない
EV_SSHDECPRIV	-106	公開鍵認証方式のユーザー認証の秘密鍵を解読できない
EV_SSHDCYRRIV	-107	公開鍵認証方式のユーザー認証の秘密鍵の保護を解除できない

3.2 SSH スタック制御用の構造体

SSH スタックでは、その制御に必要な情報を T_SSH_CON 型の構造体である「SSH コネクション制御ブロック」と T_SSH_CH 型の構造体である「SSH チャネル制御ブロック」で管理しています。その他に SSH サーバのサンプルでは、「SSH サーバ制御ブロック」を使用しています。

3.2.1 SSH コネクション制御ブロック

SSH コネクションの制御ブロックとバッファをアプリケーション側で、下記の例のようにコネクション毎に定義してください。

```
#define SSH_BUFSZ  ?????                SSH 送信バッファのサイズ（4 の倍数）
#define TCP_WINSZ  ?????                TCP 送受信ウィンドウバッファのサイズ（4 の倍数）
UW ssh_buf[(SSH_BUFSZ+TCP_WINSZ*2)/4];  SSH コネクション用バッファ（UW で 4 バイト境界に）
T_SSH_CON ssh_con = { ssh_buf, SSH_BUFSZ, TCP_WINSZ };
```

T_SSH_CON 型の構造体には多くのメンバが含まれていますが、アプリケーションで初期化が必要なのは上記の先頭の 3 つのメンバでだけで、次のようなコードで動的に設定することもできます。

```
ssh_con.buf   = ssh_buf;                SSH コネクション用バッファ
ssh_con.bufsz = SSH_BUFSZ;              SSH 送信バッファのサイズ
ssh_con.winsz = TCP_WINSZ;              TCP 送受信ウィンドウバッファのサイズ
```

なお、ssh_con、ssh_buf、SSH_BUFSZ、TCP_WINSZ という変数名やマクロ名は一例ですので、それぞれのコネクションの用途に合わせた適切な名前にしてください。

SSH 送信バッファのサイズ

SSH 送信バッファは、SSH コネクションの確立時に必要なメッセージの送信、および、SSH チャネルを通してデータを送信する際に SSH ヘッダを付加する処理に使用されます。
必要なサイズは、SSH コネクション確立時のメッセージのために最小でも 512 バイト、かつ、SSH ヘッダ全長(SSH_TOTAL_HDRSZ)の 105 バイト＋SSH 送信データのサイズ以上です。

3.2.2 SSH チャネル制御ブロック

SSH チャネルの制御ブロックと受信キューをアプリケーション側で、下記の例のようにチャネル毎に定義してください。

```
#define SSH_QUESZ  ?????                SSH 受信キューのサイズ (4 の倍数)
UW ssh_que[SSH_QUESZ/4];                SSH 受信キュー (UW で 4 バイト境界に)
T_SSH_CH ssh_ch = { &ssh_con, ssh_que, SSH_QUESZ };
```

T_SSH_CH 型の構造体には多くのメンバが含まれていますが、アプリケーションで初期化が必要なのは上記の先頭の 3 つのメンバだけで、次のようなコードで動的に設定することもできます。

```
ssh_ch.con   = &ssh_con;                SSH コネクション制御ブロック
ssh_ch.que   = ssh_que;                 SSH 受信キュー
ssh_ch.quesz = SSH_QUESZ;               SSH 受信キューのサイズ
```

SSH コネクション制御ブロックへのポインタ

SSH チャンネルを開く SSH コネクションの制御ブロックへのポインタを指定してください。SSH のセッションの開始時に、この SSH コネクションの状態を確認し、未確立の場合は確立処理を行います。すでに確立されている場合は、チャンネルのみを開きます。

SSH 受信キューとサイズ

SSH コネクションは SSH チャネルで多重化されていますので、そのチャネルでアプリケーションが受信待ちでなくても SSH コネクションから届くデータを受け取っておく必要があり、それに SSH 受信キューが使われます。サイズには、次に相手から受信するデータ長以上の値を指定してください。

第 4 章 SSH スタックの API

API 一覧

ssh_ini	SSH スタックの初期化
ssh_set_opt	SSH オプションの設定
ssh_opn_ses	SSH セッションの開始
ssh_reg_key	SSH サーバのホスト鍵のリストを登録
ssh_acp_ses	SSH セッションの開始要求待ち
ssh_snd_dat	SSH データの送信
ssh_snd_pkt	SSH パケットの送信
ssh_rcv_dat	SSH データの受信
ssh_rcv_pkt	SSH パケットの受信
ssh_sht_ses	SSH データ送信の終了
ssh_cls_ses	SSH セッションの終了
ssh_cls_con	SSH コネクションの終了
ssh_ext	SSH スタックの終了

ssh_ini

［機　能］ SSH スタックの初期化　　（SSH サーバ/クライアント）

［形　式］ ER ssh_ini(void);

［戻り値］ E_OK　　正常終了
負の値　　メモリプールの生成に失敗

［解　説］ SSH スタック内部で使用するデータの初期化やメモリプールの生成を行います。メモリプールを生成できなかった場合は、cre_mpl または acre_mpl システムコールのエラーコードを返します。この API は、SSH スタックの他の全ての API を使用する前にタスクから呼び出してください。

ssh_set_opt

［機　能］ SSH オプションの設定　　(SSH サーバ/クライアント)

［形　式］ ER ssh_set_opt(T_SSH_CON *con, INT optname, const VP optval, INT optlen);

con	SSH コネクション制御ブロックへのポインタ
optname	オプションの種類
optval	オプションが格納されている領域へのポインタ
optlen	オプションの長さ

［戻り値］

E_OK	正常終了
E_PAR	不正な引き数
E_NOSPT	未サポートのオプション
EV_SSHDECPRIV	公開鍵認証方式のユーザー認証の秘密鍵を解読できない
EV_SSHDECPUB	公開鍵認証方式のユーザー認証の公開鍵を解読できない
EV_SSHDCYRRIV	公開鍵認証方式のユーザー認証の秘密鍵の保護を解除できない

