wolfSSH ユーザ・マニュアル

（この資料は wolfSSH User Manual,
Version 1.2
第１章から第６章までの日本語翻訳版です）

目次

© 2004 – 2016 wolfSSL, Inc.

1. イントロダクション

このマニュアルはwolfSSH組み込みライブラリのテクニカルガイドとして書かれています。 wolfSSHのビルド、利用方法を説明し、ビルドオプション、機能、サポート、その他の概要を解説します。

wolfSSHはCで書かれたSSH（Secure Shell）サーバとクライアントの実装で、暗号化基本部分にwolfSSLから入手可能なwolfCryptライブラリを使用しています。 さらに、wolfSSHは多種のプラットフォームで使用できるようにするためにゼロから構築されています。 この実装はSSH v2仕様に基づいています。

1.1 プロトコル概要

SSHは、二者間でデータの多重化ストリームを提供するためのプロトコルの階層セットです。 通常、サーバー上のシェルへの接続を保護するために使用されます。 また、2台のマシン間でファイルを安全にコピーしたり、X11ディスプレイプロトコルをトンネリングするためにもよく使用されます。

1.2 wolfSSH の特長

wolfSSHライブラリはANSI Cで書かれた軽量のSSHv2サーバライブラリで、主にその小さいサイズ、スピード、そして機能セットのためにRTOS、そしてリソースに制約のある組み込み環境をターゲットにしています。ロイヤリティフリーの価格と、優れたクロスプラットフォームサポートを提供していること、また標準的なオペレーティング環境でも同じように使用できる点からも利用されています。 wolfSSHは業界標準のSSH v2をサポートし、Poly1305、ChaCha20、NTRU、SHA-3などのプログレッシブ暗号を提供します。

wolfSSHはwolfCryptライブラリを使用して動作します。 wolfCrypt暗号化ライブラリのバージョンはFIPS 140-2検証済みのものも提供しています（証明書#2425）。詳細については、wolfCrypt FIPS FAQを参照するか、fips @ wolfssl.comにお問い合わせください。

特長のまとめ

SSH v2.0（サーバー、クライアント）
最小フットプリントサイズ33kB
設定可能な受信バッファを含まない、1.4～2kBのランタイムメモリ使用量
複数のハッシュ関数：SHA-1、SHA-2（SHA-256、SHA-384、SHA-512）、BLAKE2b、Poly1305
ブロック、ストリーム、および認証された暗号：AES（CBC、CTR、GCM、CCM）、Camellia、ChaCha20
公開鍵オプション：RSA、DH、EDH、NTRU
ECCサポート（曲線付きECDHおよびECDSA：NISTP256、NISTP384、NISTP521）
Curve25519とEd25519
クライアント認証サポート（RSAキー、パスワード）
簡潔なAPI
PEMおよびDER証明書のサポート
ハードウェア暗号化サポート：Intel AES-NIサポート、Intel AVX1 / 2、RDRAND、RDSEED、Cavium NITROXサポート、STM32F2 / F4ハードウェア暗号化サポート、フリースケールCAU / mmCAU / SEC、マイクロチップPIC32MZ

© 2004 – 2016 wolfSSL, Inc.

2. wolfSSH をビルドする

wolfSSHは移植性を念頭に置いて設計されており、一般にほとんどのシステムで簡単に構築できるようになっています。 構築が困難な場合は、遠慮なく当社のサポートフォーラムhttps://www.wolfssl.com/forumsからサポートを受けてください。また、support@wolfssl.com（日本語可）に直接お問い合わせください。

このセクションでは、＊nix風およびWindows環境でwolfSSHを構築する方法を説明し、非標準環境で構築するための手引きを提供します。 セクション3に入門ガイドと例があります。

autoconf / automakeシステムを使用して構築する場合、wolfSSHは単一のMakefileを使用してライブラリのすべての部分と例を構築します。これは、Makefileを再帰的に使用するよりも簡単です。

