wolfSSH User Manual

May 23, 2018
Version 1.2

Table of Contents:

Chapter 1: Introduction

This manual is written as a technical guide to the wolfSSH embedded library. It will

explain how to build and get started with wolfSSH, provide an overview of build options, features, support, and much more.

wolfSSH is an implementation of the SSH (Secure Shell) server written in C and uses the wolfCrypt library which is also available from wolfSSL. Furthermore, wolfSSH has been built from the ground up in order for it to have multi-platform use. This implementation is based off of the SSH v2 specification.

1.1 Protocol Overview

SSH is a layered set of protocols that provide multiplexed streams of data between two peers. Typically, it is used for securing a connection to a shell on the server. However, it is also commonly used to securely copy files between two machines or tunnel the X11 display protocol.

1.2 Why Choose wolfSSH?

The wolfSSH library is a lightweight SSHv2 server library written in ANSI C and targeted for embedded, RTOS, and resource-constrained environments - primarily because of its small size, speed, and feature set. It is commonly used in standard operating environments as well because of its royalty-free pricing and excellent cross platform support. wolfSSH supports the industry standard SSH v2 and offers progressive ciphers such as Poly1305, ChaCha20, NTRU, and SHA-3.
wolfSSH is powered by the wolfCrypt library. A version of the wolfCrypt cryptography library has been FIPS 140-2 validated (Certificate #2425). For additional information, visit the wolfCrypt FIPS FAQ or contact fips@wolfssl.com

Features

- SSH v2.0 (server)
- Minimum footprint size of 33kB
- Runtime memory usage between 1.4 and 2kB, not including a configurable receive buffer
- Multiple Hashing Functions: SHA-1, SHA-2 (SHA-256, SHA-384, SHA-512), BLAKE2b, Poly1305
- Block, Stream, and Authenticated Ciphers: AES (CBC, CTR, GCM, CCM), Camellia, ChaCha20

- Public Key Options: RSA, DH, EDH, NTRU
- ECC Support (ECDH and ECDSA with curves: NISTP256, NISTP384, NISTP521
- Curve25519 and Ed25519
- Client authentication support (RSA key, password)
- Simple API
- PEM and DER certificate support
- Hardware Cryptography Support: Intel AES-NI support, Intel AVX1/2, RDRAND,
- RDSEED, Cavium NITROX support, STM32F2/F4 hardware crypto support,
- Freescale CAU / mmCAU / SEC, Microchip PIC32MZ

Chapter 2: Building wolfSSH

wolfSSH is written with portability in mind and should generally be easy to build on most systems. If you have difficulty building, please don't hesitate to seek support through our support forums, https://www.wolfssl.com/forums, or contact us directly at support@wolfssl.com.

This section explains how to build wolfSSH on *nix-like and Windows environments, and provides guidance for building in a non-standard environment. You will find a getting started guide and example in section 3.

When using the autoconf/automake system to build, wolfSSH uses a single Makefile to build all parts and examples of the library, which is both simpler and faster than using Makefiles recursively.

2.1 Getting the Source Code

The most recent, up to date version can be downloaded from the GitHub website here: https://github.com/wolfSSL/wolfSSH

Either click the "Download ZIP" button or use the following command in your terminal:

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

2.2 wolfSSH Dependencies

Since wolfSSH is dependent on wolfCrypt, a configuration of wolfSSL is necessary. wolfSSL can be downloaded here: https://github.com/wolfSSL/wolfssl.
The simplest configuration of wolfSSL required for wolfSSH is the default build that can be built from the root directory of wolfSSL with the following commands:

```
$ ./autogen.sh (only if you cloned from GitHub)
$ ./configure --enable-ssh
$ make check
$ sudo make install
```

To use the key generation function in wolfSSH, wolfSSL will need to be configured with keygen: `--enable-keygen`.

If the bulk of wolfSSL code isn't desired, wolfSSL can be configured with the crypto only option: `--enable-cryptonly`.

2.3 Building on *nix

When building on Linux, *BSD, OS X, Solaris, or other *nix-like environments, use the autoconf system. To build wolfSSH run the following commands:

```
$ ./autogen.sh (only if you cloned from GitHub)
$ ./configure
$ make
$ make install
```

You can append build options to the configure command.
For a list of available configure options and their purposes run:

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

To build wolfSSH run:

```
$ make
```

To ensure that wolfSSH has been built correctly, check to see if all of the tests have passed with:

```
$ make check
```

To install wolfSSH run:

```
$ make install
```

You may need superuser privileges to install, in which case run the install with sudo:

```
$ sudo make install
```

If you want to build only the wolfSSH library located in wolfssh/src/ and not the additional items (examples and tests) you can run the following command from the wolfSSH root directory:

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

2.4 Building on Windows

The visual studio project file can be found at: https://github.com/wolfSSL/wolfssh/blob/master/ide/winvs/wolfssh.sln

The solution file, wolfssh.sln, facilitates building wolfSSH and its example and test programs. The solution provides both Debug and Release builds of Static and Dynamic 32- or 64-bit libraries. The file user_settings.h should be used in the wolfSSL build to configure it.

This project assumes that the wolfSSH and wolfSSL source directories are installed side-by-side and do not have the version number in their names:

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

User Macros for Building on Windows

The solution is using user macros to indicate the location of the wolfSSL library and headers. All paths are set to the default build destinations in the wolfssl64 solution. The user macro `wolfCryptDir` is used as the base path for finding the libraries. It is initially set to `..\..\..\..\wolfssl`. And then, for example, the additional include directories value for the API test project is set to `$(wolfCryptDir)`.

The wolfCryptDir path must be relative to the project files, which are all one directory down

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

The other user macros are the directories where the wolfSSL libraries for the different builds may be found. So the user macro `wolfCryptDllRelease64` is initially set to:

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

This value is used in the debugging environment for the echoserver's 64-bit DLL Release build is set to:

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

When you run the echoserver from the debugger, it finds the wolfSSL DLL in that directory.

2.5 Building in a non-standard environment

While not officially supported, we try to help users wishing to build wolfSSH in a non-standard environment, particularly with embedded and cross-compiled systems. Below are some notes on getting started with this:

1. The source and header files need to remain in the same directory structure as they are in the wolfSSH download package.

2. Some build systems will want to explicitly know where the wolfSSH header files are located, so you may need to specify that. They are located in the <wolfssh_root>/wolfssh directory. Typically, you can add the <wolfssh_root> directory to your include path to resolve header problems.

3. wolfSSH defaults to a little endian system unless the configure process detects big endian. Since users building in a non-standard environment aren't using the configure process, BIG_ENDIAN_ORDER will need to be defined if using a big endian system.

4. Try to build the library and let us know if you run into any problems. If you need help, contact us at support@wolfssl.com.

2.6 Cross Compiling

Many users on embedded platforms cross compile for their environment. The easiest way to cross compile the library is to use the configure system. It will generate a Makefile which can then be used to build wolfSSH.

When cross compiling, you'll need to specify the host to configure, such as:

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

You may also need to specify the compiler, linker, etc. that you want to use:

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

After correctly configuring wolfSSH for cross compilation you should be able to follow standard autoconf practices for building and installing the library:

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

If you have any additional tips or feedback for cross compiling wolfSSH, please let us know at info@wolfssl.com.

2.7 Install to Custom Directory

To setup a custom install directory for wolfSSL use the following:

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

This will place the library in ~/wolfSSL/lib and the includes in ~/wolfssl/include.
To set up a custom install directory for wolfSSH and specify the custom wolfSSL library and include directories use the following:

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

Make sure the paths above match your actual locations.