［解　説］ ssh_ini()の直後に呼び出すことで、optname に指定した以下のオプションの設定ができます。

SSH_OPT_SERVIP	SSH サーバの IP アドレス
SSH_OPT_SERVPORT	SSH サーバのポート番号
SSH_OPT_USERNAME	ユーザー認証に使用するユーザー名
SSH_OPT_PASSWORD	パスワード認証方式のユーザー認証で使用するパスワード
SSH_OPT_USERKEYS	公開鍵認証方式で使用する公開鍵と秘密鍵とパスフレーズ
SSH_OPT_DTMO	SSH 受信データ終了判断のタイムアウト値
SSH_OPT_CALLBACK	SSH のイベント通知コールバックルーチンを登録
SSH_OPT_BANNER	SSH バナーメッセージ

SSH_OPT_SERVIP と SSH_OPT_SERVPORT で SSH サーバの IP アドレスとポート番号を設定してください。SSH_OPT_SERVPORT を設定しない場合は、SSH でデフォルトの 22 番ポートが使用されます。optval には IP アドレスやポート番号を格納したロングワード(UW)型の変数へのポインタを指定し、optlen には sizeof (UW)を指定してください。

SSH_OPT_USERNAME でパスワード認証方式や公開鍵認証方式のユーザー認証に使用するユーザー名を設定でき、SSH_OPT_PASSWORD でパスワード認証方式のパスワー

ドを設定できます。optval にはユーザー名やパスワードの文字列へのポインタを指定してください。長さは、文字列の終端の Null 文字 '¥0'で判断されますので、optlen の指定は不要です(0 を推奨)。

SSH_OPT_USERKEYS で optval に次ページの[例 1]や[例 2]のように定義された下記の構造体へのポインタを指定して、公開鍵認証方式で使用する公開鍵と秘密鍵、および秘密鍵が保護されている場合はパスフレーズを設定できます。

```
typedef struct {
    const char *pubkey;     OpenSSH 形式の公開鍵へのポインタ
    const char *privkey;    OpenSSH 形式の秘密鍵へのポインタ
    const char *passphrase; 秘密鍵のパスフレーズへのポインタ
} T_USER_KEYS;                          (パスフレーズで保護しない場合は NULL)
```

この SSH_OPT_USERKEYS の処理では、公開鍵を OpenSSH 形式からバイナリ形式に変換し、変換後のデータを解読して認証に使用する部分を抽出して内部に保存しますが、解読に失敗した場合は EV_SSHDECPUB エラーを返します。同様に、秘密鍵を OpenSSH 形式からバイナリ形式に変換し、保護されていた場合はパスフレーズでその解除を行った後、認証に使用する部分を抽出して保存しますが、解除に失敗した場合は EV_SSHDCYRRIV エラーを、解読に失敗した場合は EV_SSHDECPRIV エラーを返します。パスワード認証方式のパスワードと公開鍵認証方式の鍵の両方が設定されていた場合は公開鍵認証が優先されますが、上記のエラーとなる場合は、パスワード認証も試みます。

SSH_OPT_DTMO で、ssh_rcv_pkt()でデータが途切れたとする判断するタイムアウト値をデフォルトの 200msec から変更できます。optval には、例えば、100/MSEC のような Tick 単位の時間を格納したロングワード型の変数へのポインタを指定し、optlen には sizeof (UW)を指定してください。

SSH_OPT_CALLBACK では、optval に SSH でのイベント通知のコールバックルーチンとする関数へのポインタを指定し、optlen には sizeof (FP)を指定してください。

SSH_OPT_BANNER では、SSH サーバが接続してきたクライアントに対し、ユーザー認証の時に表示するバナーメッセージを設定できます。optval にはバナーメッセージの文字列へのポインタを指定してください。長さは、文字列の終端の Null 文字 '¥0'で判断されますので、optlen の指定は不要です(0 を推奨)。

[例 1]

```
/* ED25519 Public/Private Keys. Private Key is not encrypted. */

static T_USER_KEYS my_ed25xkey = {

"ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDLzB6pEjdandlf+3H3BpDPflTogMem5K¥
4L2BnSrwJTa mispo-Vostro-1520-26-11-2024",

"-----BEGIN OPENSSH PRIVATE KEY-----¥
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW¥
QyNTUxOQAAACAy8weqRI3Wp3ZX/tx9waQz35U6IDHpuSuC9gZOq8CU2gAAAKCPBqN/jwaj¥
fwAAAAtzc2gtZWQyNTUxOQAAACAy8weqRI3Wp3ZX/tx9waQz35U6IDHpuSuC9gZOq8CU2g¥
AAAEDefxDcm+KRniZwvj7AoUaOfOd5ipQu2C3QS97wBEgLxzLzB6pEjdandlf+3H3BpDPf¥
lTogMem5K4L2BnSrwJTaAAAAHG1pc3BvLVZvc3Ryby0xNTIwLTI2LTExLTIwMjQB¥
-----END OPENSSH PRIVATE KEY-----",

NULL
};

func()
{
    if (ssh_set_opt(&ssh_con, SSH_OPT_USERKEYS, &my_ed25xkey,
                                            sizeof (T_USER_KEYS)) != E_OK)
        print("Public/Private key setup failed¥r¥n");
```

[例 2]

```
/* ECDSA Public/Private Keys and Passphrase. Private key is encrypted with
aes256-ctr. */

static T_USER_KEYS my_ecdsa_enc = {

"ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAA¥
ABBBAt4eOQztHdRQ+KbJAiR6UQg1aEo1Tl2t6+gyYmahhUi+ZTPxvUBKwlufEc46e8T7zY¥
v7miCL+ansGfioDhYgUk= mispo@mispo-Vostro-1520",

"-----BEGIN OPENSSH PRIVATE KEY-----¥
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXBOAAAAGAAAABA5MUOGDd¥
FOmYD6xN1GTYqPAAAAAQAAAAEAAABoAAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlz¥
dHAyNTYAAABBBAt4eOQztHdRQ+KbJAiR6UQg1aEo1Tl2t6+gyYmahhUi+ZTPxvUBKwlufE¥
c46e8T7zYv7miCL+ansGfioDhYgUkAAACwNdK47zhvxBQ26rQX+Gjf+GPRLqqBoIQVXTBR¥
Q5SjkWb7aHNr9fwiBVnrQKlmvHA54Z1i+MQSHwfJ3DLakdjCUUIZ3GXFQS5O6fYzoFctnO¥
5UPfuTjYnsj/a5/MQywjWWODibhtB6JQeOr53AN6QKI43pKUZtuMWmYlUpfNMpOJxdrpBq¥
LQt43rk2K8BeKZepObQZ7wP7l4/3HKcBpAtT242VYAt4P42goL/moLoH3no=¥
-----END OPENSSH PRIVATE KEY-----",

"priv-pwd"
};

func()
{
    if (ssh_set_opt(&ssh_con, SSH_OPT_USERKEYS, &my_ecdsa_enc,
                                            sizeof (T_USER_KEYS)) != E_OK)
        print("Public/Private key setup failed¥r¥n");
```

