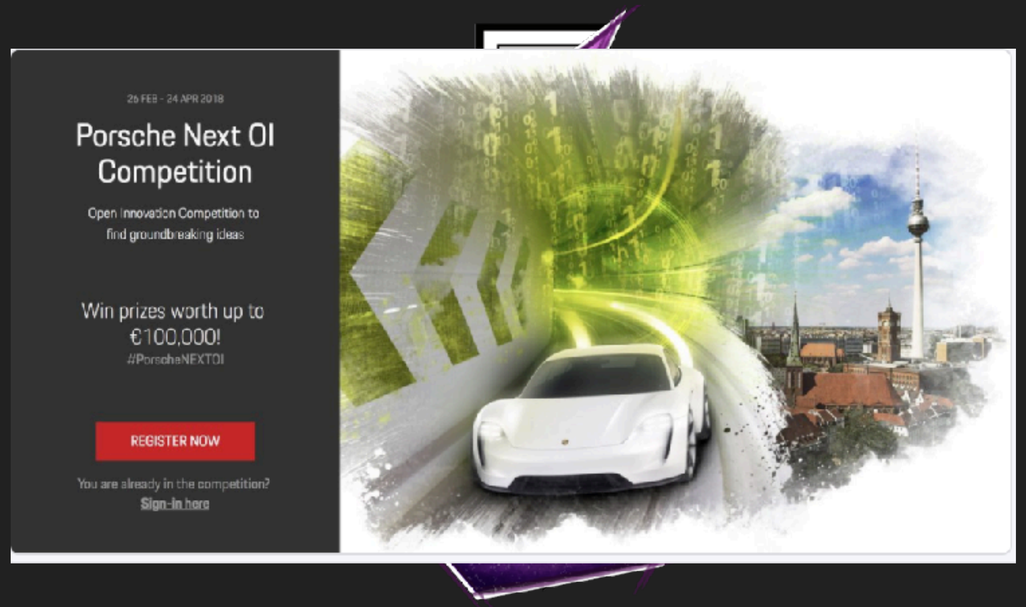
SSH SERVER/CLIENT IN ERLANG

Milad Rastian

twitter/github: @slashmili

High Mobility

Developer Platform for Connected Cars



http://porsche-next-oi-competition.com


heroku


ERLANG


TOP
SECRET


aws

SSH Protocol
SSH Introduction
Cryptography 101
SSH Internal
:ssh Module
:ssh Server
:ssh Client
:ssh Subsystem