2.1 ソースコードを入手する

安定版は弊社 wolfssl.jp のダウンロードページからzipファイルで入手することができます。フォームに必要な情報を記入して、ダウンロードしてください。

また、最新バージョンを GitHubのWeb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、または端末で以下のコマンドを使用してください。

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

2.2 wolfSSH の依存コンポーネント

wolfSSH は wolfCrypt に依存しているので、wolfSSL の設定が必要です。 wolfSSL は https://github.com/wolfSSL/wolfssl からダウンロードできます。

wolfSSH に必要な最も簡単な wolfSSL の設定はデフォルトのビルドで、これは以下のコマンドで wolfSSL のルートディレクトリからビルドすることができます。

$ ./autogen.sh（GitHub からクローンを作成した場合のみ）
$ ./configure --enable-ssh
$ make check
$ sudo make install

wolfSSH で鍵生成機能を使うには、wolfSSL を keygen：--enable-keygen で設定する必要があります。

wolfSSL のプロトコルレイヤーのコードが必要ない場合は、暗号のみのオプション--enable-cryptonly を使用して不要部分を取り除いた wolfSSL を構成できます。

2.3 *nix でのビルド

Linux, *BSD, OS X, Solaris, またはその他の *nixlikeシステムでのビルドには, Autoconf ツールを使用します。. wolfSSH をビルドするには、以下のコマンドを発行するだけです:

.$ ./autogen.sh （GitHubからクローンを作成した場合のみ）
$ ./configure
$ make
$ make install

© 2004 – 2016 wolfSSL, Inc.

./configureに適当なビルドオプションを適用することができます。利用可能なオプションの一覧は次のようにして表示させます：

./configure　--help

wolfSSHをビルドするためには次のコマンドを実行します:

$ make

wolfSSHが正しく構築されたことを確認するために、すべてのテストが正常に動作したかどうかを確認してください。

$ make check

WolfッSH をインストールするには次のコマンドを実行します:

make install

インストールには特権ユーザの権限が必要かもしれません。その場合は次のようにします：

sudo make install

ビルド結果をテストするためには、wolfMQTTルートディレクトリから、次のようにmqttclientを実行してみます:

./examples/mqttclient/mqttclient

wolfSSHライブラリだけをビルドし、その他のサンプルなどをビルドしたくない場合は、wolfSSHルート・ディレクトリから次のコマンドを実行します：

make src/libwolfssh.la

2.4 Windows でのビルド

Visual Studio 2015 の場合: wolfmqtt.sln ソリューションが以下ディレクトリにあります。
https://github.com/wolfSSL/wolfssh/blob/master/ide/winvs/wolfssh.sln

ソリューションファイルwolfssh.slnは、wolfSSHとその例およびテストプログラムの構築を容易にします。 このソリューションは、静的および動的32ビットまたは64ビットライブラリのデバッグビルドとリリースビルドの両方を提供します。 設定するには、ファイルuser_settings.hをwolfSSLビルドで使用する必要があります。

このプロジェクトは以下のようにwolfSSHとwolfSSLソースディレクトリが並んでインストールされていて、それらのディレクトリ名にバージョン番号を持っていないと仮定します：

```
Projects¥
    wolfssh¥
    wolfssl¥
```

Windows上で構築するためのユーザマクロ

Solutionは、wolfSSLライブラリとヘッダの場所を示すためにユーザマクロを使用しています。 wolfssl64ソリューションでは、すべてのパスがデフォルトのビルド先に設定されています。 ユーザマクロwolfCryptDirはライブラリを見つけるためのベースパスとして使われます。 初期設定は.. ¥ .. ¥ .. ¥ .. ¥ wolfsslです。 そして、たとえば、APIテストプロジェクトの追加のインクルードディレクトリの値は$（wolfCryptDir）に設定されます。

wolfCryptDirパスはプロジェクトファイルからの相対パスである必要があります。これらはすべて1ディレクトリ下にあります。

wolfssh/wolfssh.vcxproj

unit-test/unit-test.vcxproj

その他のユーザマクロは、異なるビルド用のwolfSSLライブラリが存在する可能性のあるディレクトリのためのものです。 そのため、ユーザーマクロwolfCryptDllRelease64は最初は次のように設定されています。