ssh_opn_ses

［機　能］ SSH セッションの開始　　（SSH クライアント）

［形　式］ ER ssh_opn_ses(T_SSH_CH *ch, TMO tmout);

ch	SSH チャネル制御ブロックへのポインタ
tmout	タイムアウト指定

［戻り値］

E_OK	正常終了
E_PAR	不正な引き数
E_NOMEM	SSH コネクションを確立するためのメモリが不足
E_OBJ	SSH コネクション制御ブロックが不正
E_SYS	暗号ライブラリ関数の内部エラー
EV_SSHNEG	SSH サーバとの暗号アルゴリズムの交渉に失敗
EV_SSHKEY	SSH サーバから受信した公開ホスト鍵の検証に失敗
EV_SSHCHOPN	SSH サーバからチャネル開始の要求を拒否された
EV_SSHCHREQ	SSH サーバからチャネル固有の要求を拒否された
EV_SSHAUTH	パスワード認証や公開鍵認証方式のユーザー認証に失敗
EV_SSHUNEX	SSH サーバから予期しないメッセージを受信
EV_SSHMSG	SSH サーバから受信したメッセージを解析できない
E_TMOUT	タイムアウト
その他	tcp_vcre_cep、tcp_con_cep、tcp_snd_dat、tcp_rcv_dat のエラー

［解　説］ SSH コネクションが未確立の場合、先に TCP 通信端点を生成して SSH コネクションを確立します。そして、確立した SSH コネクション上に SSH セッション用のチャネルを開きます。SSH コネクションは、SSH チャネル制御ブロックにリンクされている SSH コネクション制御ブロック ch->con で管理されます。
タイムアウトなしで本 API を呼び出した場合、SSH セッションが正常に開始されるまで待ち状態となります。タイムアウトありで本 API を呼び出した場合、その時間が経過しても SSH セッションが開始されなければ、E_TMOUT エラーが返ります。
全ての API で、tmout に指定できるのは、TMO_FEVR(タイムアウトなし)か 1～0x7fffffff(タイムアウトあり)で、TMO_POL(ポーリング)や TMO_NBLK(ノンロッキング)はサポートしていません。

ssh_reg_key

［機　能］ SSH サーバのホスト鍵のリストを登録　　(SSH サーバ)

［形　式］ ER ssh_reg_key(const T_SERV_KEYS *keys, int n);

keys　　公開鍵と秘密鍵へのポインタを格納した配列へのポインタ

n　　公開鍵と秘密鍵のペアの数

［戻り値］

E_OK	正常終了
E_PAR	不正な引き数
E_NOMEM	公開鍵と秘密鍵の解析に必要なメモリが不足
E_NOSPT	鍵の暗号アルゴリズムが未サポート
EV_SSHDECPRIV	公開鍵認証方式のユーザー認証の秘密鍵を解読できない
EV_SSHDECPUB	公開鍵認証方式のユーザー認証の公開鍵を解読できない
EV_SSHDCYRRIV	公開鍵認証方式のユーザー認証の秘密鍵の保護を解除できない

［解　説］ この API で、次ページの例のように初期設定された下記の構造体の配列へのポインタを keys に指定し、サーバのホスト鍵(公開鍵と秘密鍵のペア、および秘密鍵が保護されている場合はパスフレーズ)をまとめて登録してください。

```
typedef struct {
    const char *pubkey;     OpenSSH 形式の公開鍵へのポインタ
    const char *privkey;    OpenSSH 形式の秘密鍵へのポインタ
    const char *passphrase; 秘密鍵のパスフレーズへのポインタ
} T_SERV_KEYS;                          (パスフレーズで保護しない場合は NULL)
```

この API では、公開鍵を OpenSSH 形式からバイナリ形式に変換し、変換後のデータを解読してホスト認証に使用する部分を抽出して内部に保存しますが、解読に失敗した場合は EV_SSHDECPUB エラーを返します。同様に、秘密鍵を OpenSSH 形式からバイナリ形式に変換し、保護されていた場合はパスフレーズでその解除を行った後、ホスト認証に使用する部分を抽出して保存しますが、解除に失敗した場合は EV_SSHDCYRRIV エラーを、解読に失敗した場合は EV_SSHDECPRIV エラーを返します。

登録されたホスト鍵は以降のすべての SSH 接続で使用されるため、本 API を接続のたびに呼び出すのではなく、SSH スタックの初期化 ssh_ini()の後に一度だけ呼び出してください。

keys に NULL を指定すると、ホスト鍵のリストを登録した時に内部で確保したメモリブロックを返却し、登録を解除します。

[例]

```
/* Public Key/Private Key/Passphrase for each Host Key Algorithm */

static const T_SERV_KEYS svr_keys[] = {
{
"ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAA¥
ABBBBOwtRCFMUVtLMtfg/qV/AZaUyVEBsuR5dp8A3I9BdTKk9XoIPDFNs6qQhjfHG3b1XD¥
mdBpe3Hb3pQKFls+t5A8=",
"-----BEGIN OPENSSH PRIVATE KEY-----¥
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXBOAAAAGAAAABCfcanuzZ¥
VvYM7f5fSoFwa6AAAAAQAAAAEAAABoAAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlz¥
dHAyNTYAAABBBBOwtRCFMUVtLMtfg/qV/AZaUyVEBsuR5dp8A3I9BdTKk9XoIPDFNs6qQh¥
jfHG3b1XDmdBpe3Hb3pQKFls+t5A8AAACw+xP4p2mZ1Fa4j5KWwYOYOEInL4vsjyR2GW4L¥
22tvxQ6dA42hiR+sy6CnZuaXbwqe3B1THI5DrLwOz/Cu9DXriDyT5pliSbEMY9m7SnYkc9¥
urr4giCBDNVlPBZcXIIB5mLeq+6OXDtZ25NcOPmy/3TO4xSpKhNWl7Z1DTwv7EJsDsKjEz¥
KkX7nw5rQd2CdYVlybTGhcGDenrlGwu+LsFFcUFxoqPX/ZZv/JtDVzT9ZtQ=¥
-----END OPENSSH PRIVATE KEY-----",
"sshd_keypwd" },

{
"ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPZrU+EZ/kGvueMTLY+BNYfoOE4R8QJOd¥
7cOlbhlaUOs mispo@mispo-Vostro-1520",
"-----BEGIN OPENSSH PRIVATE KEY-----¥
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW¥
QyNTUxOQAAACD2a1PhGf5Br7njEy2PgTWH6NBOEfECTne3DpW4ZWlNLAAAAKB+YwTOfmME¥
zgAAAAtzc2gtZWQyNTUxOQAAACD2a1PhGf5Br7njEy2PgTWH6NBOEfECTne3DpW4ZWlNLA¥
AAAEB8EO/9Q/HCw59NxPYgccGXDclooqbidDt1lU24X8r7l/ZrU+EZ/kGvueMTLY+BNYfo¥
OE4R8QJOd7cOlbhlaUOsAAAAF21pc3BvQG1pc3BvLVZvc3Ryby0xNTIwAQIDBAUG¥
-----END OPENSSH PRIVATE KEY-----",
NULL }
};

func()
{
    if (ssh_reg_key(svr_keys, sizeof svr_keys /
                                        sizeof (T_SERV_KEYS)) != E_OK)
        print("Loading of server keys failed¥r¥n");
```

