wolfSSH Documentation

2025-04-22

Contents

1 イントロダクション

このマニュアルは組み込み用 wolfSSH ライブラリの技術解説書としてお読みいただけるように書かれています。wolfSSH をビルドして起動することから始まり、ビルドオプション、機能、サポートなどの概要を提供します。

wolfSSH は C 言語で書かれた SSH（セキュアシェル）サーバー実装で、wolfSSL から利用可能な wolfCrypt を使用します。さらに、マルチプラットフォームで使用できるようにゼロから構築されています。また、SSHv2 仕様に準拠しています。

1.1 プロトコル概要

SSH は 2 つの通信端点に多重化されたデータストリームを提供する一連の階層化されたプロトコルです。一般的には、サーバー上のシェルへの接続を保護するために利用されます。ですが、ファイルを安全にコピーしたり、X ディスプレイプロトコルをトンネリングするのにも利用されています。

1.2 wolfSSH をお勧めする理由

wolfSSH は ANSI C で記述された軽量の SSHv2 サーバーライブラリで、サイズが軽量でありスピード、機能セットに富んでいる点から、組み込み機器、リアルタイム OS およびリソース制約のある環境をターゲットにしています。wolfSSH は業界標準の SSH v2 をサポートし、さらに先進的なアルゴリズム（ChaCha20, Poly1305, NTRU と SHA-3) も提供しています。wolfSSH を支えているのは wolfCrypt 暗号化ライブラリで、このライブラリは FIPS140-2 認証（認証 #2425）を受けています。より詳細は wolfCrypt FIPS FAQ を参照されるかあるいは info@wolfssl.com までお知らせください。

1.2.1 機能（特徴）

- SSH v2.0 (サーバー機能)
- 最小フットプリント：33kB
- 実行時メモリ消費量：1.4KB ~ 2KB (受信バッファは含まず)
- ハッシュ関数: SHA-1, SHA-2 (SHA-256, SHA-384, SHA-512), BLAKE2b, Poly
- 暗号アルゴリズム：Block, Stream, and Authenticated Ciphers: AES (CBC, CTR, GCM, CCM), Camellia, ChaCha
- 公開鍵オプション: RSA, DH, EDH, NTRU
- ECDH と ECDSA で次の楕円曲線をサポート: NISTP256, NISTP384, NISTP, Curve25519, Ed
- クライアント認証をサポート (RSA key, password)
- シンプルな API
- PEM and DER certificate support
- ハードウエア暗号サポート:
  - Intel AES-NI support
  - Intel AVX1/2
  - RDRAND
  - RDSEED
  - Cavium NITROX
  - STM32F2/F4 ハードウエア暗号
  - Freescale CAU / mmCAU / SEC
  - Microchip PIC32MZ

2 wolfSSH のビルド

wolfSSH はポータビリティを念頭において開発されているので多くのシステム上に移植するのは容易にできるはずです。ですが、もし移植上で問題がありましたら https://www.wolfssl.com/forums を参照されるか support@wolfssl.com へ質問をお寄せください。

この章では wolfSSH を *nix システム（あるいはその派生システム）や Windows 上でビルドする方法を説明します。また、上記以外のシステムにおいてのビルド方法のガイダンスも提供します。次章では「サンプルプログラムを使って始めてみよう」を用意しています。

autoconf/automake システムを使ってビルドする際には wolfSSH は単一の Makefile によってすべてのコンポーネントとサンプルプログラムをビルドできます。Makefile を繰り返し使用する場合に比べてシンプルで早いです。

2.1 ソースコードの入手

最新バージョンのコードを入手する場合には次の GitHub サイトからダウンロードできます：https://github.com/wolfSSL/wolfSSH

“Download ZIP” ボタンをクリックするかターミナルを開いて次のコマンドを実行してください:

```
$ git clone https://github.com/wolfSSL/wolfssh.git
```

2.2 wolfSSH が依存するモジュール

wolfSSH は wolfCrypt に依存しているので、wolfSSL のコンフィギュレーションが必要となっています。wolfSSL はここからダウンロードできます: https://github.com/wolfSSL/wolfssl

最も簡潔な wolfSSH の構成のための wolfSSL のコンフィギュレーションを行うには wolfSSL のルートフォルダから以下のコマンドを実行します:

```
$ ./autogen.sh (GitHubからクローンした場合にのみ実行が必要)
$ ./configure --enable-ssh
$ make check
$ sudo make install
```

wolfSSH の鍵生成機能を利用する場合には `--enable-keygen` を追加してください。また、もし wolfSSL のコードが必要ない場合には `--enable-cryptonly` を追加してください。

上記により、wolfSSH の実行に必要な wolfSSL ライブラリがインストールされます。

2.3 ＊ nix システム上での wolfSSH のビルド

Linux, ＊ BSD, OS X, Solaris ＊ nix 類似のシステム上でビルドを行う場合には、autoconf システムを利用します。wolfSSH のビルドには以下のコマンドを実行します:

```
$ ./autogen.sh (GitHubからクローンした場合にのみ実行が必要)
$ ./configure
$ make
$ make install
```

configure コマンドにはオプションを追加することができます。追加可能なオプションとその用途は以下のコマンドで参照することができます:

```
$ ./configure --help
```

wolfSSH のビルドには以下を実行してください:

```
$ make
```

wolfSSH のビルドが正常に終了したことを確認する為に、以下のコマンドを実行して、全てのテストがパスすることを確認してください:

```
$ make check
```

以下を実行して wolfSSH をインストールします:

```
$ make install
```

インストールにはスーパーユーザー権限が必要なので、場合によっては以下の様に'sudo' コマンドを前置して実行する必要があるかもしれません:

```
$ sudo make install
```

場合によっては、wolfssh/src 以下の wolfSSH ライブラリだけをビルドし、その他のアイテム（サンプルプログラムやテスト）を除外したいかもしれません。その場合には wolfSSH のルートフォルダから以下のコマンドを実行してください:

```
$ make src/libwolfssh.la
```

2.4 Windows 上での wolfSSH のビルド

Visual Studio プロジェクトファイルは以下で取得できます: https://github.com/wolfSSL/wolfssh/blob/master/ide/winvs/

ソリューションファイル'wolfssh.sln' は wolfSSH, そのサンプルプログラムとテストプログラムをビルドするように構成されています。Debug ビルドと Release ビルドの構成をスタティックリンクライブラリとダイナミック（32/64 ビット）ライブラリの両形式で提供しています。user_settings.h は wolfSSL のコンフィギュレーションで必要となります。

このプロジェクトファイルでは wolfSSH と wolfSSL のソースフォルダ階層が隣同士に配置されていることを前提にしています。また、それらのルートフォルダにはバージョン番号が含まれていないフォルダ名となっていることを前提としています。つまり、次のようなフォルダ構成です:

```
Projects\
wolfssh\
wolfssl\
```

`wolfssh\ide\winvs\user_settings.h` ファイルは wolfSSL に対する設定も既に含んだ適切な内容となっています。このファイルを忘れずに `wolfssh\ide\winvs` フォルダから `wolfssl\IDE\WIN` フォルダにコピーしてください。もし、一方の内容を変更した場合には、その内容を他方にもコピーして下さい。

`WOLFCRYPT_ONLY` マクロ定義は wolfSSL コードをビルド対象から除外し、wolfCrypt のアルゴリズム部分のみをビルドするの為に指定してあります。もし、wolfSSL コードもビルドする場合にはこの定義を削除してください。

2.4.1 Windows 上でのビルドに使用するユーザーマクロ定義

ソリューションでは wolfSSL ライブラリとヘッダーファイルのロケーションを指定するためにユーザーマクロを利用します。wolfssl64 ソリューションでは全てのパスは既定のビルド出力先に設定されます。ユーザーマクロ'wolfCryptDir' はライブラリを検索するためのベースパスとして使用します。初期値として、`..\..\..\..\wolfssl` に設定されています。その後、例えば追加のインクルードファイル検索パスが追加される場合には、`$(wolfCryptDir)` に対して追加を行います。

wolfCryptDir パスはプロジェクトファイルからの相対位置で表せなければなりません。

```
wolfssh/wolfssh.vcxproj
unit-test/unit-test.vcxproj
```

そのほかのユーザーマクロは異なるビルドターゲットのためのディレクトリを表すために使用されます。例えば、wolfCryptDllRelease64 は次のフォルダを表します:

```
$(wolfCryptDir)\x64\DLL Release
```

このパスは echoserver サンプルプログラムのデバッグ環境設定で 64-bit DLL リリースビルド版の出力先を表現するのに次の様に使われます:

```
PATH=$(wolfCryptDllRelease64);%PATH%
```

echoserver プログラムをデバッガーを使って実行する際にはこの設定によって wolfSSL DLL がこのディレクトリから見つかります。

2.5 その他の環境上でのビルド

公式にはサポートしていませんが、wolfSSH を非標準の環境でビルドしたいお客様、特に組み込み機器向け環境でのビルドをご希望の方々をできるだけお手伝いしようとしています。以下はその際に理解しておいていただきたい点です:

1. ソースとヘッダーファイルは wolfSSH ダウンロードパッケージの階層構造に存在する必要があります。
2. いくつかのビルドシステムでは wolfSSH ヘッダーファイルの格納場所を明示的に指定することを求める場合があります。その格納場所は/wolfssh ディレクトリなので通常はディレクトリをインクルードファイルパスに追加することで解決します。
3. wolfSSH はコンフィギュレーションで指定されない限りリトルエンディアンをデフォルトにしています。ユーザーが使用している非標準環境では configure コマンドを使用していない場合で、ビッグエンディアンシステムに指定する場合には BIG_ENDIAN_ORDER マクロ定義が必要となります。
4. ライブラリをビルドしてみて何か問題が生じた場合には wolfSSL にお知らせください。サポートが必要な場合には、support@wolfssl.com 宛てにご連絡ください。

2.6 クロスコンパイル

組み込み機器開発環境ではクロスコンパイルを行います。そのための簡単な方法はライブラリをコンフィギュアシステムを使ってクロスコンパイルを行うことです。コンフィギュアシステムは Makefile を一つ生成し、それを使って wolfSSH をビルドします。

クロスコンパイルを行う際には、次の様にコンフィギュアを行うホストを指定する必要があります:

```
$ ./configure --host=arm-linux
```

さらにコンパイラ、リンカー等も指定する必要があるでしょう：

```
$ ./configure --host=arm-linux CC=arm-linux-gcc AR=arm-linux-ar RANLIB=arm-
    linux
```

クロスコンパイル用に wolfSSH を正しくコンフィギュレーションできた後は、標準の autoconf 作法にしたがってビルドとライブラリのインストールを行います:

```
$ make
$ sudo make install
```

ここでご紹介した以外の Tips をお持ちでしたらぜひ info@wolfssl.com まで お知らせください。

2.7 カスタムディレクトリへのインストール

wolfSSL をカスタムディレクトリへインストールする場合には次のようにしてください:

```
$ ./configure --prefix=`~`/wolfSSL
$ make
$ make install
```

上記コマンドによってライブラリを”~/wolfSSL/lib” に、インクルードファイルを”~/wolfssl/include” に配置するように指定します。wolfSSH をカスタムディレクトリに配置する場合には次の様にしてください:

```
$ ./configure  --prefix=`~`/wolfssh  --libdir=`~`/wolfssl/lib  --includedir
   =`~`/wolfssl/include
$ make
$ make install
```

上記パスがご自分の実際のディレクトリとマッチすることを確認して下さい。

3 始めよう

wolfSSH のダウンロードとビルドが終わったら、テストプログラムとサンプルプログラムが自動的に作成されているはずです。

3.1 テスト

3.1.1 wolfSSH ユニットテスト

wolfSSH のユニットテストは API の動作を確認するためのものです。ポジティブ/ネガティブの両テストケースが実行されます。テストはマニュアルで実行することができますが、他の処理の一部（例えば make check コマンド実行時）として実行される場合もあります。

全てのサンプルプログラムとテストは wolfSSH のホームディレクトリから実行されなければなりません。実行時に必要な各種証明書と鍵をプログラムが見つけることができるようにするためです。

