

SSH libraries : What they can do for you

Aris Adamantiadis <aris@0xbadc0de.be>



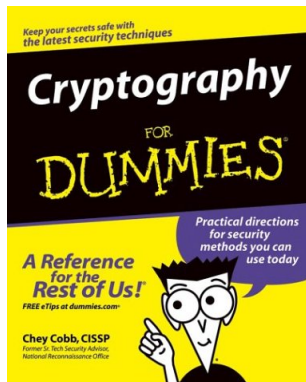
Fosdem 2011

Who am I ?



Some cryptographic transport protocols

- SSL
- TLS
 - Actually, TLS 1.0 = SSL 3.1
- SSH-1
- SSH-2



SSL/TLS

- Secure Socket Layer/Transport Layer Security
- Initially (poorly) developed by Netscape
- Widely used for online applications (https, ftps, imaps, ...)
- Based on X.509 certificates for both servers and clients.
- Many implementations (OpenSSL, GNU/TLS, NSS, YaSSL, MS Windows, ...)



SSH

- Secure SHell.
- Initially developed by Tatu Ylönen, but then lots of development by OpenSSH team.
- Defined in RFCs (RFC4250-4256).
- Many features :
 - Secure transport
 - Authentication
 - Shell/terminal handling
 - File transfer/remote file system (SFTP)



What about security

- Integrity
 - Tampering is detected
 - Availability
 - No shutdown of connection possible
 - Confidentiality
 - Nobody can eavesdrop communication
 - You are sure of the identity of the remote side
- Integrity
 - Strong HMAC detect any change
 - Availability
 - Transport protocols run over TCP
 - No protection against a forged RST packet
 - Confidentiality
 - Strong ciphers
 - Strong key exchange
 - Authentication of key exchange



What about security - Authentication (1)

- Avoid MiM
- Client must ensure authenticity of the server

	TLS	SSH
Trust	X.509 certs	Server key hashes
Key exchange	Diffie-Hellman, RSA	DH, ECDH, ECMQV
Key types	RSA, DSA	RSA, DSA, ECDSA
Verification	Trust chain	Known host file
Authority	PKI (home CA)	Local authority
Authority	3rd party CA	in-DNS hashes + DNSSEC -



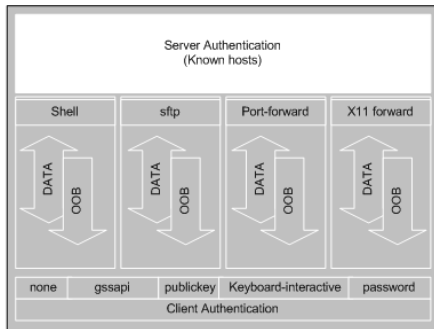
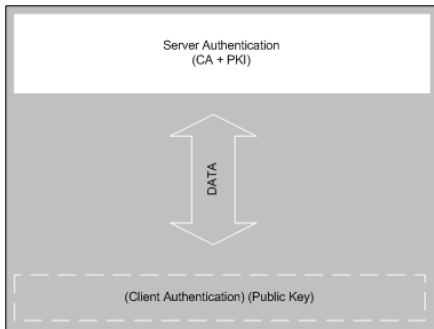
What about security - Authentication (2)

- Avoid MiM
- Client must ensure authenticity of the server
- Servers must authenticate clients

	TLS	SSH
Key pairs	X.509 Self-signed	Client public key
Crypto token	PKCS #11 w/ X.509 CA, self-signed	PKCS #11 w/ X.509 CA, public key
Password	Application	Password
OTP/Challenge	Application	Keyboard-interactive
2-factors auth.	Application	Partial authentication
SSO	Application	gss-api, Kerberos



Little comparison



Which one to choose ?

- SSL/TLS is a Transport protocol
- Symmetric client/server
- How do you authenticate your servers ?
 - Is it acceptable that China can forge valid certificates ?
- Ideal for REST-based/HTTP protocol
- SSH is an Application protocol
- Asymmetrical
- Very simple host authentication model
- Guarantee to find OpenSSH everywhere
- Ideal for multi-channels communication, system protocols
 - What about IMAP over SSH ?