ssh_acp_ses

［機　能］ SSH セッションの開始要求待ち (SSH サーバ)

［形　式］ ER ssh_acp_ses(T_SSH_CH *ch, TMO tmout);

ch	SSH チャネル制御ブロックへのポインタ
tmout	タイムアウト指定

［戻り値］

E_OK	正常終了
E_PAR	不正な引き数
E_NOMEM	SSH コネクションを確立するためのメモリが不足
E_OBJ	SSH コネクション制御ブロックが不正
E_SYS	暗号ライブラリ関数の内部エラー
EV_SSHNEG	SSH クライアントとの暗号アルゴリズムの交渉に失敗
EV_SSHAUTH	パスワード認証や公開鍵認証方式のユーザー認証に失敗
EV_SSHUNEX	SSH クライアントから予期しないメッセージを受信
EV_SSHMSG	SSH クライアントから受信したメッセージを解析できない
E_TMOUT	タイムアウト
その他	tcp_vcre_cep、tcp_vcre_rep、tcp_snd_dat、tcp_rcv_dat のエラー

［解　説］ SSH コネクションが未確立の場合、先に TCP 通信端点および TCP 受付口を生成して SSH クライアントからの接続要求を待ちます。クライアントから要求を受信した後、SSH コネクションを確立し、クライアントから SSH セッション開始要求を受信した後、チャネルを開きます。SSH コネクションは、SSH チャネル制御ブロックにリンクされている SSH コネクション制御ブロック ch->con で管理されます。

タイムアウトなしで本 API を呼び出した場合、SSH セッションが正常に開始されるまで待ち状態となります。タイムアウトありで本 API を呼び出した場合、その時間が経過しても SSH セッションが開始されなければ、E_TMOUT エラーが返ります。

全ての API で、tmout に指定できるのは、TMO_FEVR(タイムアウトなし)か 1～0x7fffffff(タイムアウトあり)で、TMO_POL(ポーリング)や TMO_NBLK(ノンロッキング)はサポートしていません。

ssh_snd_dat

［機　能］ SSH データの送信　　(SSH サーバ/クライアント)

［形　式］ ER ssh_snd_dat(T_SSH_CH *ch, const VP data, INT len, TMO tmout);

ch	SSH チャネル制御ブロックへのポインタ
data	送信データへのポインタ
len	送信データの長さ
tmout	タイムアウト指定

［戻り値］

正の値	正常終了(送信できたデータの長さ)
E_PAR	不正な引き数
E_NOMEM	SSH コネクションの送信バッファのサイズが不足
E_OBJ	SSH コネクション制御ブロックが不正
E_TMOUT	タイムアウト
その他	tcp_snd_dat のエラー

［解　説］ このAPIでは、送信データにSSHヘッダとMAC(Message Authentication Code：メッセージ認証コード)を追加し、暗号化した上で、tcp_snd_datでデータを送信します。

使用するSSHコネクションの送信バッファはフロー制御がされており、送信可能でない場合は、可能となるまで待ち状態となります。タイムアウトありで本APIを呼び出した場合、指定した時間が経過しても、送信可能にならなければ、E_TMOUTエラーが返ります。

なお、送信データの長さと戻り値は必ずしも同じにはなりません。送信可能なサイズが小さい場合、そのサイズまで送信して本APIはリターンしますので、残りの送信データへのポインタと残りの長さを指定して、このAPIを繰り返し呼び出す必要があります。

また、SSHコネクションの送信バッファへデータを渡した後に、tcp_snd_datでTCP通信端点の送信バッファ(TCPウィンドウバッファ)へ一度にデータを渡せずにtcp_snd_datを繰り返す可能性があります。tmoutの指定はtcp_snd_datでも使われますので、その場合、繰り返しの分だけタイムアウト時間が延びることになります。

ssh_snd_pkt

［機　能］ SSHパケットの送信 （SSHサーバ/クライアント）

［形　式］ ER ssh_snd_pkt(T_SSH_CH *ch, const VP data, INT len, TMO tmout);

ch	SSHチャネル制御ブロックへのポインタ
data	送信データへのポインタ
len	送信データの長さ
tmout	タイムアウト指定

［戻り値］

正の値	正常終了(送信できたデータの長さ)
E_PAR	不正な引き数
E_NOMEM	SSHコネクションの送信バッファのサイズが不足
E_OBJ	SSHコネクション制御ブロックが不正
E_TMOUT	タイムアウト
その他	tcp_snd_datのエラー

［解　説］ ssh_snd_dat()との違いは、lenで指定された長さのデータを全て送信し終わるまで、リターンしないことです。よって、正常な場合の戻り値は送信データの長さと一致しており、それをチェックして本APIを繰り返し呼び出すようなことをする必要はありません。

なお、tmoutの指定は、チャネルの送信バッファが送信可能になるまでと、tcp_snd_datで使われますので、それらの処理が内部で繰り返されると、その分だけタイムアウト時間は延びることになります。

ssh_rcv_dat

［機　能］ SSH データの受信　　(SSH サーバ/クライアント)