ユニットテストをマニュアルで実行するには次のようにします:

```
$ ./tests/unit.test
```

あるいは

```
$ make check (autoconfが使われている場合)
```

3.1.2 テストに関する注記事項

レポジトリをクローンした後、テスト用の秘密鍵はユーザーにとってはリードオンリーになっていることを確認してください。そうなっていない場合は ssh_client サンプルプログラムは警告します。

```
$ chmod 0600 ./keys/gretel-key-rsa.pem ./keys/hansel-key-rsa.pem \
             ./keys/gretel-key-ecc.pem ./keys/hansel-key-ecc.pem
```

サンプルプログラム echoserver に対しての認証はパスワードあるいは公開鍵を使って行うことができます。パスワードを使う場合は次のコマンドを使ってください:

```
$ ssh_client -p 22222 USER@localhost
```

ここで USER としてのユーザーとそのパスワードとして次の２つのペアが使えます:

```
jill:upthehill
jack:fetchapail
```

公開鍵を使った認証を行う場合には次のコマンドを使います:

```
$ ssh_client -i ./keys/USER-key-TYPE.pem -p 22222 USER@localhost
```

ここで、USER の部分には gretel か hansel が指定できて、TYPE には rsa か ecc を指定します。

echoserver はその wsUserAuth コールバック関数に偽のアカウント (jack, jill, hansel, と gretel）を用意してあります。後述するシェルサポートが有効になっている場合には、これらの偽アカウントは機能しません。これらのアカウントを使って認証を試みてもサーバーにはシステムのパスワードファイルにこれらのアカウントのおパスワードは存在していないので認証に失敗します。新たなユーザーとパスワードあるいは公開鍵リストを echoserver に追加することができます。追加されたアカウントでは、echoserver によって起動されたシェルに echoserver を起動したユーザー権限でログインすることができます。

3.2 サンプルプログラム

3.2.1 wolfSSH echoserver

echoserver サンプルプログラムは wolfSSH のサンプルプログラム中で最も多くの処理をこなすプログラムです。用意されているアカウントを認証することを許された唯一のユーザーであり、入力された文字を繰り返し出力します。後の章で説明するシェルサポートが有効になっている場合には、ユーザーシェルを起動することができます。echoserver の実行にはマシン上での実際のユーザ名とクレデンシャルを検証する為の更新した認証コールバック関数を必要とします。

ターミナルから次のコマンドを事項してください:

```
$ ./examples/echoserver/echoserver -f
```

-f オプションはエコーバックだけを行うモードを指定します。別のターミナルを開いて次のコマンドを実行してください:

```
$ ssh_client jill@localhost -p 22222
```

パスワードの入力を求められたら “upthehill” と入力してください。サーバーは次のバナーを返信してくるはずです:

```
    wolfSSH Example Echo Server
```

ssh_client にタイプした文字はサーバーからエコーバックされて表示されます。入力した文字が２度スクリーンにエコーバックされたとしたらそれはローカルのエコーバックが有効になっているからです。echoserver は正規のターミナルではないので、CR/LF　改行の変換が期待通りに機能しないかもしれません。

以下の制御文字は echoserver で特別な動作を引き起こします:

- CTRL-C:　コネクションを切断
- CTRL-E:　セッション状況をプリントアウト
- CTRL-F:　新たな鍵交換をトリガー

echoserver サンプルプログラムには以下のコマンドラインオプションが指定できます:

```
    -1              一回の接続後に終了する
    -e              クライアントからECC公開鍵を受け取る
    -E              ECC秘密鍵を使う
    -f              入力をエコーする
    -p <num>        待ち受けポート番号を指定する（デフォルトは22222)
    -N              ノンブロッキングソケットを使う
    -d <string>     SFTPコネクションのホームディレクトリを指定する
    -j <file>       接続相手からの公開鍵を受け付ける為にロードする
```

3.2.2 wolfSSH Client

このクライアントプログラムはは SSH サーバーと接続を確立します。簡単モードでは “Hello, wolfSSH!” をサーバーに送信し、サーバーからの応答を表示して終了します。疑似ターミナルオプションではこのクライアントプログラムは実際のクライアントとして機能します。

クライアントサンプルプログラムには以下のコマンドラインオプションが指定できます:

```
    -h <host>       接続先ホストアドレス（デフォルト 127.0.0.1）
    -p <num>        接続先ポート（デフォルト 22222)
    -u <username>   認証の為のユーザー名（指定必須）
    -P <password>   パスワード（省略した場合はプロンプトが表示される）
    -e              サンプルecc公開鍵を指定
```

```
-i <filename>  ユーザーの秘密鍵ファイル名
-j <filename>  ユーザーの公開鍵ファイル名
-x             接続完了後、データ送受信することなく終了
-N             ノンブロッキングソケットを使う
-t             疑似ターミナルを使用
-c <command>   リモートコマンドとpipe stdin/stdout を使用する
-a             SSH-AGENTの使用を試みる
```

3.2.3 wolfSSH portfwd

portfwd サンプルプログラムは SSH サーバーと接続を確立し、ローカルポートフォワーディングのための待ち受けポートをリスンするかあるいはリスンしているリスナーに対してリモートポートフォワーディングを要求します。接続確立の後はプログラムは終了します。

portfwd サンプルプログラムには以下のコマンドラインオプションが指定できます:

```
-h <host>      接続先SSHサーバーアドレス（デフォルト 127.0.0.1）
-p <num>       接続先SSHサーバーポート（デフォルト 22222）
-u <username>  ユーザー名(指定必須)
-P <password>  パスワード（省略した場合はプロンプトが表示される）
-F <host>      フォーワード元ホストアドレス（デフォルト 0.0.0.0）
-f <num>       フォーワード元ホストポート（指定必須）
-T <host>      フォーワード先ホストアドレス(デフォルト host)
-t <num>       フォーワード先ホストポート（指定必須）
```

3.2.4 wolfSSH scpclient

scpclient と wolfscp は SSH サーバーと接続を確立し、指定されたファイルをローカルマシンにコピー、あるいはローカルマシンのファイルをサーバーにコピーします。

scpclient サンプルプログラムには以下のコマンドラインオプションが指定できます:

```
-H <host>      接続先SSHサーバーアドレス（デフォルト 127.0.0.1）
-p <num>       接続先SSHサーバーポート（デフォルト 22222）
-u <username>  ユーザー名(指定必須)
-P <password>  パスワード（省略した場合はプロンプトが表示される）
-L <from>:<to> ローカルマシンのfromからサーバーのtoへコピーする
-S <from>:<to> サーバーのfromからローカルマシンのtoへコピーする
```

4 wolfSSH sftpclient

sftpclient, wolfsftp は SSH サーバーと接続を確立し、ディレクトリ移動、ファイル取得、ファイル配置、ディレクトリ追加・削除等を実行します。

sftpclient サンプルプログラムには以下のコマンドラインオプションが指定できます:

```
-h <host>       接続先SSHサーバーアドレス（デフォルト 127.0.0.1）
-p <num>        接続先SSHサーバーポート（デフォルト 22222）
-u <username>   ユーザー名(指定必須)
-P <password>   パスワード（省略した場合はプロンプトが表示される）
-d <path>       ローカルマシンのデフォルトのパスを設定
-N              ノンブロッキングソケットを使う
-e              ECC公開鍵を使ってユーザー認証を行う
-l <filename>   ローカルファイル名
-r <filename>   リモートファイル名
-g              ローカルファイルをリモートファイルとして送信
-G              リモートファイルをローカルファイルとして受信
```

4.0.1 wolfSSH サーバー

server はプレースホルダーとして存在しています。

4.1 SCP

wolfSSH は scp の為のサーバー側サポート（サーバーへのファイルコピーとサーバーからのファイルのコピーの両方）を含んでいます。単一ファイルのコピーとディレクトリ単位の再帰的コピーの両方をデフォルトの送信コールバックあるいは受信コールバックでサポートしています。

wolfSSH を scp サポート機能を有効にしてコンパイルするには,`--enable-scp` ビルドオプションを指定するかあるいは `WOLFSSL_SCP` マクロ定義を指定してください：

```
$ ./configure --enable-scp
$ make
```

wolfSSH サンプルサーバープログラムは単一の scp リクエストを受け付けるように設定されていて wolfSSH ライブラリをビルドする際にデフォルトでビルドされます。サンプルサーバーを起動するには以下を実行してください:

$ ./examples/server/server

標準 scp コマンド群はクライアント側で利用されます。以下はその使用例です。ここで、`scp` は使用している ssh クライアントを表します。

単一ファイルをサーバーに送信する場合で既定のユーザー “jill” を使うとすると:

```
$ scp -P 22222 <local_file> jill@127.0.0.1:<remote_path>
```

同じ単一ファイルをサーバーに送信する場合で、今度はタイムスタンプを使いバーバスモードを使うとすると:

```
$ scp -v -p -P 22222 <local_file> jill@127.0.0.1:<remote_path>
```

あるディレクトリを再帰的にサーバーに送信する場合には：

```
$ scp -P 22222 -r <local_dir> jill@127.0.0.1:<remote_dir>
```

単一ファイルをサーバーからローカルマシンにコピーするには:

```
$ scp -P 22222 jill@127.0.0.1:<remote_file> <local_path>
```

サーバーのあるディレクトリを再帰的に受信する場合には：

```
$ scp -P 22222 -r jill@127.0.0.1:<remote_dir> <local_path>
```

4.2 シェルサポート

wolfSSH の echoserver サンプルプログラムはログインを試みるユーザーの為にシェルを起動することができます。この機能は Linux と macOS でのみテスト済みです。echoserver.c ファイルはユーザー認証コールバック内にユーザーのクレデンシャルを保持するように変更が必要です。あるいはユーザー認証コールバックは提供されたパスワードを検証するように変更する必要があります。

wolfSSH をシェルサポート機能付きでビルドする場合には–enable-shell オプションを指定するかあるいは WOLFSSH_SHELL マクロ定義を指定します：

```
$ ./configure --enable-shell
$ make
```

デフォルトで echoserver はシェルを実行しようと試みます。エコーバックの機能をテストしたい場合にはコマンドラインオプションで-f を指定してください:

```
$ ./examples/echoserver/echoserver -f
```

5 ライブラリ設計

wolfSSH ライブラリはアプリケーションに直接含まれることを意図して設計されていま す。開発を行う上で念頭に置いている主な使用例は、組み込みデバイスのシリアルまたは telnet ベースのメニューを置き換えることです。このライブラリーは、入出力コールバックを使 用しネットワーキング層の違いに依存しないようになっていますが、デフォルトで * NIX および Windows ネットワーキング用のコールバックを例として提供しています。時計機能は プラットフォームによって異なるので、アプリケーションによって提供される前提の設計となっています。ライブラリー側ではタイムアウト時にアクションを実行するための機能が提供され ます。

5.1 ディレクトリ構成

wolfSSH ライブラリのヘッダーファイルは **wolfssh** ディレクトリにあります。ソースファイルでインクルードが必要となるファイルは唯一 **wolfssh/ssh.h** です。

```
#include <wolfssh/ssh.h>
```

wolfSFTP ライブラリヘッダファイルも wolfssh ディレクトリに含まれています。このヘ ッダファイルをインクルードするには：

```
#include <wolfssh/wolfsftp.h>
```

すべてのメインソースファイルは、ルートディレクトリにある **src** ディレクトリにありま す。

6 wolfSSH ユーザ認証コールバック

wolfSSH はライブラリがどのような環境に組み込まれていようが、サーバに接続しようとするユーザ を認証できなければなりません。テキストファイル、データベース、またはアプリケーシ ョンにハードコードされているパスワードまたは RSA 公開鍵を使用してユーザーを検索/認証する必要があります。

wolfSSH は、ユーザ認証メッセージで提供されたユーザ名、パスワードまたは公開鍵、および要求された認証タイプを受け取るコールバックフックを提供します。その後、コー ルバック関数は適切な検索を実行して応答を返します。ユーザはそのためのコールバック を提供する必要があります。

コールバック関数は失敗を示すエラーコードあるいは成功のいずれかを返さなければなりません。ライブラリは全ての失敗をロギング目的を除いて同一に扱います。すなわち、ユーザー認証失敗メッセージを再試行するクライアントに返信します。

パスワード検索を行う場合には平文のパスワードがコールバック関数に渡されます。ユーザー名とパスワードは一致するか検査され成功を返します。成功時には SSH ハンドシェークはただちに続行されます。現時点ではパスワードの変更はサポートされません。