Chapter 3: Getting Started

After downloading and building wolfSSH, there are some automated test and example programs to show the uses of the library.

3.1 wolfSSH Unit Test

The wolfSSH unit test is used to verify the API. Both positive and negative test cases are performed. This test can be run manually and it additionally runs as part of other automated processes such as the `make` and `make check` commands.

All examples and tests must be run from the wolfSSH home directory so the test tools can find their certificates and keys.

To run the unit test manually:

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

or

```
$ make check (when using autoconf)
```

3.2 wolfSSH Echo Server

The echo server lets an SSH client connect to it and it returns every byte written to the terminal. The commonly used SSH client does not normally echo typed characters to the display so the text seen is the echoed text. Note, end of line character translation is not performed.

The echo server listens to port 22222. It does not authenticate the client. The server is silent on its end and is stopped with control-C.

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

3.3 wolfSSH Client

Starting the client with specific username:

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

The default “username:password” to run the test is either:
“jack:fetchapail” or “jill:upthehill”
The default port is 22222.

Chapter 4: Library Design

The wolfSSH library is meant to be included directly into an application. The primary use case in mind during development is replacing serial- or telnet-based menus on embedded devices. The library is agnostic to networking using I/O callbacks, but provides callbacks for *NIX and Windows networking by default as examples. Timing is platform specific and should be provided by the application, functions will be provided to perform actions on timeouts.

4.1 Directory Layout

The wolfSSH library header files are located in the **wolfssh** directory. The only header required to be included in a source file is **wolfssh/ssh.h**. An example is shown below.

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

The wolfSFTP library header file is also included in the wolfssh directory. To call this header file use:

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

All main source files are located in the **src** directory that resides in the root directory.

Chapter 5: wolfSSH User Authentication Callback

wolfSSH needs to be able to authenticate users connecting to the server no matter which environment the library is embedded. Lookups may need to be done using passwords or RSA public keys stored in a text file, database, or hard coded into the application.

wolfSSH provides a callback hook that receives the username, either the password or public key provided in the user authentication message and the requested authentication type. The callback function then performs the appropriate lookups and gives a reply. Providing a callback is required.

The callback should return one of several failures or a success. The library will treat all the failures the same except for logging purposes, i.e. return the User Authorization Failure message to the client who will try again.

For password lookups, the plaintext password is given to the callback function.The username and password should be checked and if they match, a success returned. On success, the SSH handshake continues immediately. Password changing is not supported at this time.

For public key lookups, the public key blob from the client is given to the callback function. The public key is checked against the server's list of valid client public keys. If the public key provided matches the known public key for that user. The wolfSSH library performs the actual validation of the user authentication signature following the process described in RFC 4252 §7.

Commonly for public keys, the server stores either the users' public keys as generated by the ssh-keygen utility or stores a fingerprint of the public key. This value for a user is what is compared. The client will provide a signature of the session ID and the user authentication request message using its private key; the server verifies this signature using the public key.

5.1 Callback Function Prototype

The prototype for the user authentication callback function is:

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

This function prototype is of the type:

`WS_CallbackUserAuth`

The parameter `authType` is either:

`WOLFSSH_USERAUTH_PASSWORD`

`WOLFSSH_USERAUTH_PUBLICKEY`

The parameter, `authData,` is a pointer to the authentication data.

See section 5.4 for a description of `WS_UserAuthData`

The parameter **`ctx`** is an application defined context; wolfSSH does nothing with and knows nothing about the data in the context, it only provides the context pointer to the callback function.

5.2 Callback Function Authentication Type Constants

The following are values passed to the user authentication callback function in the `authType` parameter. It guides the callback function as to the type of authentication data to check. A system could use either a password or public key for different users.

`WOLFSSH_USERAUTH_PASSWORD`

`WOLFSSH_USERAUTH_PUBLICKEY`

5.3 Callback Function Return Code Constants

The following are the return codes the callback function shall return to the library. The failure code indicates that nothing was done and the callback couldn't do any checking.

The invalid codes indicate why the user authentication is being rejected:

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

The library indicates only *success* or *failure* to the client, the specific failure type is only used for logging.

```
WOLFSSH_USERAUTH_SUCCESS
WOLFSSH_USERAUTH_FAILURE
WOLFSSH_USERAUTH_INVALID_USER
WOLFSSH_USERAUTH_INVALID_PASSWORD
WOLFSSH_USERAUTH_INVALID_PUBLICKEY
```

5.4 Callback Function Data Types

The client data is passed to the callback function in a structure called `WS_UserAuthData`. It contains pointers to the data in the message. Common fields are in this structure. Method specific fields are in a union of structures in the user authentication data.

```
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 Password

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

The `username` and `usernameSz` parameters are the username provided by the client and its size in octets.

The `password` and `passwordSz` fields are the client's password and its size in octets.

While set if provided by the client, the parameters `hasNewPassword`, `newPassword`, and `newPasswordSz` are not used. There is no mechanism to tell the client to change its password at this time.

5.4.2 Public Key

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

wolfSSH will support multiple public key algorithms. The publicKeyType member points to the algorithm name used.

The publicKey field points at the public key blob provided by the client.

The public key checking will have either one or two steps. First, if the hasSignature field is not set, there is no signature. Only verify the username and publicKey are expected and correct. This step is optional depending on client configuration, and can save from doing costly public key operations with an invalid user. Second, the hasSignature field is set and signature field points to the client signature. Again the username and publicKey should be checked. wolfSSH will automatically check the signature.

Each of the fields has a size value in octets.

Chapter 6: Callback Function Setup API

The following functions are used to set up the user authentication callback function.

6.1 Setting the User Authentication Callback Function

```
void wolfSSH_SetUserAuth(WOLFSSH_CTX* ctx, WS_CallbackUserAuth
cb);
```

The callback function is set on the wolfSSL CTX object that is used to create the wolfSSH session objects. All sessions using this CTX will use the same callback function. This context is not to be confused with the callback function's context.

6.2 Setting the User Authentication Callback Context Data

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

Each wolfSSH session may have its own user authentication context data or share some. The wolfSSH library knows nothing of the contents of this context data. It is up to the application to create, release, and if needed provide a mutex for the data. The callback receives this context data from the library.

6.3 Getting the User Authentication Callback Context Data

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

This returns the pointer to the user authentication context data stored in the provided wolfSSH session. This is not to be confused with the wolfSSH's context data used to create the session.

6.4 Example Echo Server User Authentication

The example echo server implements the authentication callback with sample users using passwords and public keys. The example callback, wsUserAuth, is set on the wolfSSH context:

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

The example password file (passwd.txt) is a simple list of usernames and passwords seperated with a colon respectively. The defaults that exist within this file are as follows.

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

The public key file are the concatenation of the public key outputs of running ssh-keygen twice.

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

All users authorization data is stored in a linked list of pairs of usernames and SHA-256 hashes of either the password or the public key blob.

The public key blobs in the configuration file are Base64 encoded and are decoded before hashing. The pointer to the list of username-hash pairs is stored into a new wolfSSH session:

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

The callback function first checks if the authType is either public key or a password, and returns the general user authentication failure error code if neither.

Then it hashes the public key or password passed in via the authData.

It then walks through the list trying to find the username, and if not found returns the invalid user error code.

If found, it compares the calculated hash of the public key or password passed in and the hash stored in the pair.

If they match, the function returns success, otherwise it returns the invalid password or public key error code.

Chapter 7: wolfSSH SFTP Beta Introduction

This document covers building and using the SFTP feature with wolfSSH.

Versions of libraries at the time of document creation:
wolfSSH version 1.0.2 (mid release with SFTP development)
wolfSSL version 3.14.0

Chapter 8: Building wolfSSH SFTP

It is assumed that wolfSSL has already been built to be used with wolfSSH. To see building instructions for wolfSSL view the file README.md located in the root wolfSSH directory.

To build wolfSSH with support for SFTP use --enable-sftp, in the case of building with autotools, or define the macro WOLFSSH_SFTP if building without autotools. An example of this would be

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

By default the internal buffer size for handling reads and writes for get and put commands is set to 1024 bytes. This value can be overwritten in the case that the application needs to consume less resources or in the case that a larger buffer is desired. To override the default size define the macro WOLFSSH_MAX_SFTP_RW at compile time. An example of setting it would be as follows:

```
    ./configure --enable-sftp
C_EXTRA_FLAGS='WOLFSSH_MAX_SFTP_RW=2048
```

Chapter 9: Using wolfSSH SFTP Apps

A SFTP server and client application are bundled with wolfSSH. Both applications get built by autotools when building the wolfSSH library with SFTP support. The server application is located in examples/echoserver/ and is called echoserver. The client application is located in wolfsftp/client/ and is called wolfsftp.

An example of starting up a server that would handle incoming SFTP client connections would be as follow:

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

Where the command is being ran from the root wolfSSH directory. This starts up a server that is able to handle both SSH and SFTP connections.

Starting the client with specific username:

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

The default “username:password” to run the test is either:
“jack:fetchapail” or “jill:upthehill”
The default port is 22222.

The following are commands that the client and server support:

```
Commands :
     cd  <string>                        change directory
     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
```

Chapter 10: Notes and Limitations

In portions of the implementation file attributes are not being considered and default attributes or mode values are used. Specifically in wolfSSH_SFTP_Open, getting timestamps from files, and all extended file attributes.

Chapter 11: Port Forwarding

11.1 Building wolfSSH with Port Forwarding

It is assumed that wolfSSL has already been built to be used with wolfSSH. To see building instructions for wolfSSL view the file README.md located in the root wolfSSH directory.

To build wolfSSH with support for port forwarding use --enable-fwd, in the case of building with autotools, or define the macro WOLFSSH_FWD if building without autotools. An example of this would be

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

11.2 Using wolfSSH Port Forwarding Example App

An example port forwarding application is bundled with wolfSSH. The application is built by autotools when building the wolfSSH library with port forwarding support. The application is located in examples/wolffwd/ and is called wolffwd.

An example of starting up a port forwarder that would connect to a local SSH server and forward a connection from port 12345 to a local server listening on port 11111 would be as follows:

```
$ ./examples/wolffwd/wolffwd -u <username> -f 12345 -t
11111
```

Where the command is being ran from the root wolfSSH directory. The username would be your username on the local machine and your login password.

This example can be used easily with the wolfSSL example client and server:

```
$ ./examples/server/server
$ ./examples/client/client -p 12345
```

The wolfSSL server is listening to port 11111 by default. The client connects to wolffwd on port 12345 and wolffwd opens a connection to the wolfSSL server on port 11111. “Hello wolfSSL! I hear you fa shizzle!”

Chapter 12: Licensing

11.1 Open Source

wolfSSL (formerly CyaSSL), yaSSL, wolfCrypt, yaSSH and TaoCrypt software are free software downloads and may be modified to the needs of the user as long as the user adheres to version two of the GPL License. The GPLv2 license can be found on the gnu.org website (http://www.gnu.org/licenses/old-licenses/gpl-2.0.html).

wolfSSH software is a free software download and may be modified to the needs of the user as long as the user adheres to version three of the GPL license. The GPLv3 license can be found on the gnu.org website (https://www.gnu.org/licenses/gpl.html).

11.2 Commercial Licensing

Businesses and enterprises who wish to incorporate wolfSSL products into proprietary appliances or other commercial software products for re-distribution must license commercial versions. Commercial licenses for wolfSSL, yaSSL, and wolfCrypt are available for $5,000 USD per end product or SKU. Licenses are generally issued for one product and include unlimited royalty-free distribution. Custom licensing terms are also available.

Commercial licenses are also available for wolfMQTT and wolfSSH. Please contact licensing@wolfssl.com with inquiries.

11.3 Support Packages

Support packages for wolfSSL products are available on an annual basis directly from wolfSSL. With three different package options, you can compare them side-by-side and choose the package that best fits your specific needs. Please see our Support Packages page (https://www.wolfssl.com/wolfSSL/Support/support_tiers.php) for more details.

Chapter 12: Support and Consulting

12.1 How to Get Support

For general product support, wolfSSL (formerly CyaSSL) maintains an online forum for the wolfSSL product family. Please post to the forums or contact wolfSSL directly with any questions.

wolfSSL (yaSSL) Forums: https://www.wolfssl.com/forums
Email Support: support@wolfssl.com

For information regarding wolfSSL products, questions regarding licensing, or general comments, please contact wolfSSL by emailing **info@wolfssl.com**.

12.1.1 Bugs Reports and Support Issues

If you are submitting a bug report or asking about a problem, please include the following information with your submission:

1. wolfSSL version number
2. Operating System version
3. Compiler version
4. The exact error you are seeing
5. A description of how we can reproduce or try to replicate this problem

With the above information, we will do our best to resolve your problems. Without this information, it is very hard to pinpoint the source of the problem. wolfSSL values your feedback and makes it a top priority to get back to you as soon as possible.

12.2 Consulting

wolfSSL offers both on and off site consulting - providing feature additions, porting, a Competitive Upgrade Program (see section 15.2.2), and design consulting.

12.2.1 Feature Additions and Porting

We can add additional features that you may need which are not currently offered in our products on a contract or co-development basis. We also offer porting services on our products to new host languages or new operating environments.

12.2.2 Competitive Upgrade Program

We will help you move from an outdated or expensive SSL/TLS library to wolfSSL with low cost and minimal disturbance to your code base.

Program Outline:

1. You need to currently be using a commercial competitor to wolfSSL.
2. You will receive up to one week of on-site consulting to switch out your old SSL library for wolfSSL. Travel expenses are not included.
3. Normally, up to one week is the right amount of time for us to make the replacement in your code and do initial testing. Additional consulting on a replacement is available as needed.
4. You will receive the standard wolfSSL royalty free license to ship with your product.
5. The price is $10,000.

The purpose of this program is to enable users who are currently spending too much on their embedded SSL implementation to move to wolfSSL with ease. If you are interested in learning more, then please contact us at info@wolfssl.com.

12.2.3 Design Consulting

If your application or framework needs to be secured with SSL/TLS but you are uncertain about how the optimal design of a secured system would be structured, we can help!

We offer design consulting for building SSL/TLS security into devices using wolfSSL. Our consultants can provide you with the following services:

1. *Assessment:* An evaluation of your current SSL/TLS implementation. We can give you advice on your current setup and how we think you could improve upon this by using wolfSSL.

2. *Design:* Looking at your system requirements and parameters, we'll work closely with you to make recommendations on how to implement wolfSSL into your application such that it provides you with optimal security.

If you would like to learn more about design consulting for building SSL into your application or device, please contact info@wolfssl.com for more information.

Chapter 13: wolfSSH Updates

13.1 Product Release Information

We regularly post update information on Twitter. For additional release information, you can keep track of our projects on GitHub, follow us on Facebook, or follow our daily blog.

wolfSSH on GitHub	https://www.github.com/wolfssl/wolfssh
wolfSSL on Twitter	http://twitter.com/wolfSSL
wolfSSL on Facebook	http://www.facebook.com/wolfSSL
wolfSSL on Reddit	https://www.reddit.com/r/wolfssl/
Daily Blog	https://wolfssl.com/wolfSSL/Blog/Blog.html

Chapter 14: API Reference

This section describes the public application program interfaces for the wolfSSH library.

14.1 Error Codes

14.1.1 WS_ErrorCodes (enum)

The following API response codes are defined in: wolfssh/wolfssh/error.h and describe the different types of errors that can occur.

- WS_SUCCESS (0): Function success
- WS_FATAL_ERROR (-1): General function failure
- WS_BAD_ARGUMENT (-2): Function argument out of bounds
- WS_MEMORY_E (-3): Memory allocation error
- WS_BUFFER_E (-4): Input/output buffer size error
- WS_PARSE_E (-5): General parsing error
- WS_NOT_COMPILED (-6): Feature not compiled in
- WS_OVERFLOW_E (-7): Would overflow if continued
- WS_BAD_USAGE (-8): Bad example usage
- WS_SOCKET_ERROR_E (-9): Socket error
- WS_WANT_READ (-10): IO callback would read block error
- WS_WANT_WRITE (-11): IO callback would write block error
- WS_RECV_OVERFLOW_E (-12): Received buffer overflow
- WS_VERSION_E (-13): Peer using wrong version of SSH
- WS_SEND_OOB_READ_E (-14): Attempted to read buffer out of bounds
- WS_INPUT_CASE_E (-15): Bad process input state, programming error
- WS_BAD_FILETYPE_E (-16): Bad filetype
- WS_UNIMPLEMENTED_E (-17): Feature not implemented
- WS_RSA_E (-18): RSA buffer error
- WS_BAD_FILE_E (-19): Bad file
- WS_INVALID_ALGO_ID (-20): invalid algorithm ID
- WS_DECRYPT_E (-21): Decrypt error
- WS_ENCRYPT_E (-22): Encrypt error
- WS_VERIFY_MAC_E (-23): verify mac error
- WS_CREATE_MAC_E (-24): Create mac error

- WS_RESOURCE_E (-25): Insufficient resources for new channel
- WS_INVALID_CHANTYPE (-26): Invalid channel type
- WS_INVALID_CHANID(-27): Peer requested invalid channel ID
- WS_INVALID_USERNAME(-28): Invalid user name
- WS_CRYPTO_FAILED(-29): Crypto action failed
- WS_INVALID_STATE_E(-30): Invalid State
- WC_EOF(-31): End of File
- WS_INVALID_PRIME_CURVE(-32): Invalid prime curve in ECC
- WS_ECC_E(-33): ECDSA buffer error
- WS_CHANOPEN_FAILED(-34): Peer returned channel open failure
- WS_REKEYING(-35): Rekeying with peer
- WS_CHANNEL_CLOSED(-36): Channel closed

14.1.2 WS_IOerrors (enum)

These are the return codes the library expects to receive from a user-provided I/O callback. Otherwise the library expects the number of bytes read or written from the I/O action.

- WS_CBIO_ERR_GENERAL (-1): General unexpected error
- WS_CBIO_ERR_WANT_READ (-2): Socket read would block, call again
- WS_CBIO_ERR_WANT_WRITE (-2): Socket write would block, call again
- WS_CBIO_ERR_CONN_RST (-3): Connection reset
- WS_CBIO_ERR_ISR (-4): Interrupt
- WS_CBIO_ERR_CONN_CLOSE (-5): Connection closed or EPIPE
- WS_CBIO_ERR_TIMEOUT (-6): Socket timeout

14.2 Initialization / Shutdown

wolfSSH_Init()

Synopsis

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

Description

Initializes the wolfSSH library for use. Must be called once per application and before any other calls to the library.

Return Values

WS_SUCCESS
WS_CRYPTO_FAILED

Parameters

None

See Also

wolfSSH_Cleanup()

wolfSSH_Cleanup()

Synopsis

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

Description

Cleans up the wolfSSH library when done. Should be called at before termination of the application. After calling, do not make any more calls to the library.

Return Values

WS_SUCCESS
WS_CRYPTO_FAILED

Parameters

None

See Also

wolfSSH_Init()

14.3 Debugging output functions

wolfSSH_Debugging_ON()

Synopsis

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

Description

Enables debug logging during runtime. Does nothing when debugging is disabled at build time.

Return Values

None

Parameters

None

See Also

wolfSSH_Debugging_OFF()

wolfSSH_Debugging_OFF()

Synopsis

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

Description

Disables debug logging during runtime. Does nothing when debugging is disabled at build time.

Return Values

None

Parameters

None

See Also

wolfSSH_Debugging_ON()

14.4 Context Functions

wolfSSH_CTX_new()

Synopsis

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

Description

Creates a wolfSSH context object. This object can be configured and then used as a factory for wolfSSH session objects.

Return Values

WOLFSSH_CTX* – returns pointer to allocated WOLFSSH_CTX object or NULL

Parameters

side – indicate client side (unimplemented) or server side
heap – pointer to a heap to use for memory allocations

See Also

wolfSSH_wolfSSH_CTX_free()

wolfSSH_CTX_free()

Synopsis

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

Description

Deallocates a wolfSSH context object.

Return Values

None

Parameters

ctx – the wolfSSH context used to initialize the wolfSSH session

See Also

wolfSSH_wolfSSH_CTX_new()

wolfSSH_CTX_SetBanner()

Synopsis

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

Description

Sets a banner message that a user can see.

Return Values

WS_BAD_ARGUMENT
WS_SUCCESS

Parameters

ssh - Pointer to wolfSSH session
newBanner - The banner message text.

wolfSSH_CTX_UsePrivateKey_buffer()

Synopsis

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

Description

This function loads a private key buffer into the SSH context. It is called with a buffer as input instead of a file. The buffer is provided by the **in** argument of size **inSz**. The argument **format** specifies the type of buffer: **WOLFSSH_FORMAT_ASN1** or **WOLFSSL_FORMAT_PEM** (unimplemented at this time).

Return Values

WS_SUCCESS
WS_BAD_ARGUMENT – at least one of the parameters is invalid
WS_BAD_FILETYPE_E – wrong format
WS_UNIMPLEMENTED_E – support for PEM format not implemented
WS_MEMORY_E – out of memory condition
WS_RSA_E – cannot decode RSA key
WS_BAD_FILE_E – cannot parse buffer

Parameters

ctx – pointer to the wolfSSH context
in – buffer containing the private key to be loaded
inSz – size of the input buffer
format – format of the private key located in the input buffer

See Also

wolfSSH_UseCert_buffer()
wolfSSH_UseCaCert_buffer()

14.5 SSH Session Functions

wolfSSH_new()

Synopsis

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

Description

Creates a wolfSSH session object. It is initialized with the provided wolfSSH context.

Return Values

WOLFSSH* – returns pointer to allocated WOLFSSH object or NULL

Parameters

ctx – the wolfSSH context used to initialize the wolfSSH session

See Also

wolfSSH_free()

wolfSSH_free()

Synopsis

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

Description

Deallocates a wolfSSH session object.

Return Values

None

Parameters

ssh – session to deallocate

See Also

wolfSSH_new()

wolfSSH_set_fd()

Synopsis

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

Description

Assigns the provided file descriptor to the ssh object. The ssh session will use the file descriptor for network I/O in the default I/O callbacks.

Return Values

WS_SUCCESS
WS_BAD_ARGUMENT – one of the parameters is invalid

Parameters

ssh – session to set the fd
fd – file descriptor for the socket used by the session

See Also

wolfSSH_get_fd()

wolfSSH_get_fd()

Synopsis

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

Description

This function returns the file descriptor (**fd**) used as the input/output facility for the SSH connection. Typically this will be a socket file descriptor.

Return Values

int – file descriptor
WS_BAD_ARGUEMENT

Parameters