［形　式］ ER ssh_rcv_dat(T_SSH_CH *ch, VP buf, INT len, TMO tmout);

ch	SSHチャネル制御ブロックへのポインタ
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(受信したデータの長さ)
0	データ終結(SSH_MSG_CHANNEL_EOF メッセージを受信)
E_QOVR	受信キューのオーバフロー
E_OBJ	SSH コネクション制御ブロックが不正
EV_SSHUNEX	接続相手から予期しないメッセージを受信
EV_SSHMSG	接続相手から受信したメッセージを解析できない
EV_SSHMAC	受信したメッセージの MAC(メッセージ認証コード)が不正
E_TMOUT	タイムアウト
その他	tcp_rcv_dat のエラー

［解　説］ この API では、SSH コネクションから tcp_rcv_dat で受信した SSH パケットを復号し、SSH ヘッダと MAC(Message Authentication Code：メッセージ認証コード)を外したデータを、宛て先の SSH チャネルの受信キューへ保存します。この処理を、ch で指定した SSH チャネルの受信キューにデータが保存されるまで実行し、len で指定の長さを読み出して越える分は受信キューに残ります。

この一連の処理の前に受信キューにデータがあった場合は、新たな SSH パケットは受信せず、それを読み出します。いずれの場合も、受信データが len で指定の長さより短くても、それを受け取ってリターンしますので、受信したい長さとなるまで、本 API を繰り返して残りのデータを受信する必要があります。

tmout の指定は tcp_rcv_dat で使われますが、tcp_rcv_dat で TCP 通信端点の受信バッファ(TCP ウィンドウバッファ)から一度で SSH パケットを取り出せなかった場合は tcp_rcv_dat が繰り返され、その場合、繰り返しの分だけタイムアウト時間が延びることになります。

最後に接続相手から SSH_MSG_CHANNEL_EOF メッセージを受信し、受信キューにデータがなくなると、この API から 0 が返ります。

ssh_rcv_pkt

［機　能］ SSHパケットの受信　　（SSHサーバ/クライアント）

［形　式］ ER ssh_rcv_pkt(T_SSH_CH *ch, VP buf, INT len, TMO tmout);

ch	SSHチャネル制御ブロックへのポインタ
buf	受信データを入れる領域へのポインタ
len	受信データを入れる領域のサイズ
tmout	タイムアウト指定

［戻り値］

正の値	正常終了(受信したデータの長さ)
0	データ終結(SSH_MSG_CHANNEL_EOF メッセージを受信)
E_QOVR	受信キューのオーバフロー
E_OBJ	SSH コネクション制御ブロックが不正
EV_SSHUNEX	接続相手から予期しないメッセージを受信
EV_SSHMSG	接続相手から受信したメッセージを解析できない
EV_SSHMAC	受信したメッセージの MAC(メッセージ認証コード)が不正
E_TMOUT	タイムアウト
その他	tcp_rcv_dat のエラー

［解　説］ TCP と同様に SSH でもプロトコル上はデータの区切りがなく、相手が送信した単位でそのまま受信できる保証はありませんが、コマンドに対して応答があるようなデータが途切れることで区切りと判断できる場合に利用できる API です。

ssh_rcv_dat()との違いは、len で指定した長さを越えない範囲で、SSH パケットの受信処理が内部で繰り返し実行されることです。tmout のタイムアウト指定とタイムアウトエラーは最初の SSH パケットの受信にのみ適用され、繰り返しの SSH パケットの受信にはデフォルトで 200msec のタイムアウトを指定し、タイムアウトしても正常にデータが途切れたと判断しています。

よって、受信するデータの長さが分かっている場合には、len に buf のサイズではなく、データ長を指定して最後の無駄なタイムアウトを防いでください。また、受信したデータの長さが想定よりも短かった場合や不完全だった場合は、この API を繰り返し呼び出して残りを受信してください。

ssh_sht_ses

［機　能］ SSH データ送信の終了 （SSH サーバ/クライアント）

［形　式］ ER ssh_sht_ch(T_SSH_CH *ch, TMO tmout);

ch	SSH チャネル制御ブロックへのポインタ
tmout	タイムアウト指定

［戻り値］

E_OK	正常終了
E_PAR	不正な引き数
E_NOMEM	SSH コネクションの送信バッファのサイズが不足
E_OBJ	SSH コネクション制御ブロックが不正
E_TMOUT	タイムアウト
その他	tcp_snd_dat のエラー

［解　説］ この API は送信バッファからデータが送信された後で、SSH_MSG_CHANNEL_EOF メッセージを送り、データ送信が終了したことを通知します。
タイムアウトなしで本 API を呼び出した場合、SSH_MSG_CHANNEL_EOF メッセージを送信バッファに準備できるまで待ち状態となります。 タイムアウトありで本 API を呼び出した場合、指定した時間が経過しても、送信準備が完了しなければ、E_TMOUT エラーが返ります。
この API を呼び出した後も接続相手からのデータ受信は可能な状態です。通常、それは不要ですので、この API の呼び出しは省いて、ssh_cls_ses()でセッションをクローズしてください。

ssh_cls_ses

［機　能］ SSH セッションの終了　　（SSH サーバ/クライアント）

［形　式］ ER ssh_cls_ses(T_SSH_CH *ch, TMO tmout);

ch	SSH チャネル制御ブロックへのポインタ
tmout	タイムアウト指定

［戻り値］

E_OK	正常終了
E_PAR	不正な引き数
E_NOMEM	SSH コネクションの送信バッファのサイズが不足
E_OBJ	SSH コネクション制御ブロックが不正
E_TMOUT	タイムアウト
その他	tcp_snd_dat のエラー

［解　説］ このAPIは送信バッファからデータが送信された後で、SSH_MSG_CHANNEL_CLOSE メッセージを送信します。
タイムアウトなしで本 API を呼び出した場合、SSH_MSG_CHANNEL_CLOSE メッセージを送信バッファに準備できるまで待ち状態となります。 タイムアウトありで本 API を呼び出した場合、指定した時間が経過しても、送信準備が完了しなければ、E_TMOUT エラーが返ります。
同じ SSH コネクション上で通信している他の SSH チャネルがなく、かつ、SSH セッションを再開しないのであれば、続けて ssh_cls_con()を呼び出して、SSH コネクションもクローズしてください。

ssh_cls_con

［機　能］ SSH コネクションの終了　　(SSH サーバ/クライアント)

［形　式］ ER ssh_cls_con(T_SSH_CON *con, TMO tmout);

ssh_con	SSH コネクション制御ブロックへのポインタ
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	不正な引き数
E_CLS	SSH コネクションが未接続状態