公開鍵検索では、クライアントからの公開鍵 blob(バイナリデータ) がコールバック関数に渡されます。公開鍵はサーバーの有効なクライアント公開鍵のリストと照合されます。提供された公開 鍵がそのユーザーの既知の公開鍵と一致する場合。wolfSSH ライブラリは RFC4252§7 に記述されたプロセスに従ってユーザー認証署名の実際の検証を実行しま す。

一般に公開鍵の場合、サーバーは ssh-keygen ユーティリティによって生成されたユーザ ーの公開鍵を保存するか、または公開鍵のフィンガープリントを保存します。ユーザーに とってのこの値は比較の対象になるものです。クライアントはその秘密鍵を使用してセッション ID の署名とユーザー認証要求メッセージを提供します。サーバーは公開鍵を使用してこの 署名を検証します。

6.1 コールバック関数プロトタイプ

ユーザ認証コールバック関数プロトタイプは次の通りです：

```
int UserAuthCb(byte authType , const WS_UserAuthData* authData , void* ctx );
```

この関数プロトタイプのタイプは：

```
WS_CallbackUserAuth
```

パラメータ authType は:

```
WOLFSSH_USERAUTH_PASSWORD
```

か、あるいは

```
WOLFSSH_USERAUTH_PUBLICKEY
```

です。

パラメータ authData は認証データへのポインタです。

WS_UserAuthData の詳細は 5.4 を参照してください。

パラメータ **ctx** はアプリケーション定義のコンテキストです。wolfSSH はコンテキスト 内のデータについては何の知識も持たず何も操作しません。コールバック関数へのコンテキストポイ ンタを提供するだけです。

6.2 コールバック関数の認証タイプ定数

以下はユーザー認証コールバック関数に渡される authType パラメータの値です。この値によってコールバック関数に認証データのタイプを伝えます。システムはユーザー毎にパスワードか公開鍵かの利用を指定することができます。

```
WOLFSSH_USERAUTH_PASSWORD
WOLFSSH_USERAUTH_PUBLICKEY
```

6.3 コールバック関数の戻り値定数

以下は、コールバック関数がライブラリに返すリターンコードです。失敗コードはコー ルバックはチェックを行うことができず、何もしなかったことを示します。

以下のコードはユーザー認証が拒否された理由を示しています：

```
invalid username
invalid password
invalid public key
```

ライブラリはクライアントに成功または失敗のみを示し、下記の特定の失敗タイプはロギングに のみ使用されます。

```
WOLFSSH_USERAUTH_SUCCESS
WOLFSSH_USERAUTH_FAILURE
WOLFSSH_USERAUTH_INVALID_USER
WOLFSSH_USERAUTH_INVALID_PASSWORD
WOLFSSH_USERAUTH_INVALID_PUBLICKEY
```

6.4 コールバック関数のデータタイプ

クライアントデータは、WS_UserAuthData という構造体でコールバック関数に渡され ます。メッセージ内のデータへのポインタが含まれています。このフィールドには共通フィールドと UNION フィールドをメンバに持っています。メソッド固有のフィールドは、ユーザー認証データ内の UNION フィールドにあります。

```
typedef struct WS_UserAuthData {
    byte authType ;
    byte* username ;
    word32 usernameSz ;
    byte* serviceName ;
    word32 serviceNameSz ; n
    union {
        WS_UserAuthData_Password password ;
        WS_UserAuthData_PublicKey publicKey ;
    } sf;
} WS_UserAuthData;
```

6.4.1 パスワード

username および usernameSz パラメータは、クライアントによって提供されるユーザ名とオクテット単位のサイズです。

password フィールドと passwordSz フィールドは、クライアントのパスワードとそのオクテット単位のサイズです。

クライアントから提供された場合は設定されますが、パラメータ hasNewPassword、newPassword、および newPasswordSz は使用されません。現時点でクライアントにパスワードを変更するように指示するメカニズムはありません。

```
typedef struct WS_UserAuthData_Password {
    uint8_t* password ;
    uint32_t passwordSz ;
    uint8_t hasNewPasword ;
    uint8_t* newPassword ;
    uint32_t newPasswordSz ;
} WS_UserAuthData_Password;
```

6.4.2 公開鍵

wolfSSH は複数の公開鍵アルゴリズムをサポートします。publicKeyType メンバは、使用されているアルゴリズム名を指します。

publicKey フィールドは、クライアントから提供された公開鍵 blob を指します。

公開鍵の確認には 1 つまたは 2 つのステップがあります。hasSignature フィー ルドが設定されていない場合、署名はありません。ユーザー名と publicKey が正しいこ とを確認してください。この手順は、クライアントの設定によってはオプションであ り、無効なユーザーによる高価な公開鍵操作の実行を防ぐことができます。次に、hasSignature フィールドが設定され、signature フィールドがクライアントの署名を指 します。また、ユーザー名と publicKey も確認する必要があります。wolfSSH は自動 的に署名を確認します。

各フィールドはオクテットのサイズ値を持ちます。

```
typedef struct WS_UserAuthData_PublicKey {
    byte* publicKeyType;
    word32 publicKeyTypeSz;
    byte* publicKey;
    word32 publicKeySz;
    byte hasSignature;
    byte* signature;
    word32 signatureSz;
} WS_UserAuthData_PublicKey;
```

7 コールバック関数設定 API

以下の関数を使って、ユーザー認証コールバック関数の設定を行います。

7.1 ユーザ認証コールバック関数の設定

```
void wolfSSH_SetUserAuth(WOLFSSH_CTX* ctx , WS_CallbackUserAuthcb);
```

コールバック関数は、wolfSSH セッションオブジェクトを作成するために使用される WOLFSSH_CTX オブジェクトに設定されます。この CTX を使用するすべてのセッション は同じコールバック関数を使用します。このコンテキストは、コールバック関数のコン テキストと混同しないでください。

7.2 ユーザ認証コールバックコンテクストデータの設定

```
void wolfSSH_SetUserAuthCtx(WOLFSSH* ssh , void* ctx);
```

それぞれの wolfSSH セッションはそれ自身のユーザ認証コンテキストデータを持ってい るか、あるいはいくつかを共有することもできます。wolfSSH ライブラリはこのコンテ キストデータの内容について何も感知しません。データの作成、解放、および必要に応 じた排他制御の提供は、アプリケーションの責任です。コールバックはライブラリから このコンテキストデータを受け取ります。

7.3 ユーザ認証コールバックコンテクストデータの取得

```
void* wolfSSH_GetUserAuthCtx(WOLFSSH* ssh);
```

提供された wolfSSH セッションに保存されたユーザ認証コンテキストデータへのポイン タを返します。これはセッションを作成するために使用される wolfSSH のコンテキスト データと混同しないよう注意してください。

7.4 Echoserver サンプルプログラムのユーザ認証

サンプルの echoserver は、パスワードと公開鍵を使用してサンプルユーザーとの認証コ ールバックを実装しています。コールバックの例と wsUserAuth は、wolfSSH コンテキ ストに設定されています:

```
wolfSSH_SetUserAuth(ctx, wsUserAuth);
```

パスワードファイルの例（passwd.txt）は、コロンで区切られたユーザー名とパスワー ドの単純なリストです。このファイル内に存在するデフォルトは次のとおりです:

```
jill:upthehill
jack:fetchapail
```

公開鍵ファイルは、ssh-keygen を実行して得た公開鍵出力を 2 つ連結したものです。

```
ssh-rsa AAAAB3NzaC1yc...d+JI8wrAhfE4x hansel
ssh-rsa AAAAB3NzaC1yc...UoGCPIKuqcFMf gretel
```

すべてのユーザー認証データは、ユーザー名と、パスワードまたは公開鍵 blob の SHA-256 ハッシュのペアをリンクリスト形式で格納されています。

設定ファイル内の公開鍵 blob は Base64 エンコードされており、ハッシュ前にデコードされます。ユーザ名 - ハッシュペアのリストへのポインタは新しい wolfSSH セッションに保存されます。

```
wolfSSH_SetUserAuthCtx(ssh, &pwMapList);
```


コールバック関数は、最初に authType が公開鍵かパスワードかを調べ、そうでない場合は一般ユーザー認証失敗エラーコードを返します。次に、authData を介して渡された公開鍵またはパスワードをハッシュします。ユーザー名をリスト中から検索し見つけられない場合は無効ユーザーエラーコードを返します。ユーザー名が見つかった場合には、渡された公開鍵またはパスワードの計算ハッシュとペアに格納されているハッシュを比較します。一致した場合、関数は成功を返します。それ以外の場合、無効なパスワードまたは公開鍵 のエラーコードを返します。

8 wolfSSH SFTP のビルドと使用

8.1 wolfSSH SFTP のビルド

wolfSSL は既に wolfSSH の使用のためにビルドが済んでいると仮定しています。wolfSSL のビルド方法については２章を参照してください。

SFTP サポート機能を有効にして wolfSSH をビルドする場合には、autotools を使ったビルドのビルドでは–enable-sftp オプションを指定します。autotools を使わない場合には WOLFSSH_SFTP マクロ定義を指定します。コマンドラインは次のようになります:

```
$ ./configure --enable-sftp && make
```

リード・ライトをハンドリングするためのバッファサイズはデフォルトで 1024 バイトです。この値はアプリケーションがより少ないリソース消費に抑えたい場合やより大きなバッファが必要な場合には変更することができます。サイズ変更は WOLFSSH_MAX_SFTP_RW マクロを定義して行います。設定例は:

```
$ ./configure --enable-sftp C_EXTRA_FLAGS="WOLFSSH_MAX_SFTP_RW=2048"
```

8.2 wolfSSH SFTP アプリケーションの使用

SFTP サーバーとクライアントアプリケーションは woflSSH にバンドルされています。両アプリケーションとも autotools を使って wolfSSH ライブラリを SFTP サポートを有効にしてビルドする際に同時にビルドされて生成されます。クライアントアプリケーションは wolfsftp/client フォルダに存在しており wolfsftp と呼ばれます。

サーバーの起動例を示します。起動すると SFTP クライアントからの接続を待ち受けます:

```
$ ./examples/echoserver/echoserver
```

ここで、コマンドはルート wolfSSH ディレクトリから実行します。サーバーは SSH と SFTP コマンドの両方を処理することができます。

一方、クライアントを起動するには特定のユーザー名を与えて起動します:

```
$ ./wolfsftp/client/wolfsftp -u <username>
```

デフォルトの “username:password” は “jack:fetchapail” または “jill:upthehill” を与えます。デフォルトのポートは 22222 です。

サポートしているコマンドの全リストは接続後に、“help” と入力すると得られます。

```
wolfSSH sftp> help

Commands :
    cd  <string>                    change directory
    chmod <mode> <path>             change mode
    get <remote file> <local file>  pulls file(s) from server
    ls                              list current directory
    mkdir <dir name>                creates new directory on server
    put <local file> <remote file>  push file(s) to server
    pwd                             list current path
    quit                            exit
    rename <old> <new>              renames remote file
    reget <remote file> <local file>  resume pulling file
    reput <remote file> <local file>  resume pushing file
    <crtl + c>                      interrupt get/put cmd
```

他のシステムへの接続例は:

```
src/wolfssh$ ./examples/sftpclient/wolfsftp -p 22 -u user -h 192.168.1.111
```

9 ポートフォワーディング

9.1 wolfSSH をポートフォワーディング機能を有効にしてビルド

wolfSSL は既に wolfSSH の使用のためにビルドが済んでいると仮定しています。wolfSSL のビルド方法については２章を参照してください。

ポートフォワーディング機能を有効にして wolfSSH をビルドする場合には、autotools を使ったビルドのビルドでは–enable-fwd オプションを指定します。autotools を使わない場合には WOLFSSH_FWD マクロ定義を指定します。コマンドラインは次のようになります:

```
$ ./configure --enable-fwd && make
```

9.2 wolfSSH ポートフォワーディングサンプルプログラムを使用する

portfwd サンプルプログラムは “direct-tcpip” スタイルのチャネルを生成します。この例では OpenSSH サーバーがポートフォワーディング機能を有効にした状態でバックグラウンドで実行中である前提としています。このサンプルプログラムは wolfSSL クライアントのためにポートをフォワーディングしてサーバーとの通信を仲介します。すべてのプログラムは同一マシン上で動作しているもとの前提です。

```
src/wolfssl$ ./examples/server/server
src/wolfssh$ ./examples/portfwd/portfwd -p 22 -u <username> \
             -f 12345 -t 11111
src/wolfssl$ ./examples/client/client -p 12345
```