ssh – pointer to the SSL session.

See Also

wolfSSH_set_fd()

14.6 Data High Water Mark Functions

wolfSSH_SetHighwater()

Synopsis

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

Description

Sets the highwater mark for the ssh session.

Return Values

WS_SUCCESS
WS_BAD_ARGUMENT

Parameters

ssh - Pointer to wolfSSH session
highwater - data indicating the highwater security mark

wolfSSH_GetHighwater()

Synopsis

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

Description

Returns the highwater security mark

Return Values

word32 - The highwater security mark.

Parameters

ssh - Pointer to wolfSSH session

wolfSSH_SetHighwaterCb()

Synopsis

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

Description

The wolfSSH_SetHighwaterCb function sets the highwater security mark for the SSH session as well as the high water call back.

Return Values

none

Parameters

ctx – The wolfSSH context used to initialize the wolfSSH session.
highwater - The highwater security mark.
cb - The call back highwater function.

wolfSSH_SetHighwaterCtx()

Synopsis

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

Description

The wolfSSH_SetHighwaterCTX function sets the highwater security mark for the given context.

Return Values

none

Parameters

ssh - pointer to wolfSSH session
ctx - pointer to highwater security mark in the wolfSSH context.

wolfSSH_GetHighwaterCtx()

Synopsis

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

Description

The wolfSSH_GetHighwaterCtx() returns the highwaterCtx security mark from the SSH session.

Return Values

void* - the highwater security mark
NULL - if there is an error with the WOLFSSH object.

Parameters

ssh - pointer to WOLFSSH object

14.7 Error Checking

wolfSSH_get_error()

Synopsis

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

Description

Returns the error set in the wolfSSH session object.

Return Values

WS_ErrorCodes (enum)

Parameters

ssh – pointer to WOLFSSH object

See Also

wolfSSH_get_error_name()

wolfSSH_get_error_name()

Synopsis

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

Description

Returns the name of the error set in the wolfSSH session object.

Return Values

const char* – error name string

Parameters

ssh – pointer to WOLFSSH object

See Also

wolfSSH_get_error()

wolfSSH_ErrorToName()

Synopsis

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

Description

Returns the name of an error when called with an error number in the parameter.

Return Values

const char* – name of error string

Parameters

err - the int value of the error

14.8 I/O Callbacks

wolfSSH_SetIORecv()

Synopsis

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

Description

This function registers a receive callback for wolfSSL to get input data.

Return Values

None

Parameters

ctx – pointer to the SSH context
cb – function to be registered as the receive callback for the wolfSSH context, **ctx**. The signature of this function must follow that as shown above in the Synopsis section.

wolfSSH_SetIOSend()

Synopsis

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

Description

This function registers a send callback for wolfSSL to write output data.

Return Values

None

Parameters

ctx – pointer to the wolfSSH context
cb – function to be registered as the send callback for the wolfSSH context, **ctx**. The signature of this function must follow that as shown above in the Synopsis section.

wolfSSH_SetIOReadCtx()

Synopsis

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

Description

This function registers a context for the SSH session receive callback function.

Return Values

None

Parameters

ssh – pointer to WOLFSSH object
ctx – pointer to the context to be registered with the SSH session (**ssh**) receive callback function.

wolfSSH_SetIOWriteCtx()

Synopsis

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

Description

This function registers a context for the SSH session's send callback function.

Return Values

None

Parameters

ssh – pointer to WOLFSSH session.
ctx – pointer to be registered with the SSH session's (**ssh**) send callback function.

wolfSSH_GetIOReadCtx()

Synopsis

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

Description

This function return the ioReadCtx member of the WOLFSSH structure.

Return Values

Void* - pointer to the `ioReadCtx` member of the WOLFSSH structure.

Parameters

ssh – pointer to WOLFSSH object

wolfSSH_GetIOWriteCtx()

Synopsis

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

Description

This function returns the ioWriteCtx member of the WOLFSSH structure.

Return Values

Void* – pointer to the `ioWriteCtx` member of the WOLFSSH structure.

Parameters

ssh – pointer to WOLFSSH object

14.9 User Authentication

wolfSSH_SetUserAuth()

Synopsis

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

Description

The wolfSSH_SetUserAuth() function is used to set the user authentication for the current wolfSSH context if the context does not equal NULL.

Return Values

None

Parameters

ctx – pointer to the wolfSSH context
cb – call back function for the user authentication

wolfSSH_SetUserAuthCtx()

Synopsis

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

Description

The wolfSSH_SetUserAuthCtx() function is used to set the value of the user authentication context in the SSH session.

Return Values

None

Parameters

ssh – pointer to WOLFSSH object
userAuthCtx – pointer to the user authentication context

wolfSSH_GetUserAuthCtx()

Synopsis

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

Description

The wolfSSH_GetUserAuthCtx() function is used to return the pointer to the user authentication context.

Return Values

Void* – pointer to the user authentication context
Null – returns if ssh is equal to NULL

Parameters

ssh – pointer to WOLFSSH object

14.10 Set Username

wolfSSH_SetUsername()

Synopsis

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

Description

Sets the username required for the SSH connection.

Return Values

WS_BAD_ARGUMENT
WS_SUCCESS
WS_MEMORY_E

Parameters

ssh - Pointer to wolfSSH session
username - The input username for the SSH connection.

14.11 Connection Functions

wolfSSH_accept()

Synopsis

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

Description

wolfSSH_accept is called on the server side and waits for an SSH client to initiate the SSH handshake.

wolfSSL_accept() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSH_accept() will return when the underlying I/O could not satisfy the needs of wolfSSH_accept to continue the handshake. In this case, a call to wolfSSH_get_error() will yield either **WS_WANT_READ** or **WS_WANT_WRITE**. The calling process must then repeat the call to wolfSSH_accept when data is available to read and wolfSSH will pick up where it left off. When using a non-blocking socket, nothing needs to be done, but select() can be used to check for the required condition.

If the underlying I/O is blocking, wolfSSH_accept() will only return once the handshake has been finished or an error occurred.

Return Values

WS_SUCCESS - The function succeeded.
WS_BAD_ARGUMENT - A parameter value was null.
WS_FATAL_ERROR – There was an error, call wolfSSH_get_error() for more detail

Parameters

ssh – pointer to the wolfSSH session

See Also

wolfSSH_stream_read()

wolfSSH_stream_send()

wolfSSH_connect()

Synopsis

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

Description

This function is called on the client side and initiates an SSH handshake with a server. When this function is called, the underlying communication channel has already been set up.

wolfSSH_connect() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSH_connect() will return when the underlying I/O could not satisfy the needs of wolfSSH_connect to continue the handshake. In this case, a call to wolfSSH_get_error() will yield either **WS_WANT_READ** or **WS_WANT_WRITE**. The calling process must then repeat the call to wolfSSH_connect() when the underlying I/O is ready and wolfSSH will pick up where it left off. When using a non-blocking socket, nothing needs to be done, but select() can be used to check for the required condition.

If the underlying I/O is blocking, wolfSSH_connect() will only return once the handshake has been finished or an error occurred.

Return Values

WS_BAD_ARGUMENT
WS_FATAL_ERROR
WS_SUCCESS - This will return if the call is successful.

Parameters

ssh - Pointer to wolfSSH session

wolfSSH_shutdown()

Synopsis

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

Description

Closes and disconnects the SSH channel.

Return Values

WS_BAD_ARGUMENT - returned if the parameter is NULL
WS_SUCCES - returns when everything has been correctly shutdown

Parameters

ssh - Pointer to wolfSSH session

wolfSSH_stream_read()

Synopsis

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

Description

wolfSSH_stream_read reads up to **bufSz** bytes from the internal decrypted data stream buffer. The bytes are removed from the internal buffer.

wolfSSH_stream_read() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSH_stream_read() will return when the underlying I/O could not satisfy the needs of wolfSSH_stream_read to continue the read. In this case, a call to wolfSSH_get_error() will yield either **WS_WANT_READ** or **WS_WANT_WRITE**. The calling process must then repeat the call to wolfSSH_stream_read when data is available to read and wolfSSH will pick up where it left off. When using a non-blocking socket, nothing needs to be done, but select() can be used to check for the required condition.

If the underlying I/O is blocking, wolfSSH_stream_read() will only return when data is available or an error occurred.

Return Values

>0 – number of bytes read upon success
0 – returned on socket failure caused by either a clean connection shutdown or a socket.
WS_BAD_ARGUMENT – returns if one or more parameters is equal to NULL
WS_EOF – returns when end of stream is reached
WS_FATAL_ERROR – there was an error, call **wolfSSH_get_error()** for more detail

Parameters

ssh – pointer to the wolfSSH session

buf – buffer where wolfSSH_stream_read() will place the data
bufSz – size of the buffer

See Also

wolfSSH_accept()
wolfSSH_stream_send()

wolfSSH_stream_send()

Synopsis

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

Description

wolfSSH_stream_send writes **bufSz** bytes from buf to the SSH stream data buffer. The bytes are removed from the internal buffer.

wolfSSH_stream_send() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSH_stream_send() will return when the underlying I/O could not satisfy the needs of wolfSSH_stream_send to continue. In this case, a call to wolfSSH_get_error() will yield either **WS_WANT_READ** or **WS_WANT_WRITE**. The calling process must then repeat the call to wolfSSH_stream_send when the socket it ready to send and wolfSSH will pick up where it left off. When using a non-blocking socket, nothing needs to be done, but select() can be used to check for the required condition.

If the underlying I/O is blocking, wolfSSH_stream_send() will only return when the data has been sent or an error occurred.

Return Values

>0 – number of bytes written upon success
0 – returned on socket failure caused by either a clean connection shutdown or a socket error, call **wolfSSH_get_error()** for more detail
WS_FATAL_ERROR – there was an error, call wolfSSH_get_error() for more detail
WS_BAD_ARGUMENT if any of the parameters is null

Parameters

ssh – pointer to the wolfSSH session
buf – buffer wolfSSH_stream_send() will send

bufSz – size of the buffer

See Also

wolfSSH_accept()
wolfSSH_stream_read()

wolfSSH_stream_exit()

Synopsis

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

Description

This function is used to exit the SSH stream.

Return Values

WS_BAD_ARGUMENT - returned if a parameter value is NULL
WS_SUCCESS - returns if function was a success

Parameters

ssh – Pointer to wolfSSH session
status – the status of the SSH connection

wolfSSH_TriggerKeyExchange()

Synopsis

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

Description

Triggers key exchange process. Prepares and sends packet of allocated handshake info.

Return Values

WS_BAD_ARGUEMENT – if **ssh** is NULL
WS_SUCCESS

Parameters

ssh – pointer to the wolfSSH session

14.12 Testing Functions

wolfSSH_GetStats()

Synopsis

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

Description

Updates **txCount**, **rxCount**, **seq**, and **peerSeq** with their respective **ssh** session statistics.

Return Values

none

Parameters

ssh – pointer to the wolfSSH session
txCount – address where total transferred bytes in **ssh** session are stored.
rxCount – address where total received bytes in **ssh** session are stored.
seq – packet sequence number is initially 0 and is incremented after every packet
peerSeq – peer packet sequence number is initially 0 and is incremented after every packet

wolfSSH_KDF()

Synopsis