［解　説］ このAPIは送信バッファからデータが送信された後で、SSH_MSG_DISCONNECT メッセージを送受信してSSHコネクションを切断します。
タイムアウトなしで本 API を呼び出した場合、切断が完了するまで待ち状態となります。 タイムアウトありで本 API を呼び出した場合、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。
SSH セッションの実行中に各 API(ssh_snd_dat、ssh_snd_pkt、ssh_rcv_dat、ssh_rcv_pkt、ssh_sht_ses、ssh_cls_ses)でエラーが発生した場合、または相手から SSH コネクションが切断された場合は、この API により SSH コネクションの終了処理を行う必要があります。

ssh_ext

［機　能］ SSH スタックの終了 （SSH サーバ/クライアント）

［形　式］ ER ssh_ext(void);

［戻り値］ E_OK　　正常終了

［解　説］ SSH スタック内部で使用しているメモリプールを削除します。サーバに未解放のホスト鍵がある場合は、解放処理を行います。この API を呼び出す前に、全ての SSH セッションと SSH コネクションをクローズしておいてください。
この API を発行した後で、再び SSH を使用する場合は、SSH スタックの初期化 ssh_ini()を呼び出してください。

SSH コールバックルーチン

［機　能］ SSH のイベント通知　　(SSH クライアント)

［形　式］ ER callback(FN evcd, VP parblk, int len);
evcd　イベントコード
parblk　パラメータが格納されている領域へのポインタ
len　パラメータの長さ

［戻り値］ E_OK　正常終了
負の値　異常終了

［解　説］ SSH の処理中に発生したイベントを受け取ります。 evcd に渡されるイベントは下記の 2 つで、いずれも SSH コネクションの確立中にクライアント側で発生するものです。

TEV_HSTKEY_CHK　サーバの公開ホスト鍵を受信
TEV_USER_BANNER　バナーメッセージを受信

なお、上記の形式で callback と表現されているのは、ssh_set_opt()でコールバックルーチンとして登録されるユーザー作成の関数で、名前は任意です。
このコールバックルーチンは、SSH セッションを開始する ssh_opn_ses()の内部から呼び出されますので、矛盾する SSH スタックの他の API や TCP/IP API、および応答に影響する待ちが生じるシステムコールを、ここでは実行しないでください。

TEV_HSTKEY_CHK では、受信した公開ホスト鍵が parblk で渡されますので、検証を行ってください。その実装方法は「第 8 章 公開ホスト鍵の検証」を参照してください。

TEV_USER_BANNER では、ログイン時の SSH サーバからのメッセージが parblk で渡されますので、必要ならば表示を行ってください。

このコールバックルーチンから E_OK でリターンすれば SSH コネクションの確立処理が続行され、負数でリターンすればそれが中断されます。

第 5 章 SSH サーバのサンプル

5.1 はじめに

SSH サーバの実装例 nonsshd.c が、NORTi¥NETSMP¥SRC フォルダに収録されており、SSH スタックの API を使用して SSH クライアントからのコマンドを受け取り、実行結果を返します。これを利用する場合は、次ページ以降の API を呼び出すアプリケーションで nonssh.h および nonteln.h をインクルードし、nonsshd.c をプロジェクトに加えてビルドしてください。
また、Telnet サーバ用として nonshel.c に実装されているシェルタスクや、nonteld.c に実装されている print、puts、rputs、putch、input、gets、getch、kbhit などの入出力関数を利用しているため、nonshel.c および nonteld.c もプロジェクトに加えてビルドしてください。

5.2 コンフィグレーション

SSH サーバのサンプルでは、Telnet サーバと同様に、タスク、メッセージバッファ、TCP 通信端点、TCP 受付口それぞれ 1 個を使用しますので、カーネルおよび TCP/IP スタックのコンフィグレーションでは、それらの ID の上限値を＋1 してください。
各評価ボード用の SSH 対応のサンプルプログラム ssh???.c には、SSH サーバへのログインに使用するユーザー名、およびパスワードまたは公開鍵を下記のように仮に定義してありますので、実際に使用するユーザー名/パスワード/公開鍵に変更してください。

```
static const T_SSH_LOGIN usrpkey[] = {
{ /* either password or public key is used for user authentication */
  "mispo", "12345",
"ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPYhtY7wRdawSaV8VLuqWHcNYjJDAOS/b¥
uQFce8b+AVz" },
        :
};
```

また、sshdkeys.h には、サーバのホスト鍵ペアのリストを下記のように仮に定義してありますので、実際に使用するホスト鍵のリストに置き換えてください。

```
static const T_SERV_KEYS svr_keys[] = {
{
"ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAA¥
ABBBBOwtRCFMUVtLMtfg/qV/AZaUyVEBsuR5dp8A3I9BdTKk9XoIPDFNs6qQhjfHG3b1XD¥
mdBpe3Hb3pQKFls+t5A8=",
        :
};
```

5.3 SSH サーバの API

sshd_ini

［機　能］ SSH サーバの初期化

［形　式］ ER sshd_ini(T_SSHD *sshd, ID tskid, ID mbfid, VP keys, int kno);

sshd	SSH サーバ制御ブロックへのポインタ
tskid	タスク ID(自動割り当ての場合は 0)
mbfid	メッセージバッファ ID(自動割り当ての場合は 0)
keys	ホスト鍵ペアの配列を格納した領域へのポインタ
kno	上記の配列に格納されているホスト鍵ペアの数

［戻り値］

E_OK	正常終了
E_PAR	不正な引き数
E_ID	ID番号が不正または不足
E_OBJ	タスクまたはメッセージバッファが生成済み

［解　説］ アプリケーション側で定義した SSH サーバ制御ブロックとホスト鍵ペアを指定して呼び出すと、SSH サーバ用のタスクとメッセージバッファを生成し、SSH サーバ制御ブロックを初期化します。初期化された制御ブロックを使って shell_ini()を呼び出してシェルタスクも初期化すると、SSH サーバが起動します。

なお、本 API を呼び出す前に ssh_ini()を実行して SSH スタックを初期化してください。

shell_ini

［機　能］　シェルタスクの初期化

［形　式］　ER shell_ini(VP t, ID tskid, ID mbfid, FP callback);

t	SSH サーバ制御ブロックへのポインタ
tskid	タスク ID(自動割り当ての場合は 0)
mbfid	メッセージバッファ ID(自動割り当ての場合は 0)
callback	ユーザー認証用コールバックルーチンのアドレス