既定で wolfSSL サーバーはポート 11111 でリスンします。クライアントはポート 12345 に接続を試みるように設定されています。portfwd は username で OpenSSH サーバーにログインし、自身はポート 12345 でリスンを開始しつつ、ポート 11111 でリスンしているサーバーに SSH サーバー経由で接続を行います。結果としてパケットはクライアントとサーバーの間でルーティングされることになります。

portfwd サンプルプログラムのソースファイルは wolfSSH でのポートフォワーディング機能の利用と設定方法を示す良い例となるはずです。

echoerver サンプルプログラムはローカルポートフォワーディングとリモートポートフォワーディングを扱います。ssh ツールに接続するには以下のいずれかのコマンドを実行してください。コマンド実行はどのマシンからでも実行できます。

```
src/wolfssl$ ./examples/server/server
src/wolfssh$ ./examples/echoserver/echoserver
anywhere 1$ ssh -p 22222 -L 12345:localhost:11111 jill@localhost
anywhere 2$ ssh -p 22222 -R 12345:localhost:11111 jill@localhost
src/wolfssl$ ./examples/client/client -p 12345
```

上記実行により、wolfSSL クライアントとサーバーサンプルプログラム間で portfwd サンプルプログラムと同様のポートフォワーディングを行います。

10 メモと制限事項

実装ファイル属性の一部は考慮されておらず、デフォルトの属性またはモード値が使用されます。特に wolfSSH_SFTP_Open では、ファイルからタイムスタンプを取得し、すべての拡張ファイル属性を取得します。

11 ライセンス

11.1 オープンソース

wolfSSL(以前の Cyassl)、Yassl、wolfCrypt、Yassh、Theocrypt ソフトウェアはフリーソフトウェアのダウンロードであり、ユーザーが GPL ライセンスのバージョンに準拠する限り、ユーザーのニーズに変更される場合があります。GPLV2 ライセンスは、gnu.org Web サイト (http://www.gnu.org/licenses/old-licenses/gpl-2.0.html) にあります。

wolfSSH ソフトウェアはフリーソフトウェアのダウンロードであり、ユーザーが GPL ライセンスのバージョン 3 を順守する限り、ユーザーのニーズに変更される場合があります。GPLV3 ライセンスは、gnu.org Web サイト (https://www.gnu.org/licenses/gpl.html) にあります。

11.2 商用ライセンス

wolfSSL 社製品の商用ライセンスは、適用範囲のお客様製品の情報をお伺いした上で、個別に見積書を発行いたします。一番一般的なライセンスは期間無制限、出荷量無制限のライセンスですが、カスタム条件でのライセンス提案も承っております。

詳細は info@wolfssl.jp 宛にお問い合わせください。

11.2.1 サポートパッケージ

wolfSSL 製品のサポートパッケージは、wolfSSL から直接利用できます。3 つの異なるパッケージオプションを使用すると、それらを並べて比較して、特定のニーズに最適なパッケージを選択できます。詳細については、サポートパッケージページをご覧ください。

12 サポートとコンサルティング

12.1 サポートを得るには

一般的な製品サポートのために、wolfSSL(旧 Cyassl) は、wolfSSL 製品ファミリーのオンラインフォーラムを維持しています。フォーラムに投稿するか、弊社までご連絡ください。

wolfssl(yassl) フォーラム: https://www.wolfssl.com/forumshoremail **サポート：** support@wolfssl.com

wolfSSL 製品、ライセンスに関する質問、または一般的なコメントに関する情報については、**info@wolfssl.com** 宛にメールしてください。

12.1.1 バグレポートと障害のサポート

バグレポートを提出したり、問題についてお尋ねになる場合は、次の情報もあわせてお知らせください:

1. wolfSSL バージョン番号
2. オペレーティングシステムバージョン
3. コンパイラバージョン
4. 表示されている正確なエラー番号
5. 障害の再現方法

上記の情報が提供いただけると障害解決に向けて最善を尽くすことができますが、情報のご提供がなければ、問題の原因を特定することは非常に困難となります。wolfSSL はお寄せいただいたフィードバックを大切にし、できるだけ早くご回答することを最優先事項にします。

12.2 コンサルティング

wolfSSL は、機能の追加、移植、競争力のあるアップグレードプログラム、およびデザインコンサルティングを提供します。

詳細は info@wolfssl.jp 宛にお問い合わせください。

12.2.1 機能追加と移植

現時点で、ご要望いただいているのに弊社製品で提供されていない機能を、契約または共同開発ベースで追加することができます。また、当社の製品を新しいホスト言語または新しい操作環境に移植するサービスも提供しています。

詳細は info@wolfssl.jp 宛にお問い合わせください。

12.2.2 デザインコンサルティング

アプリケーションまたはフレームワークを SSL/TLS で保護する必要があるが、安全なシステムの最適な設計がどのように構造化されるべきかについて不確かな場合は、お手伝いできます！

wolfSSL を使用して、SSL/TLS セキュリティをデバイスにビルドするためのデザインコンサルティングを提供しています。

13 wolfSSH の更新

13.1 製品のリリース情報

更新情報を Twitter に定期的に投稿しています。追加のリリース情報については、GitHub でプロジェクトを追跡したり、Facebook でフォローしたり、毎日のブログをフォローしたりできます。

GitHub での wolfSSH https://www.github.com/wolfssl/wolfssh

Twitter での wolfSSL http://twitter.com/wolfSSL

Facebook での wolfSSL http://www.facebook.com/wolfSSL

Reddit での wolfSSL https://www.reddit.com/r/wolfssl/

ブログ https://wolfssl.jp/blog/

14 API リファレンス

このセクションでは、wolfSSH Library の公開 API について説明します。

14.1 エラーコード

14.1.1 WS_ErrorCodes (enum)

以下の戻り値は、wolfssh/wolfssh/error.h で定義されていて、発生する可能性のあるさまざまなタイプのエラーを表します。

- WS_SUCCESS (0): 関数は成功
- WS_FATAL_ERROR (-1): 一般的な失敗
- WS_BAD_ARGUMENT (-2): 引数が範囲外
- WS_MEMORY_E (-3): メモリ確保に失敗
- WS_BUFFER_E (-4): 入/出力バッファのサイズエラー
- WS_PARSE_E (-5): 一般的な解析エラー
- WS_NOT_COMPILED (-6): 機能が組み込まれていない
- WS_OVERFLOW_E (-7): 継続するとオーバーフローする可能性あり
- WS_BAD_USAGE (-8): 使用方法が間違っている
- WS_SOCKET_ERROR_E (-9): ソケットで発生したエラー
- WS_WANT_READ (-10): IO コールバックで読み込みがブロック（再度リードせよ）
- WS_WANT_WRITE (-11): IO コールバックで書き込みがブロック（再度ライトせよ）
- WS_RECV_OVERFLOW_E (-12): 受信バッファがオーバーフローした
- WS_VERSION_E (-13): 相手が異なる SSH バージョンを使っている
- WS_SEND_OOB_READ_E (-14): 帯域外データを読み出そうとした
- WS_INPUT_CASE_E (-15): プロセス入力状態不正あるいはプログラミングエラー
- WS_BAD_FILETYPE_E (-16): ファイルタイプ不正
- WS_UNIMPLEMENTED_E (-17): 機能が未実装
- WS_RSA_E (-18): RSA バッファーエラー
- WS_BAD_FILE_E (-19): ファイル不正
- WS_INVALID_ALGO_ID (-20): 無効なアルゴリズム ID
- WS_DECRYPT_E (-21): 復号エラー
- WS_ENCRYPT_E (-22): 暗号化エラー
- WS_VERIFY_MAC_E (-23): mac 検証エラー
- WS_CREATE_MAC_E (-24): mac 作成エラー
- WS_RESOURCE_E (-25): 新たなチャネル作成にリソース不足
- WS_INVALID_CHANTYPE (-26): 無効なチャネルタイプ
- WS_INVALID_CHANID(-27): ピアが無効なチャネル ID を要求した
- WS_INVALID_USERNAME(-28): 無効なユーザー名
- WS_CRYPTO_FAILED(-29): 暗号アクションが失敗
- WS_INVALID_STATE_E(-30): 無効な状態
- WC_EOF(-31): ファイルの終了
- WS_INVALID_PRIME_CURVE(-32): 無効な ECC プライムカーブ
- WS_ECC_E(-33): ECDSA バッファーエラー
- WS_CHANOPEN_FAILED(-34): ピアがチャネルオープン失敗を返した
- WS_REKEYING(-35): ピアとリキーイング
- WS_CHANNEL_CLOSED(-36): チャネルがクローンした

14.1.2 WS_IOerrors (enum)

以下は、ライブラリがユーザー提供の I/O コールバックから受け取ることを期待しているリターンコードです。それ以外の場合、ライブラリは、I/O アクションから読み取られたバイト数を期待しています。

- WS_CBIO_ERR_GENERAL (-1): 一般的な予期しないエラー
- WS_CBIO_ERR_WANT_READ (-2): ソケットの読み取りブロック（再度リードせよ）
- WS_CBIO_ERR_WANT_WRITE (-2): ソケットの書き込みブロック（再度ライトせよ）
- WS_CBIO_ERR_CONN_RST (-3): コネクションがリセットされた
- WS_CBIO_ERR_ISR (-4): 割り込み発生
- WS_CBIO_ERR_CONN_CLOSE (-5): コネクションがクローンした
- WS_CBIO_ERR_TIMEOUT (-6): ソケットタイムアウト

14.2 初期化 /シャットダウン

14.2.1 wolfSSH_Init()

用法

説明

wolfSSH ライブラリを初期化します。アプリケーションごとに 1 回、ライブラリへの他の呼び出しの前に呼び出される必要があります。

戻り値

WS_SUCCESS

WS_CRYPTO_FAILED

引数

なし

```
#include <wolfssh/ssh.h>
int wolfSSH_Init(void);
```

関連項目

wolfSSH_Cleanup()

14.2.2 wolfSSH_Cleanup()

用法

説明

wolfSSH ライブラリをクリーンアップします。アプリケーションの終了前に呼び出す必要があります。本関数呼び出し後は、ライブラリ API の呼び出しはできません。

戻り値

WS_SUCCESS

WS_CRYPTO_FAILED

引数

なし

```
#include <wolfssh/ssh.h>
int wolfSSH_Cleanup(void);
```

関連項目

wolfSSH_Init()

14.3 デバッグ出力関数

14.3.1 wolfSSH_Debugging_ON()

用法

説明

実行中にデバッグロギングを有効にします。ビルド時にデバッグが無効になっている場合、何もしません。

戻り値

なし

引数

なし

```
#include <wolfssh/ssh.h>
void wolfSSH_Debugging_ON(void);
```

関連項目

wolfSSH_Debugging_OFF()

14.3.2 wolfSSH_Debugging_OFF()

用法

説明

実行時にデバッグロギングを無効にします。ビルド時にデバッグが無効になっている場合、何もしません。

戻り値

なし

引数

なし

```
#include <wolfssh/ssh.h>
void wolfSSH_Debugging_OFF(void);
```

関連項目

wolfSSH_Debugging_ON()

14.4 コンテキスト関数

14.4.1 wolfSSH_CTX_new()

用法

説明

wolfSSH コンテキストオブジェクトを作成します。このオブジェクトは wolfSSH セッションオブジェクトのファクトリとして使用されます。

戻り値

WOLFSSH_CTX – 割り当てられた WOLFSSH_CTX オブジェクトへのポインターあるいは NULL

引数

side – クライアントサイド (実装なし) またはサーバーサイドを示します

heap – メモリ割り当てに使用するヒープへのポインター

```
#include <wolfssh/ssh.h>
WOLFSSH_CTX* wolfSSH_CTX_new(byte side , void* heap );
```

関連項目

wolfSSH_CTX_free()

14.4.2 wolfSSH_CTX_free()