$(wolfCryptDir)¥x64¥DLL Release

この値は、echoserverの64ビットDLLリリースビルドのデバッグ環境で使用されます。
PATH=$(wolfCryptDllRelease64);%PATH%

デバッガから echoserver を実行すると、そのディレクトリにwolfSSL DLLが見つかります。

2.5 非標準環境でのビルド

正式サポートではありませんが、wolfSSL を非標準の環境、特に組込み向けのクロスコンパイルシステムでビルドしようと思われる方々をできる限りお手伝いします。以下は、この環境での作業開始のための注意点です。

ソースファイルとヘッダファイルは、wolfSSHダウンロードパッケージに含まれているのと同じディレクトリ構造に残る必要があります。

ビルドシステムによっては、wolfSSHヘッダファイルがどこにあるのかを明示的に指定する必要があるかもしれません。 それらは<wolfssh_root>/wolfsshディレクトリにあります。 通常、ヘッダの問題を解決するために<wolfssh_root>ディレクトリをインクルードパスに追加します。

configureプロセスがビッグエンディアンを検出しない限り、wolfSSHはデフォルトでリトルエンディアンシステムになります。 非標準環境で構築しているユーザーはconfigureプロセスを使用していないため、ビッグエンディアンシステムを使用している場合はBIG_ENDIAN_ORDERを定義する必要があります。

ライブラリを構築し、何か問題が発生した場合はご遠慮なくお知らせください。 サポートが必要な場合は、support @ wolfssl.com（日本語可）までご連絡ください。

2.6 クロスコンパイル

組込み用プラットフォームでユーザ環境向けに wolfSSH をクロスコンパイルするケースが多々あります。./configure システムを使用するともっとも簡単にクロスコンパイルすることができます。これによって、wolfSSH ビルドに使用することができる Makefile を生成します。
クロスコンパイルの場合、./configue に対してホストを指定する必要があります。例えば以下のように：

./configure --host=arm-linux

また、使用したいコンパイラー、リンカーなどの指定も必要となるかも知れません：

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

クロスコンパイル向けに wolfSSH を正しくコンフィグレーションした後、標準の autoconf 実行でライブラリーのビルドとインストールを行ってください。

make
sudo make install

もし、wolfSSHクロスコンパイルに関してさらにヒントやフィードバックがありましたらinfo@wolfssl.jpまでお知らせください。

2.7 指定ディレクトリにインストールする

wolfSSLのインストールディレクトリを特定の場所に指定したい場合は:

wolfSSLのビルドで:

1. ./configure prefix=~/wolfssl
2. make
3. make install

これによってライブラリは~/wolfssl/lib に、インクルード・パスは ~/wolfssl/include に設定されます。wolfSSH に対して特定のインストールディレクトリ、wolfSSLのインストール場所を指定するには:

wolfSSHにて:

1. ./configure prefix=~/wolfssh
   libdir=~/wolfssl/lib includedir=~/wolfssl/include
2. make
3. make install

上記指定のパスが実際の格納位置とマッチしていることを確認してください。

© 2004 – 2016 wolfSSL, Inc.

3. 使ってみる

wolfSSHをダウンロード、ビルドした後、ライブラリの使用法を示すための自動テストとサンプルプログラムがいくつかあります。

3.1 wolfSSH 単体テスト

wolfSSH ユニットテストは API を検証するために使用されます。 正常ケースと異常ケースの両方のテストケースが実行されます。 このテストは手動で実行することができ、さらに make や make check コマンドなどの他の自動化プロセスの一部として実行されます。