［戻り値］

E_OK	正常終了
E_ID	ID 番号が不正または不足
E_OBJ	タスクが生成済み

［解　説］　本 API は、NORTi Version 4 の NETSMP¥SRC フォルダの nonshel.c に実装されています。(その改訂版が SSL for NORTi に収録さている場合もあります)
Telnet サーバ制御ブロックの代わりに、sshd_ini()で初期化した SSH サーバ制御ブロックとユーザー認証用コールバックルーチンを指定して本 API を呼び出すと、シェルタスクと共に SSH サーバが起動します。

sshd_callback

［機　能］ SSH コマンド処理のコールバックルーチン

［形　式］ BOOL sshd_callback(T_TERMINAL *t, char *s);
t　　入出力端末制御ブロックへのポインタ
s　　コマンドの文字列

［戻り値］ TRUE　　正常終了(SSH セッションは終了しません)
FALSE　　SSH セッションの終了が要求された

［解　説］ SSH サーバで SSH コマンドを受信した時に呼び出される関数を、この形式でアプリケーション側に用意してください。(各評価ボード用の SSH 対応のサンプルプログラム ssh???.c に例が実装されています)
s には SSH コマンドとして受信した文字列が渡されますので、それを解釈して対応した処理を行えます。処理の結果として TRUE を返すと SSH セッションが継続し、FALSE を返すと SSH セッションが終了します。
SSH スタック内部からではなく、nonshel.c のシェルタスクから呼び出さますので、この関数では待ち状態となる API も発行できます。
Telnet サーバで Telnet コマンドを受信した時と同様の動作で、SSH の場合、t には SSH サーバ制御ブロックへのポインタが渡されますが、それは通常、参照する必要がありません。

ユーザー認証コールバックルーチン

［機　能］　ユーザー名とパスワード/公開鍵でユーザー認証

［形　式］　ER sshd_login_check(const char *name, const char *pk, int klen);

name　　ユーザー名の文字列へのポインタ

pk　　パスワードの文字列、または公開鍵を格納した領域へのポインタ

klen　　公開鍵の長さ(パスワード認証の場合は 0)

［戻り値］　E_OK　　ログイン可

-1　　ログイン不可

［解　説］　SSH クライアントから受信したユーザー名とパスワードまたは公開鍵の照合を行うコールバックルーチンを上記形式の関数でアプリケーション側に用意し、shell_ini()の 4 番目の引数に指定してください。各評価ボード用の SSH 対応のサンプルプログラム ssh???.c では、この関数名で実装されていますが、名前は任意です。

この関数は SSH サーバへのログインの際に呼び出されて、E_OK でリターンすると SSH コネクションの確立処理が続行され、負の値を返すとそれが中断されます。

ユーザー認証がパスワード認証方式で行われる場合、pk には Null 文字'¥0'で終端されたパスワードの文字列と klen には 0 が渡されます。公開鍵認証方式の場合は、pk に公開鍵を格納した領域へのポインタと klen にその長さが渡されます。

この関数は、SSH セッションを開始する ssh_acp_ses()の内部から呼び出されますので、矛盾する SSH スタックの他の API や TCP/IP API、および応答に影響する待ちが生じるシステムコールを実行しないでください。

第 6 章 SSH クライアントのサンプル

6.1 はじめに

SSH クライアントの実装例 nonsshcs.c が、NORTi¥NETSMP¥SRC フォルダに収録されており、SSH スタックの API を使用して SSH サーバに対して SSH コマンドを送って結果を受け取ります。これを利用する場合は、次ページの API を呼び出すアプリケーションで nonssh.h をインクルードし、nonsshcs.c をプロジェクトに加えてビルドしてください。

6.2 コンフィグレーション

この SSH クライアントでは TCP 通信端点を 1 個使用しますので、TCP/IP スタックのコンフィグレーションでは、その ID の上限値を＋1 してください。
nonsshcs.c には、SSH サーバにログインするためのユーザー名とパスワードを下記のように仮に定義してありますので、実際に使用するユーザー名/パスワードに変更してください。
なお、SSH スタックの API にその機能はありますが、複数の SSH セッションを同時に実行することは、この SSH クライアントでは行っていません。

```
#ifndef SSH_UNAME
#define SSH_UNAME   "mispo"
#endif
#ifndef SSH_PSWD
#define SSH_PSWD    "12345"
#endif
```

6.3 SSH クライアントの API

ssh_command

［機　能］ SSH クライアントのコマンドを実行

［形　式］ int ssh_command(int argc, char *argv[]);
argc　コマンドライン引数の数
argv　コマンドライン引数の文字列へのポインタ配列

［戻り値］ 0　正常終了
負　内部で呼び出している SSH スタックの API のエラー

［解　説］ 各評価ボード用の SSH 対応のサンプルプログラム ssh???.c では、コンソールから次の形式のコマンドが入力されると、この API が呼び出されます。

ssh [-i <nif>] <ipaddr>
-i　ネットワークインタフェース名（例：-i eth1）
<ipaddr> SSH サーバの IP アドレス（例：192.168.1.3）

複数のネットワークインタフェース(Ethernet ポート)があって、デフォルトのネットワーク以外を使用する場合は、-i オプションでそれを指定できます。DNS を使える場合、IP アドレスには SSH サーバのホスト名も指定できます。

この API では、SSH サーバと接続して SSH セッションを開始し、コンソールから入力されたコマンドの文字列を SSH サーバへ送信し、SSH サーバから受信した応答の文字列をコンソールへ出力することを繰り返します。
そして、exit コマンドや logout コマンドでログアウトすると処理を終結して、リターンします。

第 7 章 ユーザー認証情報

SSH サーバでは、SSH コネクションの確立時のユーザー認証を、パスワード認証方式または公開鍵認証方式のいずれかで行います。そのため、あらかじめユーザー名とパスワード、もしくは、クライアントの公開鍵を、下記の構造体の配列に後述の例のように定義してください。

```
typedef struct {
    const char *uname;     ユーザー名
    const char *pswd;      パスワード      (公開鍵のみを使用する場合は NULL)
    const char *pubkey;    ユーザー公開鍵 (パスワードのみを使用する場合は NULL)
} T_SSH_LOGIN;
```