用法

説明

WOLFSSH_CTX オブジェクトを解放します

戻り値

なし

引数

ctx – WOLFSSH_CTX オブジェクト

```
#include <wolfssh/ssh.h>
void wolfSSH_CTX_free(WOLFSSH_CTX* ctx );
```

関連項目

wolfSSH_CTX_new()

14.4.3 wolfSSH_CTX_SetBanner()

用法

説明

バナーメッセージをセットします

戻り値

WS_BAD_ARGUMENT

WS_SUCCESS

引数

ssh - wolfSSH オブジェクト

newBanner - バナーメッセージ文字列

```
#include <wolfssh/ssh.h>
int wolfSSH_CTX_SetBanner(WOLFSSH_CTX* ctx , const char* newBanner );
```

14.4.4 wolfSSH_CTX_UsePrivateKey_buffer()

用法

説明 この関数は、秘密鍵バッファを SSH コンテキストにロードします。ファイルの代わりにバッファーを入力として呼び出されます。バッファは、**insz** の **in** 引数によって提供されます。

引数

format バッファのタイプを指定します：**wolfssh_format_asn1** または **wolfssl_format_pem** (現時点では未実装)。

戻り値

WS_SUCCESS

WS_BAD_ARGUMENT – 少なくとも一つの引数が不正

WS_BAD_FILETYPE_E – フォーマットが不正

WS_UNIMPLEMENTED_E – PEM フォーマットは未対応

WS_MEMORY_E – メモリ確保エラー

WS_RSA_E – RSA 鍵をデコードできない

WS_BAD_FILE_E – バッファを解析できない

引数

ctx – wolfSSH_CTX オブジェクトへのポインター

in – 秘密鍵を含むバッファへのポインター

inSz – 入力バッファのサイズ

format – 秘密鍵のフォーマット

```
#include <wolfssh/ssh.h>
int wolfSSH_CTX_UsePrivateKey_buffer(WOLFSSH_CTX* ctx , const byte* in ,
    word32 inSz , int format);
```

関連項目

wolfSSH_UseCert_buffer()

wolfSSH_UseCaCert_buffer()

14.5 SSH セッション関数

14.5.1 wolfSSH_new()

用法

説明

wolfSSH セッションオブジェクトを確保し、与えられた wolfSSH_CTX オブジェクトを使って初期化します。

戻り値

WOLFSSH* – WOLFSSH オブジェクトへのポインターあるいは NULL

引数

ctx – wolfSSH セッションの初期化に使用される wolfSSH コンテキスト

```
#include <wolfssh/ssh.h>
WOLFSSH* wolfSSH_new(WOLFSSH_CTX* ctx );
```

関連項目

wolfSSH_free()

14.5.2 wolfSSH_free()

用法

説明

wolfSSH オブジェクトを解放します

戻り値

なし

引数

ssh – 解放する WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
void wolfSSH_free(WOLFSSH* ssh );
```

関連項目

wolfSSH_new()

14.5.3 wolfSSH_set_fd()

用法

説明

与えられたファイルディスクリプタを ssh オブジェクトに関連付けます。ファイルディスクリプタはネットワーク I/O に使用され、I/O コールバック関数に渡されます。

戻り値

WS_SUCCESS

WS_BAD_ARGUMENT – 引数の少なくともひとつが不正

引数

ssh – WOLFSSH オブジェクトへのポインター

fd – セッションで使用されるソケットディスクリプター

```
#include <wolfssh/ssh.h>
int wolfSSH_set_fd(WOLFSSH* ssh , int fd );
```

関連項目

wolfSSH_get_fd()

14.5.4 wolfSSH_get_fd()

用法

説明

SSH コネクションの入出力機能で使用されるファイルディスクリプタ (**fd**) を返します。一般的にはソケットファイルディスクリプタを返します。

戻り値

int – ファイルディスクリプタ

WS_BAD_ARGUEMENT

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
int wolfSSH_get_fd(const WOLFSSH* ssh );
```

関連項目

wolfSSH_set_fd()

14.6 ハイウォーターマーク機能

14.6.1 wolfSSH_SetHighwater()

用法

説明

SSH セッションで使用するハイウォーターマークをセットします。

戻り値

WS_SUCCESS

WS_BAD_ARGUMENT

引数

ssh - WOLFSSH オブジェクトへのポインター

highwater - ハイウォーターマークを示すデータ

```
#include <wolfssh/ssh.h>
int wolfSSH_SetHighwater(WOLFSSH* ssh , word32 highwater );
```

14.6.2 wolfSSH_GetHighwater()

用法

説明

ハイウォーターマークを返します。

戻り値

word32 - ハイウォーターマーク

引数

ssh - WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
word32 wolfSSH_GetHighwater(WOLFSSH* ssh );
```

14.6.3 wolfSSH_SetHighwaterCb()

用法

説明

SSH セッションにハイウォーターマークとハイウォーターコールバック関数を設定します。

戻り値

なし

引数

ctx – wolfSSH コンテキスト

highwater - ハイウォーターマーク

cb - ハイウォーターコールバック関数

```
#include <wolfssh/ssh.h>
void wolfSSH_SetHighwaterCb(WOLFSSH_CTX* ctx , word32 highwater ,
WS_CallbackHighwater cb );
```

14.6.4 wolfSSH_SetHighwaterCtx()

用法

説明

ハイウォーターコールバック関数に渡されるコンテキストを設定します。

戻り値

なし

引数

ssh - WOLFSSH オブジェクトへのポインター

ctx - ハイウォーターコールバック関数に渡されるコンテキスト

```
#include <wolfssh/ssh.h>
void wolfSSH_SetHighwaterCtx(WOLFSSH* ssh, void* ctx);
```

14.6.5 wolfSSH_GetHighwaterCtx()

用法

説明

SSH セッションにセットされたハイウォーターマークを返します。

戻り値

void* - ハイウォーターマーク

NULL - WOLFSSH オブジェクトにハイウォーターマークがセットされていない場合

引数

ssh - WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
void wolfSSH_GetHighwaterCtx(WOLFSSH* ssh );
```

14.7 エラーチェック

14.7.1 wolfSSH_get_error()

用法

説明

wolfSSH セッションオブジェクトにセットされたエラーコードを返します。

戻り値

WS_ErrorCodes (enum)

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
int wolfSSH_get_error(const WOLFSSH* ssh );
```

関連項目

wolfSSH_get_error_name()

14.7.2 wolfSSH_get_error_name()

用法

説明

wolfSSH セッションオブジェクトにセットされたエラーの名前を返します。

戻り値

const char* – エラー名文字列

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
const char* wolfSSH_get_error_name(const WOLFSSH* ssh );
```

関連項目

wolfSSH_get_error()

14.7.3 wolfSSH_ErrorToName()

用法

説明

引数で指定されたエラーコードに対応するエラーの名前を返します。

戻り値

const char* – エラー名文字列

引数

err - エラーコード

```
#include <wolfssh/ssh.h>
const char* wolfSSH_ErrorToName(int err );
```

14.8 I/O コールバック関数

14.8.1 wolfSSH_SetIORecv()

用法

説明

入力データを受信する為の受信コールバック関数を登録します。

戻り値

なし

引数

ctx – wolfSSH コンテキスト

cb – wolfSSH コンテキストに関連つけられる、受信コールバック関数

```
#include <wolfssh/ssh.h>
void wolfSSH_SetIORecv(WOLFSSH_CTX* ctx , WS_CallbackIORecv cb );
```

14.8.2 wolfSSH_SetIOSend()

用法

説明

送信データを送信するための送信コールバック関数を登録します。

戻り値

なし

引数

ctx – wolfSSH コンテキスト

cb – wolfSSH コンテキストに関連つけられる、送信コールバック関数。

```
#include <wolfssh/ssh.h>
void wolfSSH_SetIOSend(WOLFSSH_CTX* ctx , WS_CallbackIOSend cb );
```

14.8.3 wolfSSH_SetIOReadCtx()

用法

説明

受信コールバック関数に渡されるコンテキストを設定します

戻り値

なし

引数

ssh – WOLFSSH オブジェクトへのポインター

ctx – コンテキストへのポインター。受信コールバック関数に渡される

```
#include <wolfssh/ssh.h>
void wolfSSH_SetIOReadCtx(WOLFSSH* ssh , void* ctx );
```

14.8.4 wolfSSH_SetIOWriteCtx()

用法

説明

送信コールバック関数に渡されるコンテキストを設定します

戻り値

なし

引数

ssh – WOLFSSH オブジェクトへのポインター

ctx – コンテキストへのポインター。送信コールバック関数に渡される

```
#include <wolfssh/ssh.h>
void wolfSSH_SetIOWriteCtx(WOLFSSH* ssh , void* ctx );
```

14.8.5 wolfSSH_GetIOReadCtx()

用法

説明

WOLFSSH オブジェクトの ioReadCtx メンバーを返します。

戻り値

void* - WOLFSSH オブジェクトの ioReadCtx メンバーへのポインター

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
void* wolfSSH_GetIOReadCtx(WOLFSSH* ssh );
```

14.8.6 wolfSSH_GetIOWriteCtx()

用法

説明

WOLFSSH オブジェクトの ioWriteCtx メンバーを返します。

戻り値

void* – WOLFSSH オブジェクトの ioWriteCtx メンバーへのポインター

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
void* wolfSSH_GetIOWriteCtx(WOLFSSH* ssh);
```

14.9 ユーザー認証

14.9.1 wolfSSH_SetUserAuth()

用法

説明

現在の WOLFSSL_CTX オブジェクトに対してユーザー認証コールバック関数を登録します。

戻り値

なし

引数

ctx – WOLFSSH_CTX オブジェクトへのポインター

cb – ユーザー認証コールバック関数

```
#include <wolfssh/ssh.h>
void wolfSSH_SetUserAuth(WOLFSSH_CTX* ctx, WS_CallbackUserAuth cb)
```

14.9.2 wolfSSH_SetUserAuthCtx()

用法

説明

ユーザー認証コールバック関数に渡されるコンテキストを登録します。

戻り値

なし

引数

ssh – WOLFSSH オブジェクトへのポインター

userAuthCtx – ユーザー認証コールバック関数へ渡すコンテキスト

```
#include <wolfssh/ssh.h>
void wolfSSH_SetUserAuthCtx(WOLFSSH* ssh , void* userAuthCtx)
```

14.9.3 wolfSSH_GetUserAuthCtx()

用法

説明

ユーザー認証コールバック関数に渡されるコンテキストを返します。

戻り値

void* – ユーザー認証コールバック関数へ渡すコンテキスト

NULL – ssh 引数が NULL の場合

引数

ssh – pointer to WOLFSSH object

```
#include <wolfssh/ssh.h>
void* wolfSSH_GetUserAuthCtx(WOLFSSH* ssh )
```

14.10 ユーザー名設定機能

14.10.1 wolfSSH_SetUsername()

用法

説明

SSH コネクションに必要なユーザー名を設定します。

戻り値

WS_BAD_ARGUMENT

WS_SUCCESS

WS_MEMORY_E

引数

ssh - WOLFSSH オブジェクトへのポインター

username - ユーザー名文字列

```
#include <wolfssh/ssh.h>
int wolfSSH_setUsername(WOLFSSH* ssh , const char* username);
```

14.11 接続機能

14.11.1 wolfSSH_accept()

用法

説明

wolfssh_accept はサーバー側で呼び出され、SSH クライアントが SSH ハンドシェイクを開始するのを待ちます。

wolfssl_accept() は、ブロッキング I/O ノンブロッキング I/O の両方で機能します。使用している I/O が非ブロッキングである場合、wolfSSH_accept() は、ハンドシェークが完了できなかった場合は即戻ります。この場合、wolfssh_get_error() を呼び出すと、**WS_WANT_READ** または **WS_WANT_WRITE** のいずれかが返されます。

この場合呼び出し元は、読み取るべきデータを受信して wolfSSH が中断されたところからピックアップできるように、wolfSSH_accept への呼び出しを繰り返す必要があります。非ブロッキングソケットを使用する場合、何も実行する必要はありませんが、select() を使用して必要な条件を確認できます。