```
    #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);
```

Description

This is used so that the API test can do known answer tests for the key derivation.

The Key Derivation Function derives a symmetric **key** based on source keying material, **k** and **h**. Where **k** is the Diffie-Hellman shared secret and **h** is the hash of the handshake that was produced during initial key exchange. Multiple types of keys could be derived which are specified by the **keyId** and **hashId**.

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

Return Values

WS_SUCCESS
WS_CRYPTO_FAILED

Parameters

hashId – type of hash to generate keying material.
e.g. (WC_HASH_TYPE_SHA and WC_HASH_TYPE_SHA256)
keyId – letter A - F to indicate which key to make
key – generated key used for comparisons to expected key

keySz – needed size of **key**
k – shared secret from the Diffie-Hellman key exchange
kSz – size of the shared secret (**k**)
h – hash of the handshake that was produced during key exchange
hSz – size of the hash (**h**)
sessionId – unique identifier from first **h** calculated.
sessionIdSz – size of the **sessionId**

14.13 Session Functions

wolfSSH_GetSessionType()

Synopsis

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

Description

The wolfSSH_GetSessionType() is used to return the type of session

Return Values

WOLFSSH_SESSION_UNKNOWN
WOLFSSH_SESSION_SHELL
WOLFSSH_SESSION_EXEC
WOLFSSH_SESSION_SUBSYSTEM

Parameters

ssh - pointer to wolfSSH session

wolfSSH_GetSessionCommand()

Synopsis

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

Description

This function is used to return the current command in the session.

Return Values

const char* - Pointer to command

Parameters

ssh - pointer to wolfSSH session

14.14 Port Forwarding Functions

wolfSSH_ChannelFwdNew()

Synopsis

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

Description

Sets up a TCP/IP forwarding channel on a WOLFSSH session. When the SSH session is connected and authenticated, a local listener is created on the interface for address *host* on port *hostPort*. Any new connections on that listener will trigger a new channel request to the SSH server to establish a connection to *host* on port *hostPort*.

Return Values

WOLFSSH_CHAN* – NULL on error or new channel record

Parameters

ssh – wolfSSH session
host – host address to bind listener
hostPort – host port to bind listener
origin – IP address of the originating connection
originPort – port number of the originating connection

wolfSSH_ChannelFree()

Synopsis

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

Description

Releases the memory allocated for the channel *channel*. The channel is removed from its session's channel list.

Return Values

int – error code

Parameters

channel – wolfSSH channel to free

wolfSSH_worker()

Synopsis

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

Description

The wolfSSH worker function babysits the connection and as data is received processes it. SSH sessions have many bookkeeping messages for the session and this takes care of them automatically. When data for a particular channel is received, the worker places the data into the channel. (The function wolfSSH_stream_read() does much the same but also returns the receive data for a single channel.) wolfSSH_worker() will perform the following actions:

1. Attempt to send any pending data in the *outputBuffer*.
2. Call *DoReceive()* on the session's socket.
3. If data is received for a particular channel, return data received notice and set the channel ID.

Return Values

int – error or status

WS_CHANNEL_RXD – data has been received on a channel and the ID is set

Parameters

ssh – pointer to the wolfSSH session
id – pointer to the location to save the ID value

wolfSSH_ChannelGetId()

Synopsis

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

Description

Given a channel, returns the ID or peer's ID for the channel.

Return Values

int – error code

Parameters

channel – pointer to channel
id – pointer to location to save the ID value
peer – either self (my channel ID) or peer (my peer's channel ID)

wolfSSH_ChannelFind()

Synopsis

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

Description

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

Return Values

WOLFSSH_CHANNEL* – pointer to the channel, NULL if the ID isn't in the list

Parameters

ssh – wolfSSH session
id – channel ID to find
peer – either self (my channel ID) or peer (my peer's channel ID)

wolfSSH_ChannelRead()

Synopsis

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

Description

Copies data out of a channel object.

Return Values

int – bytes read

- **>0** – number of bytes read upon success
- **0** – returns on socket failure cause by either a clean connection shutdown or a socket error, call wolfSSH_get_error() for more detail
- **WS_FATAL_ERROR** – there was some other error, call wolfSSH_get_error() for more detail

Parameters

channel – pointer to the wolfSSH channel
buf – buffer where wolfSSH_ChannelRead will place the data
bufSz – size of the buffer

wolfSSH_ChannelSend()

Synopsis

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

Description

Sends data to the peer via the specified channel. Data is packaged into a channel data message. This will send as much data as possible via the peer socket. If there is more to be sent, calls to *wolfSSH_worker()* will continue sending more data for the channel to the peer.

Return Values

int – bytes sent

- **>0** – number of bytes sent upon success
- **0** – returns on socket failure cause by either a clean connection shutdown or a socket error, call wolfSSH_get_error() for more detail
- **WS_FATAL_ERROR** – there was some other error, call wolfSSH_get_error() for more detail

Parameters

channel – pointer to the wolfSSH channel
buf – buffer wolfSSH_ChannelSend() will send
bufSz – size of the buffer

wolfSSH_ChannelExit()

Synopsis

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

Description

Terminates a channel, sending the close message to the peer, marks the channel as closed. This does not free the channel and it remains on the channel list. After closure, data can not be sent on the channel, but data may still be available to be received. (At the moment, it sends EOF, close, and deletes the channel.)

Return Values

int – error code

Parameters

channel – wolfSSH session channel

wolfSSH_ChannelNext()

Synopsis

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

Description

Returns the next channel after *channel* in *ssh*'s channel list. If *channel* is NULL, the first channel from the channel list for *ssh* is returned.

Return Values

WOLFSSH_CHANNEL* – pointer to either the first channel, next channel, or NULL

Parameters

ssh – wolfSSH session
channel – wolfSSH session channel

Chapter 15: wolfSSL SFTP API Reference

15.1 Connection Functions

---

wolfSSH_SFTP_accept()

Synopsis:

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

Description:

Function to handle an incoming connection request from a client.

Return Values:

Returns WS_SFTP_COMPLETE on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection

Example:

```
WOLFSSH* ssh;

//create new WOLFSSH structure
...

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

See Also:

wolfSSH_SFTP_free()
wolfSSH_new()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_connect()

Synopsis:

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

int wolfSSH_SFTP_connect(WOLFSSH* ssh);
```

Description:

Function for initiating a connection to a SFTP server.

Return Values:

WS_SFTP_COMPLETE: on success.

Parameters:

ssh - pointer to WOLFSSH structure to be used for connection

Example:

```
WOLFSSH* ssh;

//after creating a new WOLFSSH structrue

wolfSSH_SFTP_connect(ssh);
```

See Also:

wolfSSH_SFTP_accept()
wolfSSH_new()
wolfSSH_free()

wolfSSH_SFTP_negotiate()

Synopsis:

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

Description:

Function to handle either an incoming connection from client or to send out a connection request to a server. It is dependent on which side of the connection the created WOLFSSH structure is set to for which action is performed.

Return Values:

Returns WS_SUCCESS on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection

Example:

```
WOLFSSH* ssh;

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

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

See Also:

wolfSSH_SFTP_free()

wolfSSH_new()
wolfSSH_SFTP_connect()
wolfSSH_SFTP_accept()

15.2 Protocol Level Functions

wolfSSH_SFTP_RealPath()

Synopsis:

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

Description:

Function to send REALPATH packet to peer. It gets the name of the file returned from peer.

Return Values:

Returns a pointer to a `WS_SFTPNAME` structure on success and `NULL` on error.

Parameters:

ssh - pointer to `WOLFSSH` structure used for connection
dir - directory / file name to get real path of

Example:

```
WOLFSSH* ssh;

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

if (wolfSSH_SFTP_read(ssh) != WS_SUCCESS) {

//handle error case

}
```

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_Close()

Synopsis:

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

Description:

Function to to send a close packet to the peer.

Return Values:

WS_SUCCESS on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
handle - handle to try and close
handleSz - size of handle buffer

Example:

```
    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
    }
```

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_Open()

Synopsis:

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

Description:

Function to to send an open packet to the peer. This sets handleSz with the size of resulting buffer and gets the resulting handle from the peer and places it in the buffer handle.

Available reasons for open:
WOLFSSH_FXF_READ
WOLFSSH_FXF_WRITE
WOLFSSH_FXF_APPEND
WOLFSSH_FXF_CREAT
WOLFSSH_FXF_TRUNC
WOLFSSH_FXF_EXCL

Return Values:

WS_SUCCESS on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
dir - name of file to open
reason - reason for opening the file
atr - initial attributes for file
handle - resulting handle from open
handleSz - gets set to the size of resulting handle

Example:

```
    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

    }
```

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_SendReadPacket()

Synopsis:

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

Description:

Function to to send a read packet to the peer. The buffer handle should contain the result of a previous call to `wolfSSH_SFTP_Open`. The resulting bytes from a read are placed into the “out” buffer.

Return Values:

Returns the number of bytes read on success.
A negative value is returned on failure.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
handle - handle to try and read from
handleSz - size of handle buffer
ofst - offset to start reading from
out - buffer to hold result from read
outSz - size of out buffer

Example:

```
    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
```

See Also:

wolfSSH_SFTP_SendWritePacket()
wolfSSH_SFTP_Open()

wolfSSH_SFTP_SendWritePacket()

Synopsis:

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

      int wolfSSH_SFTP_SendWritePacket(WOLFSSH* ssh, byte*
handle, word32
      handleSz, word64 ofst, byte* out, word32 outSz);
```

Description:

Function to send a write packet to the peer.
The buffer handle should contain the result of a previous call to `wolfSSH_SFTP_Open()`.

Return Values:

Returns the number of bytes written on success.
A negative value is returned on failure.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
handle - handle to try and read from
handleSz - size of handle buffer
ofst - offset to start reading from
out - buffer to send to peer for writing
outSz - size of out buffer

Example:

```
    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
```

See Also:

wolfSSH_SFTP_SendReadPacket()
wolfSSH_SFTP_Open()

wolfSSH_SFTP_STAT()

Synopsis:

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

Description:

Function to send a STAT packet to the peer. This will get the attributes of file or directory. If the file or attribute does not exist the peer will return resulting in this function returning an error value.

Return Values:

WS_SUCCESS on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
dir - NULL terminated name of file or directory to get attributes of
atr - resulting attributes are set into this structure

Example:

```
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

}
```

See Also:

wolfSSH_SFTP_LSTAT()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_LSTAT()

Synopsis:

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

Description:

Function to send a LSTAT packet to the peer. This will get the attributes of file or directory. It follows symbolic links where a STAT packet will not follow symbolic links. If the file or attribute does not exist the peer will return resulting in this function returning an error value.

Return Values:

`WS_SUCCESS` on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
dir - NULL terminated name of file or directory to get attributes of
atr - resulting attributes are set into this structure

Example:

```
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
}
```

See Also:

wolfSSH_SFTP_STAT()
wolfSSH_SFTP_connect()

wolfSSH_SFTPNAME_free()

Synopsis:

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

void wolfSSH_SFTPNAME_free(WS_SFTPNMAE* name);
```

Description:

Function to free a single `WS_SFTPNAME` node. Note that if this node is in the middle of a list of nodes then the list will be broken.