定義できる公開ホスト鍵は OpenSSH 形式の 1 行の文字列で､それをソースファイルにコピーする際に複数行となる場合は、改行位置に「¥」を付けてください。例えば、各評価ボード用の SSH 対応のサンプルプログラム ssh???.c には、以下の 3 組のユーザー名とパスワード/公開鍵を定義してあり、SSH サーバのコールバックルーチン(サンプルプログラムでは sshd_login_check())で照合を行っています。

```
static const T_SSH_LOGIN usrpkey[] = {
{ /* either password or public key is used for user authentication */
  "mispo", "12345",
"ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPYhtY7wRdawSaV8VLuqWHcNYjJDAOS/b¥
uQFce8b+AVz" },
{ /* only public key is used for user authentication */
  "mispo1", NULL,
"ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAA¥
ABBBNWH5UxhKAsnn1Dryzdu3iX4p5JuOYkWXKaarKympIu6dWjNrqBb+aeFb5GGZDaQDq+¥
xoL7Pr5kuI2QW6dFuZnO=" },
{ /* only password is used for user authentication */
  "mispo2", "12345", NULL }
};
```

コールバックルーチンには、SSH クライアントから受信したユーザー名とパスワード、あるいは、バイナリ形式から OpenSSH 形式に変換した公開鍵が渡されますので、このように定義してあったユーザー認証情報と比較してください。そして、コールバックルーチンから E_OK でリターンすれば接続となり、負数でリターンすれば接続は中断となります。

第 8 章 公開ホスト鍵の検証

SSH クライアントは、SSH コネクションの確立時に SSH サーバから受信した公開ホスト鍵で本物のサーバかを確認するホスト認証を行えます。そのため、照合用の公開ホスト鍵を、あらかじめ SSH クライアント側にも定義してください。
定義できる公開ホスト鍵は OpenSSH 形式の 1 行の文字列で､それをソースファイルにコピーする際に複数行となる場合は、改行位置に「¥」を付けてください。また、複数の公開ホスト鍵を定義する場合は「,」で区切ってください。
例えば、SSH クライアントの実装例 nonsshcs.c には、以下の 3 つの公開ホスト鍵を定義してあります。

```
const char *trustedkeys[] = {

/* Local Linux Server ECDSA NIST P-256 host key */
"ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAy¥
NTYAAABBBD5LJOqKTOOnbZC+uGoPb6HNDayX7zlmDbefmQa+uNO9AcfMbME6qqMwu¥
1C6/Z8zRHogdCJLOFFP5UFTOwqyc8A= root@mispo-Vostro-1520",

/* Test.Rebex.Net ECDSA NIST P-256 host key */
"ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAy¥
NTYAAABBBLZcZopPvkxYERubWeSrWOSHpxJdR14WFVES/Q3hFguTn6L+OEANqYcbR¥
XhGBUV6SjR7SaxZACXSxOzgCtG4kwc=",

/* Test.Rebex.Net Ed25519 host key */
"ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIOdXzF+Jx/wvEBun5fxi8FQK3Omi¥
LZFNDOrxkYwNcYlE"
};
```

照合を行うのは､SSH オプションの設定 ssh_set_opt()で登録しておいたコールバックルーチンの役割で、それが、SSH セッションの開始 ssh_opn_ses()で SSH コネクションの確立時に呼び出されます。コールバックルーチンには、SSH サーバから受信した公開ホスト鍵がバイナリ形式から OpenSSH 形式に変換されて渡されますので、上記のように定義してあった公開ホスト鍵と比較してください。そして、コールバックルーチンから E_OK でリターンすれば接続となり、負の値でリターンすれば接続は中断となります。
コールバックルーチンが登録されてない場合は、公開ホスト鍵の検証を行わずに SSH コネクションが確立します。

以下の nonsshcs.c に実装されているコールバックルーチンの例では、サーバから受信した公開ホスト鍵が trustedkeys[]に見つからない場合、その鍵を表示しています。

```
ER ssh_callback(FN evcd, VP p, int len)
{
    int i;

    switch (evcd) {
    case TEV_HSTKEY_CHK:
        for (i = 0; i < sizeof trustedkeys / sizeof (char *); i++) {
            if (memcmp(p, trustedkeys[i], len) == 0)
                break;
        }
        if (i < sizeof trustedkeys / sizeof (char *)) {
            ssh_print("Hostkey is trusted¥r¥n");
        } else {
            ssh_print("Hostkey is not found in trusted list¥r¥n");
            ssh_print(p);
            return -1;
        }
        break;
    case TEV_USER_BANNER:
        print(p);
        break;
    }
    return E_OK;
}
```

第 9 章 公開鍵/秘密鍵ペアの生成方法

SSH サーバ/SSH クライアントのサンプルの公開鍵と秘密鍵の生成には、ssh-keygen ユーティリティを利用しましたので、その基本的な使い方について説明します。Linux PC 上で、下記の最小限のオプションを指定すれば、OpenSSH 形式の鍵を生成できます。

オプション	説明
-t <type>	鍵の種類 (ecdsa、ed25519)
-b <bits>	鍵のビット数 (256、384、521)
-a <rounds>	KDF(鍵導出関数)の反復回数
-o	OpenSSH 形式
-Z <cipher>	鍵を暗号化する際の方式 (aes256-ctr)
-f <filename>	鍵のファイル名

(例 1) ecdsa-sha2-nistp256 の鍵ペアの生成

```
ssh-keygen -t ecdsa -b 256 -a 1 -o -Z aes256-ctr -f ecdsa256
```

上記のコマンドを実行すると、パスフレーズの入力が求められます。パスフレーズを入力すると保護された秘密鍵、リターンキーのみを押すとパスフレーズで保護されない秘密鍵が ecdsa256 というファイル名で、公開鍵が ecdsa256.pub というファイル名で生成されます。

同様に、以下のコマンドを実行すると、それぞれ指定したファイル名で鍵が生成されます。

(例 2) ecdsa-sha2-nistp384 の鍵ペアの生成

```
ssh-keygen -t ecdsa -b 384 -a 1 -o -Z aes256-ctr -f ecdsa384
```

(例 3) ecdsa-sha2-nistp521 の鍵ペアの生成

```
ssh-keygen -t ecdsa -b 521 -a 1 -o -Z aes256-ctr -f ecdsa521
```

(例 4) ssh-ed25519 の鍵ペアの生成

```
ssh-keygen -t ed25519 -a 1 -o -Z aes256-ctr -f ed25519
```

SSH スタック 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