Existing SSH libraries

Name	Language	OS	License
libssh	C	Unix, Windows	LGPL
libssh2	C	Unix, Windows	BSD
Granados	C#	.NET	Apache
Net::SSH	Ruby	Ruby	MIT
SSH.NET	C#	.NET	BSD based ?
JSch	Java	JVM	BSD
sshj	Java	JVM	Apache
ne7ssh	C++	Linux	QPL
paramiko	Python	Python VM	LGPL



Some history

- Started as SSH PoC in 2003
- Server part developed in 2005 with Google SoC
- Andreas joined me in 2008
- Now libssh is around 33K LOC (OpenSSH 5.8p1 is 100K)
- Used by many F/OSS projects, including KDE



Features

- Client-side, Server Side
- SSH2, SSH1 for client
- Authentication using password, keyboard-interactive, publickey (including with SSH Agent)
- Depends either on OpenSSL or GCrypt
- Runs on Windows, Unix, VMS !
- SCP, SFTP, Compression, Forwarding, ...



Documentation

- Critical for a library
- All API carefully documented with Doxygen
- Look by yourself ! <http://api.libssh.org/>
- Tutorial explaining most basic operations
- *examples/* directory, plenty of working code



Documentation

The Tutorial

Introduction

libssh is a C library that enables you to write a program that uses the SSH protocol. With it, you can remotely execute programs, transfer files, or use a secure and transparent tunnel for your remote programs. The SSH protocol is encrypted, ensures data integrity, and provides strong means of authenticating both the server of the client. The library hides a lot of technical details from the SSH protocol, but this does not mean that you should not try to know about and understand these details.

libssh is a Free Software / Open Source project. The libssh library is distributed under LGPL license. The libssh project has nothing to do with "libssh2", which is a completely different and independant project.

libssh can run on top of either `libgcrypt` (<http://directory.fsf.org/project/libgcrypt/>) or `libcrypto` (<http://www.openssl.org/docs/crypto/crypto.html>), two general-purpose cryptographic libraries.

This tutorial concentrates for its main part on the "client" side of libssh. To learn how to accept incoming SSH connexions (how to write a SSH server), you'll have to jump to the end of this document.

This tutorial describes libssh version 0.5.0. This version is the development version and is *not* published yet. However, the examples should work with little changes on versions like 0.4.2 and later.

Table of contents:

[Chapter 1: A typical SSH session](#)

[Chapter 2: A deeper insight on authentication](#)

[Chapter 3: Opening a remote shell](#)

[Chapter 4: Passing a remote command](#)

[Chapter 5: The SFTP subsystem](#)

[Chapter 6: The SCP subsystem](#)

[Chapter 7: Forwarding connections \(tunnel\)](#)

[Chapter 8: Threads with libssh](#)

[To be done](#)



Development model

- Own infrastructure
- git, redmine, mailing list, website, test center
- Around 10 total contributors, 3 or 4 regular committers
- Testcase based development, with nightly builds
- Look by yourself ! *<http://test.libssh.org/>*



Test dashboard

Nightly Expected														
Site	Build Name	Update		Configure		Build			Test				Build Time	
		Files	Min	Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass		Min
ansion.libssh.org	CentOS 5.5-GCC 4.1-x86_64-default	0	0	0	0	0	0	0	0.2	0	0	8	0.1	2011-02-05T02:25:41 CET
utapau.libssh.org	FreeBSD 8.1-GCC 4.2-x86_64-default	0	0	0	0	0	0	0	0.2	0	0	8	0.1	2011-02-05T02:20:31 CET
naboo.libssh.org	OpenSUSE 11.3-GCC 4.5-i686-default	0	0	0	0	0	0	0	0.3	0	0	14	0.1	2011-02-05T02:06:14 CET
naboo.libssh.org	OpenSUSE 11.3-GCC 4.5-i686-release	0	0	0	0	0	0	10	0.4	0	0	14	0.1	2011-02-05T02:06:45 CET
naboo.libssh.org	OpenSUSE 11.3-GCC 4.5-sshv1_only	0	0	0	0	0	0	0	0.2	0	0	8	0.1	2011-02-05T02:13:40 CET
naboo.libssh.org	OpenSUSE 11.3-GCC 4.5-x86_64-client_only	0	0	0	0	0	0	0	0.2	0	0	8	0.1	2011-02-05T02:11:48 CET
	OpenSUSE 11.3-													