使用している I/O がブロッキングの場合、wolfSSH_accept() は、ハンドシェークが終了したか、エラーが発生した場合にのみ戻ります。

戻り値

WS_SUCCESS - 成功

WS_BAD_ARGUMENT - 引数が NULL

WS_FATAL_ERROR – エラーが発生した。wolfSSH_get_error() を呼び出して詳細を取得すべき

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
int wolfSSH_accept(WOLFSSH* ssh);
```

関連項目

wolfSSH_stream_read()

14.11.2 wolfSSH_connect()

用法

説明

この関数はクライアント側で呼び出され SSH ハンドシェークをサーバーに対して開始します。この関数が呼び出される時点では下層の通信チャネルは接続が完了している必要があります。

wolfSSH_connect() 関数はブロッキングとノンブロッキング I/O の両方で動作できます。ノンブロッキング I/O の場合にはハンドシェークが完了できなかった場合は即戻ります。この場合、wolfssh_get_error() を呼び出すと、**WS_WANT_READ** または **WS_WANT_WRITE** のいずれかが返されます。

この場合呼び出し元は、読み取るべきデータを受信して wolfSSH が中断されたところからピックアップできるように、wolfSSH_connect への呼び出しを繰り返す必要があります。非ブロッキングソケットを使用する場合、何も実行する必要はありませんが、select() を使用して必要な条件を確認できます。

使用している I/O がブロッキングの場合、wolfSSH_accept() は、ハンドシェークが終了したか、エラーが発生した場合にのみ戻ります。

戻り値 WS_SUCCESS - 接続に成功

WS_BAD_ARGUMENT - 引数が NULL

WS_FATAL_ERROR - エラーが発生した。wolfSSH_get_error() を呼び出して詳細を取得すべき

引数

ssh - WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
int wolfSSH_connect(WOLFSSH* ssh);
```

14.11.3 wolfSSH_shutdown()

用法

説明

SSH チャネルの接続を終了してクローズします

戻り値

WS_BAD_ARGUMENT - 引数が NULL

WS_SUCCES - 正常にシャットダウンが成功した

引数

ssh - WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
int wolfSSH_shutdown(WOLFSSH* ssh);
```

14.11.4 wolfSSH_stream_read()

用法

説明

wolfSSH_stream_read() は内部のバッファから復号済みデータを **bufSz** で指定されたバイト数まで読みだします。読み込まれたデータはバッファから取り除かれます。

wolfSSH_stream_read() はブロッキングとノンブロッキング I/O の両方で動作できます。ノンブロッキング I/O の場合にはハンドシェークが完了できなかった場合は即戻ります。この場合、wolfssh_get_error() を呼び出すと、**WS_WANT_READ** または **WS_WANT_WRITE** のいずれかが返されます。

この場合呼び出し元は、読み取るべきデータを受信して wolfSSH が中断されたところからピックアップできるように、wolfSSH_stream_read() の呼び出しを繰り返す必要があります。非ブロッキング I/O が使用されている場合、何も実行する必要はありませんが、select() を使用して必要な条件を確認できます。

ブロッキング I/O が使用されている場合は、wolfSSH_stream_read() は、データが IsAbaible またはエラーが発生した場合にのみ戻ります。

戻り値

>0 – 読み取りに成功したバイト数

0 – クリーンコネクションシャットダウンかソケットエラー

WS_BAD_ARGUMENT – 引数の一つが NULL

WS_EOF – ストリームの終端に到達

WS_FATAL_ERROR – エラーが発生。**wolfSSH_get_error()** を呼び出して詳細を取得すべき

WS_REKEYING - リキーイング処理中。wolfSSH_worker() を呼び出して完了させること

引数

ssh – WOLFSSH オブジェクトへのポインター

buf – wolfSSH_stream_read() が読みだしたデータを格納するバッファへのポインター

bufSz – バッファサイズ

```
#include <wolfssh/ssh.h>
int wolfSSH_stream_read(WOLFSSH* ssh, byte* buf, word32 bufSz);
```

関連項目

wolfSSH_accept()

wolfSSH_stream_send()

14.11.5 wolfSSH_stream_send()

用法

説明

wolfSSH_stream_send() はバッファで与えたデータを **bufSz** で指定されたバイト数まで SSH ストリームデータバッファに書き込みます。

wolfSSH_stream_send() はブロッキングとノンブロッキング I/O の両方で動作できます。ノンブロッキング I/O の場合にはハンドシェークが完了できなかった場合は即戻ります。この場合、wolfssh_get_error() を呼び出すと、**WS_WANT_READ** または **WS_WANT_WRITE** のいずれかが返されます。

この場合呼び出し元は、データの書き込みがペンディングされ、wolfSSH が中断されたところから書き込みを再開できるように、wolfSSH_stream_send() の呼び出しを繰り返す必要があります。非ブロッキングソケットを使用する場合、何も実行する必要はありませんが、select() を使用して必要な条件を確認できます。

ブロッキング I/O が使用されている場合は、wolfSSH_stream_send() は、データが送信されたときかエラーが発生した時のみ戻ります。

WS_WANT_READ または WS_WANT_WRITE のいずれもかえされていない場合 (すなわち **WS_REKEYING** が返された場合）は、内部処理が終了するまで wolfSSH_worker() を呼び出し続ける必要があります。

戻り値

>0 – SSH ストリームバッファに書き込んだバイト数

0 – クリーンコネクションシャットダウンかソケットエラー。**wolfSSH_get_error()** を呼び出して詳細を取得すること

WS_FATAL_ERROR – エラーが発生。**wolfSSH_get_error()** を呼び出して詳細を取得すること

WS_BAD_ARGUMENT - 引数の一つが NULL

WS_REKEYING - リキーイング処理中。wolfSSH_worker() を呼び出して完了させること

引数

ssh – WOLFSSH オブジェクトへのポインター

buf – wolfSSH_stream_send() が送信するデータを格納するバッファへのポインター

bufSz – size of the buffer

```
#include <wolfssh/ssh.h>
int wolfSSH_stream_send(WOLFSSH* ssh , byte* buf , word32  bufSz);
```

関連項目

wolfSSH_accept()

wolfSSH_stream_read()

14.11.6 wolfSSH_stream_exit()

用法

説明

SSH ストリームを終了させます。

戻り値

WS_BAD_ARGUMENT - 引数の一つが NULL

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

status – SSH コネクションの状態

```
#include <wolfssh/ssh.h>
int wolfSSH_stream_exit(WOLFSSH* ssh, int status);
```

14.11.7 wolfSSH_TriggerKeyExchange()

用法

説明

鍵交換処理を開始します。ハンドシェークに必要なパケットを用意して送信します。

戻り値

WS_BAD_ARGUEMENT – 引数が NULL

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
int wolfSSH_TriggerKeyExchange(WOLFSSH* ssh );
```

14.12 テスト機能

14.12.1 wolfSSH_GetStats()

用法

説明

ssh セッションに関連した、**txCount** , **rxCount** , **seq** , と **peerSeq** を更新します。

戻り値

なし

引数

ssh – WOLFSSH オブジェクトへのポインター

txCount – 総送信済みデータ数を返却する為の変数のアドレス

rxCount – 総受信済みデータ数を返却する為の変数のアドレス

seq – パケットシーケンス番号を返却する為の変数のアドレス。パケットシーケンス番号は 0 から始まりパケット毎にインクリメントされる

peerSeq – 相手パケットシーケンス番号を返却する為の変数のアドレス。パケットシーケンス番号は 0 から始まりパケット毎にインクリメントされる

```
#include <wolfssh/ssh.h>
void wolfSSH_GetStats(WOLFSSH* ssh , word32* txCount , word32* rxCount ,
word32* seq , word32* peerSeq )
```

14.12.2 wolfSSH_KDF()

用法

説明

API テストが鍵派生の既知の回答テストを行うことができるように使用されます。鍵派生関数は鍵マテリアル **k** と **h** を元に対称鍵を生成します。ここで、**k** はデフィーヘルマンのシェアードシークレットであり、**h** は初期の鍵交換中に生成されたハンドシェークのハッシュ値です。**keyid** および **hashid** によって指定される複数のタイプの鍵が導出される可能性があります。

```
Initial IV client to server: keyId = A
Initial IV server to client: keyId = B
Encryption key client to server: keyId = C
Encryption key server to client: keyId = D
Integrity key client to server: keyId = E
Integrity key server to client : keyId = F
```

戻り値

WS_SUCCESS

WS_CRYPTO_FAILED

引数

hashId – キーイングマテリアルを生成させる為のハッシュのタイプ (WC_HASH_TYPE_SHA あるいは WC_HASH_TYPE_SHA256)

keyId – 生成する鍵を示す文字 A から F

key – 期待されている鍵との比較に使用される生成済みの鍵

keySz – 鍵 **key** の生成に必要なサイズ

k – デフィーヘルマン鍵交換で得たシェアードシークレット

kSz – シェアードシークレット **k** のサイズ

h – 鍵交換中に生成されたハンドシェークのハッシュ値

hSz – ハッシュ **h** のサイズ

sessionId – 最初のハッシュ **h** のユニークな ID

sessionIdSz – **sessionId** のサイズ

```
#include <wolfssh/ssh.h>
int wolfSSH_KDF(byte hashId , byte keyId , byte* key , word32  keySz ,
const byte* k , word32 kSz , const byte* h , word32  hSz ,
const byte* sessionId , word32 sessionIdSz );
```

14.13 セッション機能

14.13.1 wolfSSH_GetSessionType()

用法

説明

wolfSSH_GetSessionType() はセッションの種類を返します。

戻り値

WOLFSSH_SESSION_UNKNOWN

WOLFSSH_SESSION_SHELL

WOLFSSH_SESSION_EXEC

WOLFSSH_SESSION_SUBSYSTEM

引数

ssh - WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
WS_SessionType wolfSSH_GetSessionType(const WOLFSSH* ssh );
```

14.13.2 wolfSSH_GetSessionCommand()

用法

説明

セッションの現在のコマンドを返します

戻り値

const char* - コマンドへのポインター

引数

ssh - WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/ssh.h>
const char* wolfSSH_GetSessionCommand(const WOLFSSH* ssh );
```

14.14 ポートフォワーディング関数

14.14.1 wolfSSH_ChannelFwdNew()

用法

説明

wolfSSH セッションに TCP/IP 転送チャネルを設定します。SSH セッションが接続され、認証された場合、ポート _hostport_ の address_host_ のインターフェイスにローカルリスナーが作成されます。そのリスナーの新しい接続があれば、SSH サーバーへの新しい ChannelRequest をトリガーして、ポート _hostport_ で _host_ への接続を確立します。

戻り値

WOLFSSH_CHAN* – エラーの場合は NULL、成功の場合は新たな新しいチャンネルレコード

引数

ssh - WOLFSSH オブジェクトへのポインター

host – バインドリスナーのホストアドレス

hostPort – バインドリスナーのポート

origin – 接続元の IP アドレス

originPort – 接続元のポート

```
#include <wolfssh/ssh.h>
WOLFSSH_CHANNEL* wolfSSH_ChannelFwdNew(WOLFSSH* ssh ,
const char* host , word32 hostPort ,
const char* origin , word32 originPort );
```

14.14.2 wolfSSH_ChannelFree()

用法

説明

チャネル _channel_ のメモリを解放します。チャネルはセッションのチャネルリストから削除されます。

戻り値

int – エラーコード

引数

channel – 解放される wolfSSH チャネル

```
#include <wolfssh/ssh.h>
int wolfSSH_ChannelFree(WOLFSSH_CHANNEL* channel );
```

14.14.3 wolfSSH_worker()

用法

説明

wolfSSH Worker 機能は接続を見守り、データが受信されると処理します。SSH セッションにはセッションの多くの管理すべきメッセージがあり、これにより自動的にケアがあります。特定のチャネルのデータが受信されると、ワーカーはデータをチャネルに配置します。(function wolfssh_stream_read()dosmuch も同じですが、単一のチャネルの受信データも返します。）wolfssh_worker() は次のアクションを実行します：

1. *outputbuffer* 内に保留中のデータを送信しようとします。
2. セッションのソケットに対して *DoReceive()* を呼び出します。
3. 特定のチャネルのデータが受信された場合、データを返して通知を受け取り、チャネル ID を指定して通知します。

戻り値

int – エラーコードあるいはステータス

WS_CHANNEL_RXD – チャネルに受信済みのデータとチャネル ID がセットされている

引数

ssh - WOLFSSH オブジェクトへのポインター

id – ID を格納する変数へのポインター

```
#include <wolfssh/ssh.h>
int wolfSSH_worker(WOLFSSH* ssh , word32* channelId );
```

14.14.4 wolfSSH_ChannelGetId()

用法

説明

引数で与えられたチャネルに対して ID あるいは相手の ID を返します。

戻り値

int – エラーコード

引数

channel – チャネルへのポインター

id – ID を格納する変数へのポインター

peer – 自チャネル ID か相手チャネル ID

```
#include <wolfssh/ssh.h>
int wolfSSH_ChannelGetId(WOLFSSH_CHANNEL* channel ,  word32* id , byte peer);
```

14.14.5 wolfSSH_ChannelFind()

用法

説明

Given a session *ssh* , find the channel associated with *id*.