テストツールがテストに使用する証明書と鍵を見つけることができるように、すべての例とテストは wolfSSH ホームディレクトリから実行されなければなりません。

手動で単体テストを実行するには以下のコマンドを実行します。

$ ./tests/unit.test

または、

$ make check （autoconf を使用している場合）

3.2 wolfSSH echoserver

echoserver は SSH クライアントと接続され、端末に書き込まれたすべてのバイトを返します。 一般的に使用されている SSH クライアントは通常、入力された文字をディスプレイに表示しないので、表示されるテキストは表示されるテキストです。 行末文字変換は実行されないことに注意してください。

echoserver はポート 22222 をリッスンします。クライアント認証を行いません。 サーバーは入力終了時は無応答となり、control-C で停止しています。

$ ./examples/echoserver/echoserver

3.3 wolfSSH client

次のように、ユーザ名を指定して client を開始します。

$ ./examples/echoserver/echoserver

テストを実行するためのデフォルトの「username：password」は、次のいずれかです。

“ jack：fetchapail”または“ jill：upthehill”

デフォルトのポートは 22222 です。

4. ライブラリー設計

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。 開発上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。 時計はプラットフォームによって異なり、アプリケーションによって提供されるようになっています。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。

4.1 ディレクトリー構成

wolfSSHライブラリヘッダファイルはwolfsshディレクトリにあります。 ソースファイルに含める必要がある唯一のヘッダはwolfssh/ssh.hです。 例を以下に示します。

#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呼び出すには：

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。

5. 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もサーバに接続してい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 公開鍵を使用して検索する必要があるはずです。

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。

コールバックはいくつかの失敗または成功のうちの 1 つを返さなければなりません。ライブラリはロギング目的を除いてすべての失敗を同じように扱います。つまり、再度試行するクライアントに「User Authorization Failure」メッセージを返します。

パスワード検索では、プレーンテキスト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を確認し、一致した場合は成功を返します。成功すると、SSH ハンドシェイクは直ちに続行されます。現時点ではパスワードの変更はサポートされていません。

公開鍵検索では、クライアントからの公開鍵 BLOB 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されるものです。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。

5.1 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

authType パラメータの値は次のいずれかです。

WOLFSSH_USERAUTH_PASSWORD

WOLFSSH_USERAUTH_PUBLICKEY

authData は認証データへのポインタです。

5.4 項で WS_UserAuthData について説明します。

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もせず、感知しません。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。

5.2 コールバック関数の認証種別定数

以下は、authType パラメーターでユーザー認証コールバック関数に渡される値です。確認する認証データの種類に関するコールバック関数を導きます。 システムは、異なるユーザーに対してパスワードまたは公開鍵を使用することができます。

WOLFSSH_USERAUTH_PASSWORD

WOLFSSH_USERAUTH_PUBLICKEY

5.3 コールバック関数の返却値定数

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。

無効なコードは、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、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

5.4 コールバック関数のデータ型

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があります。 メソッド固有のフィールドは、ユーザー認証データ内の構造体の和集合です。

```
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;
```

5.4.1 パスワード

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

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。

5.4.2 公開鍵

```
typedef struct WS_UserAuthData_PublicKey {
```

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

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。 まず、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。

6. コールバック関数設定用 API

ユーザー認証コールバック機能を設定するには、以下の機能を使用します。

6.1 ユーザ認証コールバック関数の設定

void wolfSSH_SetUserAuth(WOLFSSH_CTX* ctx, WS_CallbackUserAuth cb);

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 wolfSSL 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。

6.2 ユーザ認証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。

6.3 ユーザ認証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。

6.4 Echoserver サンプルプログラムのユーザ認証

void* wolfSSH_GetUserAuthCtx(WOLFSSH* ssh);

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 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、公開鍵出力を 2 つ繋いだものです。

ssh-rsa AAAAB3NzaC1yc...d+Jl8wrAhfE4x hansel
ssh-rsa AAAAB3NzaC1yc...UoGCPlKuqcFMf 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 のいずれかの SHA-256 ハッシュのペアのリンクリストに格納されています。

設定ファイル内の公開鍵 BLOB は Base64 で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。

その後、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。

それはそれからユーザ名を見つけることを試みるリストを通って歩き、見つけられなければ無効ユーザエラーコードを戻します。

見つかった場合は、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。