Test dashboard

naboo.libssh.org	OpenSUSE_11.3-LLVM_2.8-x86_64-default	0	0	0	0	0	0	0	0.2	0	0	14	0.1	2011-02-05T02:05:47 CET
Totals	14 Builds	0	0	0	0	0	0	20	3.4	0	0	148	1.3	

No Nightly Stable Builds

No Nightly Builds

No Continuous Builds

No Experimental Builds

Coverage

Site	Build Name	Percentage	LOC Tested	LOC Untested	Date
naboo.libssh.org	OpenSUSE_11.3-GCC_4.5-x86_64-default	29.43%	3821	9164	2011-02-05T02:07:21 CET

Dynamic Analysis

Site	Build Name	Checker	Defect Count	Date
naboo.libssh.org	OpenSUSE_11.3-GCC_4.5-x86_64-default	Valgrind	0	2011-02-05T02:07:21 CET



Test dashboard








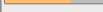

Coverage started on Saturday, February 05 2011

Coverage Summary	
Total Coverage	29.43
Tested lines	3821
Untested lines	9164
Files Covered	44 of 61
Files Satisfactorily Covered	25
Files Unsatisfactorily Covered	36

Coverage Legend
Satisfactory coverage
Unsatisfactory coverage
Dangerously low coverage

 [\[Show coverage over time\]](#)

[Low \(27\)](#) | [Medium \(9\)](#) | [Satisfactory \(25\)](#)

Filename	Status	Percentage	Line not covered	Priority
./src/keyfiles.c	Medium	 47.13%	138/261	None
./src/error.c	Medium	 50.00%	11/22	None
./src/dh.c	Medium	 53.38%	193/414	None
./src/session.c	Medium	 54.95%	100/222	None
./src/match.c	Medium	 59.57%	19/47	None
./src/buffer.c	Medium	 65.03%	64/183	None
./src/packet.c	Medium	 65.28%	75/216	None
./src/client.c	Medium	 66.43%	141/420	None
./src/socket.c	Medium	 67.49%	105/323	None



Some samples - Connect to SSH

```
#include <libssh/libssh.h>
...
ssh_session session = ssh_new();
int r;
ssh_options_set(session, SSH_OPTIONS_HOST, "localhost");
ssh_options_set(session, SSH_OPTIONS_USER, "aris");
r=ssh_connect(session);
if(r==SSH_OK){
    // connected
    ssh_disconnect(session);
}
ssh_free(session);
```



Some samples - Check known host

```
int r = ssh_is_server_known(session);
switch (r) {
    case SSH_SERVER_KNOWN_OK:
        break; /* ok */
    case SSH_SERVER_KNOWN_CHANGED:
    case SSH_SERVER_FOUND_OTHER:
        break; /* not ok */
    case SSH_SERVER_FILE_NOT_FOUND:
    case SSH_SERVER_NOT_KNOWN:
        hexa = ssh_get_hexa(hash, hlen);
        [...] // show hash and ask agreement
        ssh_write_knownhost(session);
}
```



Some samples - Authenticate

```
int rc = ssh_userauth_autopubkey(session, NULL);  
if(rc==SSH_AUTH_SUCCESS)  
    // good !  
if(rc==SSH_AUTH_PARTIAL)  
    // two factor authentication  
rc = ssh_userauth_password(session, NULL, "hunter2");
```

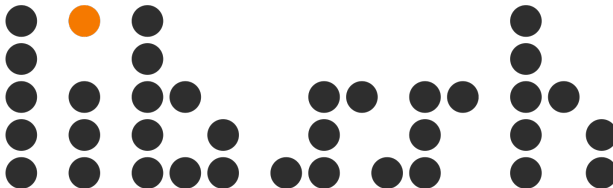


Some samples - Execute a command

```
ssh_channel channel = ssh_channel_new(session);
int r;
char buffer[256];
ssh_channel_open_session(channel);
ssh_channel_request_exec(channel, "ls -l /");
do {
    r=ssh_channel_read(channel, buffer, sizeof(buffer), 0);
    write(1, buffer, r);
} while (r>0);
ssh_channel_close(channel);
ssh_channel_free(channel);
```



Thanks for your attention



Any question ?