戻り値

WOLFSSH_CHANNEL* – チャネルへのポインター,ID がリストになければ NULL

引数

ssh - WOLFSSH オブジェクトへのポインター

id – 検索したいチャネル ID

peer – どちらの側 (自 channel ID か相手 channel ID)

```
#include <wolfssh/ssh.h>
WOLFSSH_CHANNEL* wolfSSH_ChannelFind(WOLFSSH* ssh ,
word32 id , byte peer );
```

14.14.6 wolfSSH_ChannelRead()

用法

説明

チャネルオブジェクトからデータをコピーします

戻り値

int – 読みだしたバイト数

>0 – 成功時には読みだしたバイト数を返します

0 – クリーンコネクションシャットダウンかソケットエラーが発生している。エラー詳細を取得するために wolfSSH_get_error() を呼び出すこと。

WS_FATAL_ERROR – そのほかのエラーが発生。エラー詳細を取得するために wolfSSH_get_error() を呼び出すこと。

引数

channel – wolfSSH channel へのポインター

buf – wolfSSH_ChannelRead が読みだしたデータを格納するバッファアドレス

bufSz – バッファのサイズ

```
#include <wolfssh/ssh.h>
int wolfSSH_ChannelRead(WOLFSSH_CHANNEL* channel, byte* buf, word32 bufSz );
```

14.14.7 wolfSSH_ChannelSend()

用法

説明

指定したチャネル経由でデータを相手に送信します。データはチャネルデータメッセージにパッキングされて送られます。さらに送信すべきデータがある場合には、*wolfSSH_worker()* を呼び出すと相手へのデータ送信を継続します。

戻り値

int – 送信したバイト数

>0 – 成功時には送信したバイト数を返す

0 – クリーンコネクションシャットダウンかソケットエラーが発生している。エラー詳細を取得するために wolfSSH_get_error() を呼び出すこと。

WS_FATAL_ERROR – そのほかのエラーが発生。エラー詳細を取得するために wolfSSH_get_error() を呼び出すこと。

引数

channel – wolfSSH channel へのポインター

buf – wolfSSH_ChannelSend() が送信のために読みだすバッファへのポインター

bufSz – バッファのサイズ

```
#include <wolfssh/ssh.h>
int* wolfSSH_ChannelSend(WOLFSSH_CHANNEL* channel, const byte* buf, word32
    bufSz);
```

14.14.8 wolfSSH_ChannelExit()

用法

説明

チャネルを終了し、相手へのメッセージ送信を停止し、チャネルがクローズしたとマークします。この関数はチャネルと残ったデータを解放しませんし、チャネルはリストに残ります。クローズ後は未送信データはそのままですが、受信は可能です。（現時点では EOF と close を送りチャネルを削除します）

戻り値

int – エラーコード

引数

channel – wolfSSH channel へのポインター

```
#include <wolfssh/ssh.h>
int wolfSSH_ChannelExit(WOLFSSH_CHANNEL* channel );
```

14.14.9 wolfSSH_ChannelNext()

用法

説明

ssh の *channel* の次のチャネルを返します。*channel* が NULL の場合には、チャネルリスト内の最初のチャネルを返します。

戻り値

WOLFSSH_CHANNEL – 最初のチャネルあるいは次のチャネルへのポインターあるいは NULL

引数

ssh - WOLFSSH オブジェクトへのポインター

channel – wolfSSH channel へのポインター

```
#include <wolfssh/ssh.h>
WOLFSSH_CHANNEL* wolfSSH_ChannelFwdNew(WOLFSSH* ssh , WOLFSSH_CHANNEL* channel
    );
```

15 wolfSSL SFTP API リファレンス

15.1 接続機能

15.1.1 wolfSSH_SFTP_accept()

用法

説明

クライアントからの接続要求を処理します

戻り値

WS_SFTP_COMPLETE - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_accept(WOLFSSH* ssh );
```

使用例

```
WOLFSSH* ssh;

//create new WOLFSSH structure
...

if (wolfSSH_SFTP_accept(ssh) != WS_SUCCESS) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_free()

wolfSSH_new()

wolfSSH_SFTP_connect()

15.1.2 wolfSSH_SFTP_connect()

用法

説明

SFTP サーバーへの接続を開始します。

戻り値

WS_SFTP_COMPLETE - 成功

引数

ssh - – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_connect(WOLFSSH* ssh );
```

使用例

```
WOLFSSH* ssh;

//after creating a new WOLFSSH structure

wolfSSH_SFTP_connect(ssh);
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_new()

wolfSSH_free()

15.1.3 wolfSSH_SFTP_negotiate()

用法

説明

本関数はクライアントからの接続要求かサーバーへの接続要求のいずれかを処理します。いずれを処理するかは wolfSSH オブジェクトにセットされているアクションに依存します。

戻り値

WS_SUCCESS - 成功

引数

ssh - – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_negotiate(WOLFSSH* ssh)
```

使用例

```
WOLFSSH* ssh;

//create new WOLFSSH structure with side of connection
set
....

if (wolfSSH_SFTP_negotiate(ssh) != WS_SUCCESS) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_free()

wolfSSH_new()

wolfSSH_SFTP_connect()

wolfSSH_SFTP_accept()

15.2 プロトコル関係

15.2.1 wolfSSH_SFTP_RealPath()

用法

説明

REALPATH パケットを相手に送信し、相手から取得したファイル名を返します。

戻り値

成功時には WS_SFTPNAME 構造体へのポインターを返します。エラー発生時には NULL を返します。

引数

ssh – WOLFSSH オブジェクトへのポインター

dir - 実際のパスを取得するためのディレクトリ/ファイル名

```
#include <wolfssh/wolfsftp.h>
WS_SFTPNAME* wolfSSH_SFTP_RealPath(WOLFSSH* ssh , char* dir);
```

使用例

```
WOLFSSH* ssh ;

//set up ssh and do sftp connections
...

if (wolfSSH_SFTP_read( ssh ) != WS_SUCCESS) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

15.2.2 wolfSSH_SFTP_Close()

用法

説明

相手にクローズパケットを送信します。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

handle - 閉じようとするハンドル

handleSz - ハンドルバッファーのサイズ

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_Close(WOLFSSH* ssh , byte* handle , word32 handleSz );
```

使用例

```
WOLFSSH* ssh;
byte handle[HANDLE_SIZE];
word32 handleSz = HANDLE_SIZE;

//set up ssh and do sftp connections
```

```
...

if (wolfSSH_SFTP_Close(ssh, handle, handleSz) != WS_SUCCESS) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

15.2.3 wolfSSH_SFTP_Open()

用法

説明

Open パケットを相手に送信します。結果を受け取るバッファサイズのを handleSz で指定し、相手から受け取ったハンドルをバッファに格納します。

open の理由として取り得る値は:

WOLFSSH_FXF_READ

WOLFSSH_FXF_WRITE

WOLFSSH_FXF_APPEND

WOLFSSH_FXF_CREAT

WOLFSSH_FXF_TRUNC

WOLFSSH_FXF_EXCL

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

dir - 開くファイルの名前

reason - ファイルを開く理由

atr - ファイルの初期属性

handle - 結果として得られるハンドル

handleSz - ハンドル用バッファのサイズ

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_Open(WOLFSSH* ssh , char* dir , word32  reason,
    WS_SFTP_FILEATRB* atr , byte* handle , word32* handleSz);
```

使用例

```
WOLFSSH* ssh ;
char name[NAME_SIZE];
byte handle[HANDLE_SIZE];
word32 handleSz = HANDLE_SIZE;
WS_SFTP_FILEATRB atr;
```

```
//set up ssh and do sftp connections
...

if (wolfSSH_SFTP_Open( ssh , name , WOLFSSH_FXF_WRITE | WOLFSSH_FXF_APPEND |
    WOLFSSH_FXF_CREAT , &atr , handle , &handleSz ) != WS_SUCCESS) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

15.2.4 wolfSSH_SFTP_SendReadPacket()

用法

説明

read パケットを相手に送信します。ハンドル用のバッファは直前の wolfSSH_SFTP_Open で得られたハンドルを格納していなければなりません。読みだすことができたデータは out バッファに格納されます。

戻り値

成功時には読みだしたデータ数を返します。エラー発生時には、負の値を返します。

引数

ssh – WOLFSSH オブジェクトへのポインター

handle - 読みだそうとするハンドル

handleSz - ハンドルバッファのサイズ

ofst - 読み出しを開始するオフセット

out - 読み出した結果を格納するバッファ

outSz - バッファサイズ

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_SendReadPacket(WOLFSSH* ssh , byte*  handle , word32 handleSz
     , word64 ofst , byte* out , word32 outSz );
```

使用例

```
WOLFSSH* ssh;
byte handle[HANDLE_SIZE];
word32 handleSz = HANDLE_SIZE;
byte out[OUT_SIZE];
word32 outSz = OUT_SIZE;
word32 ofst = 0;
int ret;

//set up ssh and do sftp connections
...
//get handle with wolfSSH_SFTP_Open()

if ((ret = wolfSSH_SFTP_SendReadPacket(ssh, handle, handleSz, ofst, out, outSz
    )) < 0) {
```

```
//handle error case
}
//ret holds the number of bytes placed into out buffer
```

関連項目

wolfSSH_SFTP_SendWritePacket()

wolfSSH_SFTP_Open()

15.2.5 wolfSSH_SFTP_SendWritePacket()

用法

説明

write パケットを相手に送信します。ハンドル用のバッファは直前の wolfSSH_SFTP_Open で得られたハンドルを格納していなければなりません。

戻り値

成功時には書き込んだサイズを返します。エラー発生時には負の値を返します。

引数

ssh – WOLFSSH オブジェクトへのポインター

handle - 書き込もうとするハンドル

handleSz - ハンドルバッファのサイズ

ofst - 書き込みを開始するオフセット

out - 書き込むデータを保持するバッファ

outSz - バッファサイズ

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_SendWritePacket(WOLFSSH* ssh, byte*  handle, word32  handleSz
    , word64 ofst, byte* out, word32 outSz);
```

使用例

```
WOLFSSH* ssh;
byte handle[HANDLE_SIZE];
word32 handleSz = HANDLE_SIZE;
byte out[OUT_SIZE];
word32 outSz = OUT_SIZE;
word32 ofst = 0;
int ret;

//set up ssh and do sftp connections
...
//get handle with wolfSSH_SFTP_Open()

if ((ret = wolfSSH_SFTP_SendWritePacket(ssh, handle, handleSz, ofst, out,
    outSz)) < 0) {
//handle error case
}
//ret holds the number of bytes written
```

関連項目

wolfSSH_SFTP_SendReadPacket()

wolfSSH_SFTP_Open()

15.2.6 wolfSSH_SFTP_STAT()

用法

説明

STAT パケットを相手に送信します。ファイルあるいはディレクトリの属性を取得します。ファイルが存在しないかあるいは属性が存在しない場合は相手はエラーを返します。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

dir - NULL ターミネートされたファイルあるいはディレクトリ名

atr - 属性値がこの構造体に返却されます

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_STAT(WOLFSSH* ssh , char* dir, WS_SFTP_FILEATRB* atr);
```