Return Values:

None

Parameters:

name - structure to be free'd

Example:

```
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);
}
```

See Also:

wolfSSH_SFTPNAME_list_free

wolfSSH_SFTPNAME_list_free()

Synopsis:

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

Description:

Function to free a all `WS_SFTPNAME` nodes in a list.

Return Values:

None

Parameters:

name - head of list to be free'd

Example:

```
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);
}
```

See Also:

wolfSSH_SFTPNAME_free()

15.3 Reget / Reput Functions

wolfSSH_SFTP_SaveOfst()

Synopsis:

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

Description:

Function to save an offset for an interrupted get or put command. The offset can be recovered by calling `wolfSSH_SFTP_GetOfst`

Return Values:

Returns `WS_SUCCESS` on success.

Parameters:

ssh - pointer to WOLFSSH structure for connection
from - NULL terminated string of source path
to - NULL terminated string with destination path
ofst - offset into file to be saved

Example:

```
    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
    }
```

See Also:

wolfSSH_SFTP_GetOfst()
wolfSSH_SFTP_Interrupt()

wolfSSH_SFTP_GetOfst()

Synopsis:

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

Description:

Function to retrieve an offset for an interrupted get or put command.

Return Values:

Returns offset value on success. If not stored offset is found then 0 is returned.

Parameters:

ssh - pointer to WOLFSSH structure for connection
from - NULL terminated string of source path
to - NULL terminated string with destination path

Example:

```
    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
```

See Also:

wolfSSH_SFTP_SaveOfst()
wolfSSH_SFTP_Interrup()

wolfSSH_SFTP_ClearOfst()

Synopsis:

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

Description:

Function to clear all stored offset values.

Return Values:

WS_SUCCESS on success

Parameters:

ssh - pointer to `WOLFSSH` structure

Example:

```
WOLFSSH* ssh;

//set up ssh and do sftp connections
…

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

See Also:

wolfSSH_SFTP_SaveOfst()
wolfSSH_SFTP_GetOfst()

wolfSSH_SFTP_Interrupt()

Synopsis:

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

Description:

Function to set interrupt flag and stop a get/put command.

Return Values:

None

Parameters:

ssh - pointer to WOLFSSH structure

Example:

```
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);
```

See Also:

wolfSSH_SFTP_SaveOfst()
wolfSSH_SFTP_GetOfst()

15.4 Command Functions

wolfSSH_SFTP_Remove()

Synopsis:

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

Description:

Function for sending a “remove” packet across the channel.
The file name passed in as “f” is sent to the peer for removal.

Return Values:

WS_SUCCESS: returns WS_SUCCESS on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
f - file name to be removed

Example:

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

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

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

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_MKDIR()

Synopsis:

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

     int wolfSSH_SFTP_MKDIR(WOLFSSH* ssh, char* dir,
                                          WS_SFTP_FILEATRB*
atr);
```

Description:

Function for sending a “mkdir” packet across the channel. The directory name passed in as “dir” is sent to the peer for creation. Currently the attributes passed in are not used and default attributes is set instead.

Return Values:

WS_SUCCESS: returns WS_SUCCESS on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
dir - NULL terminated directory to be created
atr - attributes to be used with directory creation

Example:

```
WOLFSSH* ssh;

int ret;

char* dir[DIR_SZ];

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

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

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_RMDIR()

Synopsis:

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

int wolfSSH_SFTP_RMDIR(WOLFSSH* ssh, char* dir);
```

Description:

Function for sending a “rmdir” packet across the channel. The directory name passed in as “dir” is sent to the peer for deletion.

Return Values:

WS_SUCCESS: returns WS_SUCCESS on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
dir - NULL terminated directory to be remove

Example:

```
WOLFSSH* ssh;

int ret;

char* dir[DIR_SZ];

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

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

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_Rename()

Synopsis:

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

Description:

Function for sending a “rename” packet across the channel. This tries to have a peer file renamed from “old” to “nw”.

Return Values:

WS_SUCCESS: returns WS_SUCCESS on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
old - Old file name
nw - New file name

Example:

```
    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);
```

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_LS()

Synopsis:

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

WS_SFTPNAME* wolfSSH_SFTP_LS(WOLFSSH* ssh, char* dir);
```

Description:

Function for performing LS operation which gets a list of all files and directories in the current working directory. This is a high level function that performs REALPATH, OPENDIR, READDIR, and CLOSE operations.

Return Values:

On Success, returns a pointer to a list of `WS_SFTPNAME` structures.
NULL on failure.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
dir - directory to list

Example:

```
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);
```

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()
wolfSSH_SFTPNAME_list_free()

wolfSSH_SFTP_Get()

Synopsis:

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

    int wolfSSH_SFTP_Get(WOLFSSH* ssh, char* from, char* to,
byte resume,
    WS_STATUS_CB* statusCb);
```

Description:

Function for performing get operation which gets a file from the peer and places it in a local directory. This is a high level function that performs LSTAT, OPEN, READ, and CLOSE operations. To interrupt the operation call the function wolfSSH_SFTP_Interrupt. (See the API documentation of this function for more information on what it does)

Return Values:

WS_SUCCESS: on success.
All other return values should be considered error cases.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
from - file name to get
to - file name to place result at
resume - flag to try resume of operation. 1 for yes 0 for no
statusCb - callback function to get status

Example:

```
    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
    }
```

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()

wolfSSH_SFTP_Put()

Synopsis:

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

Description:

Function for performing put operation which pushes a file local file to a peers directory.
This is a high level function that performs OPEN, WRITE, and CLOSE operations.
To interrupt the operation call the function `wolfSSH_SFTP_Interrupt`.
(See the API documentation of this function for more information on what it does)

Return Values:

WS_SUCCESS on success.
All other return values should be considered error cases.

Parameters:

ssh - pointer to WOLFSSH structure used for connection
from - file name to push
to - file name to place result at
resume - flag to try resume of operation. 1 for yes 0 for no
statusCb - callback function to get status

Example:

```
    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
    }
```

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()

15.5 SFTP Server Functions

wolfSSH_SFTP_read()

Synopsis:

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

Description:

Main SFTP server function that handles incoming packets. This function tries to read from the I/O buffer and calls internal functions to depending on the SFTP packet type received.

Return Values:

WS_SUCCESS: on success.

Parameters:

ssh - pointer to WOLFSSH structure used for connection

Example:

```
WOLFSSH* ssh;

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

See Also:

wolfSSH_SFTP_accept()
wolfSSH_SFTP_connect()