使用例

```
WOLFSSH* ssh;
byte name[NAME_SIZE];
int ret;
WS_SFTP_FILEATRB atr;

//set up ssh and do sftp connections
...

if ((ret = wolfSSH_SFTP_STAT(ssh, name, &atr)) < 0) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_LSTAT()

wolfSSH_SFTP_connect()

15.2.7 wolfSSH_SFTP_LSTAT()

用法

説明

LSTAT パケットを相手に送信します。ファイルあるいはディレクトリの属性値を取得します。STAT パケットがシンボリックリンクをたどりませんが LSTAT パケットはシンボリックリンクをたどって処理します。ファイルが存在しないかあるいは属性が存在しない場合は相手はエラーを返します。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

dir - NULL ターミネートされたファイルあるいはディレクトリ名

atr - 属性値がこの構造体に返却されます

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_LSTAT(WOLFSSH* ssh, char* dir, WS_SFTP_FILEATRB* atr);
```

使用例

```
WOLFSSH* ssh;
byte name[NAME_SIZE];
int ret;
WS_SFTP_FILEATRB atr;

//set up ssh and do sftp connections
...

if ((ret = wolfSSH_SFTP_LSTAT(ssh, name, &atr)) < 0) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_STAT()

wolfSSH_SFTP_connect()

15.2.8 wolfSSH_SFTPNAME_free()

用法

説明

単一の WS_SFTPNAME ノードを解放します。指定したノードがノードリストの途中のものであった場合には、リストは壊れます。

戻り値

なし

引数

name - 解放されるノード

```
#include <wolfssh/wolfsftp.h>
void wolfSSH_SFTPNAME_free(WS_SFTPNAME* name );
```

使用例

```
WOLFSSH* ssh;
WS_SFTPNAME* name;

//set up ssh and do sftp connections
...
name = wolfSSH_SFTP_RealPath(ssh, path);
if (name != NULL) {
wolfSSH_SFTPNAME_free(name);
}
```

関連項目

wolfSSH_SFTPNAME_list_free()

15.2.9 wolfSSH_SFTPNAME_list_free()

用法

説明

リスト中の全 WS_SFTPNAME ノードを解放します。

戻り値

なし

引数

name - 解放するリストの先頭

```
#include <wolfssh/wolfsftp.h>
void wolfSSH_SFTPNAME_list_free(WS_SFTPNMAE* name );
```

使用例

```
WOLFSSH* ssh;
WS_SFTPNAME* name;

//set up ssh and do sftp connections
...

name = wolfSSH_SFTP_LS(ssh, path);
if (name != NULL) {
wolfSSH_SFTPNAME_list_free(name);
}
```

関連項目

wolfSSH_SFTPNAME_free()

15.3 Reget/Reput 機能

15.3.1 wolfSSH_SFTP_SaveOfst()

用法

説明 get あるいは put コマンドが中断された場合のオフセットを保存します。オフセットは wolfSSH_SFTP_GetOfst で復元できます。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

from - NULL 終端されたソースパスを示す文字列

to - NULL 終端されたデスティネーションパスを示す文字列

ofst - 記憶されるべきファイルのオフセット

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_SaveOfst(WOLFSSH* ssh , char* from , char*
to ,
word64 ofst );
```

使用例

```
WOLFSSH* ssh;
char from[NAME_SZ];
char to[NAME_SZ];
word64 ofst;

//set up ssh and do sftp connections
...

if (wolfSSH_SFTP_SaveOfst(ssh, from, to, ofst) !=  WS_SUCCESS) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_GetOfst()

wolfSSH_SFTP_Interrupt()

15.3.2 wolfSSH_SFTP_GetOfst()

用法

説明

get あるいは put コマンドが中断された場合のオフセットを取得します。

戻り値

成功時にはオフセット値を返します。オフセットが保存されていない場合には 0 が返されます。

引数

ssh – WOLFSSH オブジェクトへのポインター

from - NULL 終端されたソースパスを示す文字列

to - NULL 終端されたデスティネーションパスを示す文字列

```
#include <wolfssh/wolfsftp.h>
word64 wolfSSH_SFTP_GetOfst(WOLFSSH* ssh, char* from, char* to);
```

使用例

```
WOLFSSH* ssh;
char from[NAME_SZ];
char to[NAME_SZ];
word64 ofst;

//set up ssh and do sftp connections
...

ofst = wolfSSH_SFTP_GetOfst(ssh, from, to);
//start reading/writing from ofst
```

関連項目

wolfSSH_SFTP_SaveOfst()

wolfSSH_SFTP_Interrup()

15.3.3 wolfSSH_SFTP_ClearOfst()

用法

説明

保存されている全オフセット値をクリアします。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_ClearOfst(WOLFSSH* ssh);
```

使用例

関連項目

wolfSSH_SFTP_SaveOfst()

wolfSSH_SFTP_GetOfst()

15.3.4 wolfSSH_SFTP_Interrupt()

用法

説明

中断フラグをセットし、get/put コマンドを停止します。

戻り値

なし

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/wolfsftp.h>
void wolfSSH_SFTP_Interrupt(WOLFSSH* ssh);
```

使用例

```
WOLFSSH* ssh;

//set up ssh and do sftp connections
...

if (wolfSSH_SFTP_ClearOfst(ssh) != WS_SUCCESS) {
//handle error
}
```

```
WOLFSSH* ssh;
char from[NAME_SZ];
char to[NAME_SZ];
word64 ofst;

//set up ssh and do sftp connections
...

wolfSSH_SFTP_Interrupt(ssh);
wolfSSH_SFTP_SaveOfst(ssh, from, to, ofst);
```

関連項目

wolfSSH_SFTP_SaveOfst()

wolfSSH_SFTP_GetOfst()

15.4 コマンド機能

15.4.1 wolfSSH_SFTP_Remove()

用法

説明

“remove” パケットをチャネルを通じて送信します。削除するファイル名 “f” は相手に渡されます。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター **f** - 削除したいファイル名

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_Remove(WOLFSSH* ssh , char* f );
```

使用例

```
WOLFSSH* ssh;
int ret;
char* name[NAME_SZ];

//set up ssh and do sftp connections
...

ret = wolfSSH_SFTP_Remove(ssh, name);
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

15.4.2 wolfSSH_SFTP_MKDIR()

用法

説明

チャネルを通して “mkdir” パケットを送信します。相手に作成するディレクトリ名が “dir” として渡されます。現時点では、属性は使用されず、既定の属性が使用されます。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

dir - NULL 終端された作成するディレクトリ名を示す文字列

atr - ディレクトリ作成に使う属性値

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_MKDIR(WOLFSSH* ssh, char* dir,  WS_SFTP_FILEATRB* atr);
```

使用例

```
WOLFSSH* ssh;
int ret;
char* dir[DIR_SZ];

//set up ssh and do sftp connections
...

ret = wolfSSH_SFTP_MKDIR(ssh, dir, DIR_SZ);
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

15.4.3 wolfSSH_SFTP_RMDIR()

用法

説明

“rmdir” パケットをチャネルを通じて送信します。削除するディレクトリ名は “dir” として相手に送られます。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

dir - NULL 終端された削除するディレクトリ名を示す文字列

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_RMDIR(WOLFSSH* ssh , char* dir );
```

使用例

```
WOLFSSH* ssh;
int ret;
char* dir[DIR_SZ];
```

```
//set up ssh and do sftp connections
...

ret = wolfSSH_SFTP_RMDIR(ssh, dir);
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

15.4.4 wolfSSH_SFTP_Rename()

用法

説明

“rename” パケットをチャネルを通じて送信します。相手側のファイル名を “old” から “nw” に変更しようとします。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

old - 旧ファイル名

nw - 新ファイル名

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_Rename(WOLFSSH* ssh , const char* old ,  const char* nw);
```

使用例

```
WOLFSSH* ssh;
int ret;
char* old[NAME_SZ];
char* nw[NAME_SZ]; //new file name

//set up ssh and do sftp connections
...

ret = wolfSSH_SFTP_Rename(ssh, old, nw);
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

15.4.5 wolfSSH_SFTP_LS()

用法

説明

LS 操作（全ファイルとディレクトリのリストを取得する）を現在のワーキングディレクトリで実行します。この関数は REALPATH, OPENDIR, READDIR と CLOSE 操作を実行する高水準関数です。

戻り値

成功時には WS_SFTPNAME 構造体のリストを返します。失敗時には NULL を返します。

引数

ssh – WOLFSSH オブジェクトへのポインター

dir - リストを作成するディレクトリ名

```
#include <wolfssh/wolfsftp.h>
WS_SFTPNAME* wolfSSH_SFTP_LS(WOLFSSH* ssh , char* dir );
```

使用例

```
WOLFSSH* ssh;
int ret;
char* dir[DIR_SZ];
WS_SFTPNAME* name;
WS_SFTPNAME* tmp;

//set up ssh and do sftp connections
...

name = wolfSSH_SFTP_LS(ssh, dir);
tmp = name;
while (tmp != NULL) {
printf("%s\n", tmp->fName);
tmp = tmp->next;
}
wolfSSH_SFTPNAME_list_free(name);
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

wolfSSH_SFTPNAME_list_free()

15.4.6 wolfSSH_SFTP_Get()

用法

説明

相手からファイルを取得する get 操作を実行し、ローカルディレクトリに配置します。この関数は高水準関数であり、LSTAT, OPEN, READ, と CLOSE を実行します。関数の実行を中断したい場合には、wolfSSH_SFTP_Interrupt を呼び出すことができます。

戻り値

WS_SUCCESS - 成功 その他の値はすべてエラーとみなすべきです。

引数

ssh – WOLFSSH オブジェクトへのポインター

from - 取得するファイルの名前

to - 配置する際のファイルの名前

resume - 操作を再開するか（1 は再開する、0 はしない）

statusCb - ステータスを取得するコールバック関数

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_Get(WOLFSSH* ssh , char* from , char* to ,  byte resume ,
   WS_STATUS_CB* statusCb );
```

使用例

```
static void myStatusCb(WOLFSSH* sshIn, long bytes, char* name)
{
char buf[80];
WSNPRINTF(buf, sizeof(buf), "Processed %8ld\t bytes
\r", bytes);
WFPUTS(buf, fout);
(void)name;
(void)sshIn;
}
...
WOLFSSH* ssh;
char* from[NAME_SZ];
char* to[NAME_SZ];

//set up ssh and do sftp connections
...

if (wolfSSH_SFTP_Get( ssh , from , to , 0 , & myStatusCb ) != WS_SUCCESS) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

15.4.7 wolfSSH_SFTP_Put()

用法

説明

ローカルのファイルを相手のディレクトリに配置する put 操作を実行します。この関数は高水準関数であり、OPEN, WRITE, と CLOSE 操作を実行します。操作を中断する場合には wolfSSH_SFTP_Interrupt を呼び出してください。

戻り値

WS_SUCCESS - 成功

その他の値はすべてエラーとみなすべきです。

引数

ssh – WOLFSSH オブジェクトへのポインター

from - 配置したい対象ファイルの名前

to - 配置先でのファイルの名前

resume - 操作を再開するかのフラグ (1 は再開、0 は再開しない）

statusCb - ステータスを取得するコールバック関数

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_Put(WOLFSSH* ssh, char* from, char* to, byte resume,
    WS_STATUS_CB* statusCb);
```

使用例

```
static void myStatusCb(WOLFSSH* sshIn, long bytes, char* name)
{
char buf[80];
WSNPRINTF(buf, sizeof(buf), "Processed %8ld\t bytes
\r", bytes);
WFPUTS(buf, fout);
(void)name;
(void)sshIn;
}
...

WOLFSSH* ssh;
char* from[NAME_SZ];
char* to[NAME_SZ];

//set up ssh and do sftp connections
...

if (wolfSSH_SFTP_Put(ssh, from, to, 0, &myStatusCb) !=
WS_SUCCESS) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()

15.5 SFTP サーバー機能

15.5.1 wolfSSH_SFTP_read()

用法

説明

メインの SFTP サーバー機能を提供する関数です。到着するパケットを処理し、I/O バッファからデータを読み出し SFTP パケットのタイプに応じて内部の関数を呼び出します。

戻り値

WS_SUCCESS - 成功

引数

ssh – WOLFSSH オブジェクトへのポインター

```
#include <wolfssh/wolfsftp.h>
int wolfSSH_SFTP_read(WOLFSSH* ssh );
```

使用例

```
WOLFSSH* ssh;

//set up ssh and do sftp connections
...
if (wolfSSH_SFTP_read(ssh) != WS_SUCCESS) {
//handle error case
}
```

関連項目

wolfSSH_SFTP_accept()

wolfSSH_SFTP_connect()