Quick Overview

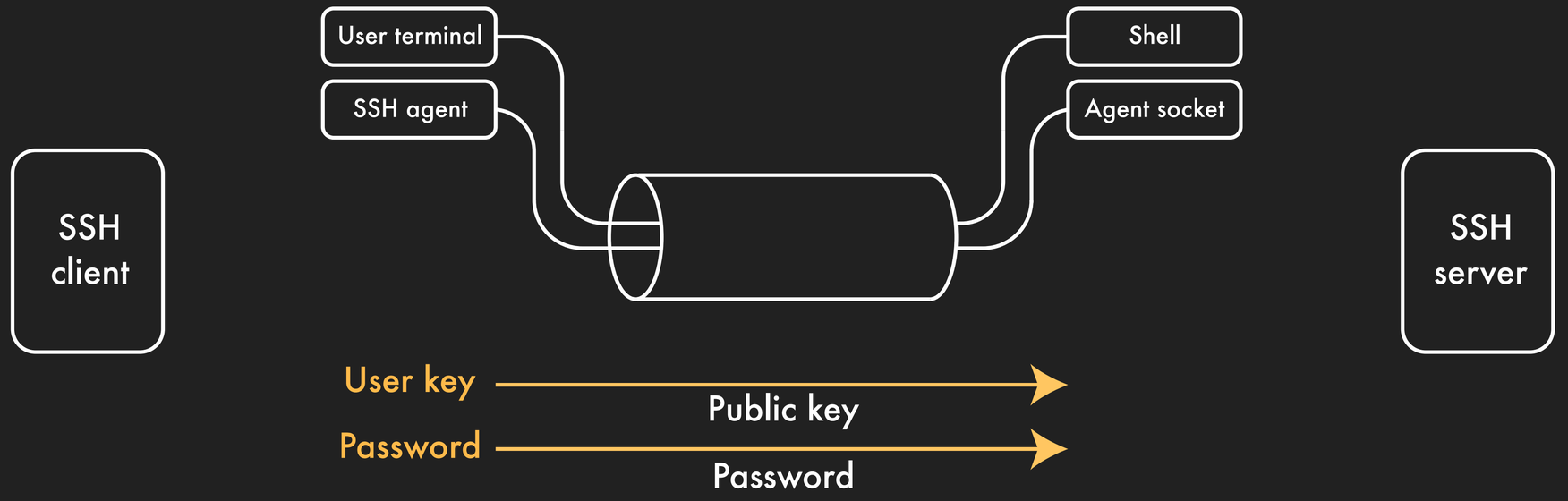
SSH Introduction

Name	First release date	Last release version	Last release date	License	Based on
AbsoluteTelnet	1996	10.15	2016-01-21	Proprietary	
Apache SSHD	2009	1.2.0	2016-03-09	Apache 2.0	
AsyncSSH	2013-09-14	1.11.1	2017-11-15	EPL v1.0	
Bitvise SSH	2001	6.47	2016-01-05	Proprietary	
ConnectBot	2007-11	1.8.6	2015-08-28	Apache 2.0	Trilead SSH-2
cryptlib		3.4.2	2012-12-17	Dual license: Sleepycat or commercial	
Dropbear	2003-01-06	2016.73	2016-03-18	MIT style	
Erlang ssh library	2005-10-25	4.5 (OTP 20.0)	2017-06-21	Apache-2.0	
FlowSsh		5.38	2016-01-05	Proprietary	
Georgia SoftWorks SSH	2004-07	8.07.0002	2016-02-26	Proprietary	
iTerminal		4.0	2016-02-17	Proprietary	libssh2
JSch		0.1.53	2015-06-05	BSD style	
JuiceSSH	2012-12	2.1.2	2015-12-26	Proprietary	JSch
libssh	2003-09-03	0.7.3	2016-02-23	LGPL 2.1	
libssh2	2004-12-08	1.7.0	2016-02-23	BSD style	
lsh	1999-05-23	2.1	2013-06-26	GPL 2	
MindTerm	1998-11-13	4.1.5	2014-04-01	Proprietary	
Mobile SSH		2.17	2016-01-23	Proprietary	OpenSSH
Mocca Telnet (iOS)		3.1	2016-02-19	Proprietary	
Net::SSH		3.0.1	2015-09-19	MIT style	
OpenSSH	1999-12-01	7.4	2016-12-19	BSD	
Paramiko	2003-09-13	2.0.0	2016-04-28	LGPL 2.1	
phpseclib		1.0.2	2016-05-07	MIT style	
Poderosa		4.3.16	2015-08-03	Apache-2.0	
Prompt 2		2.5	2016-02-16	Proprietary	OpenSSH
PuTTY	1998	0.67	2016-03-05	MIT style	

http://ssh-comparison.quendi.de/

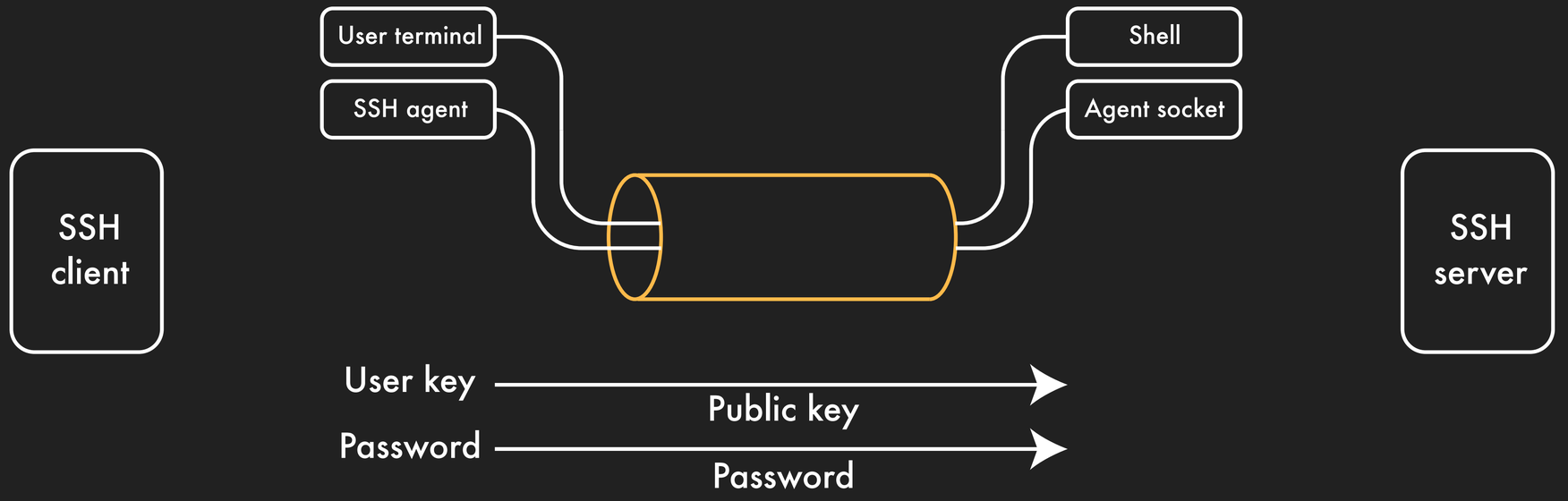
SSH Protocol

user authentication layer



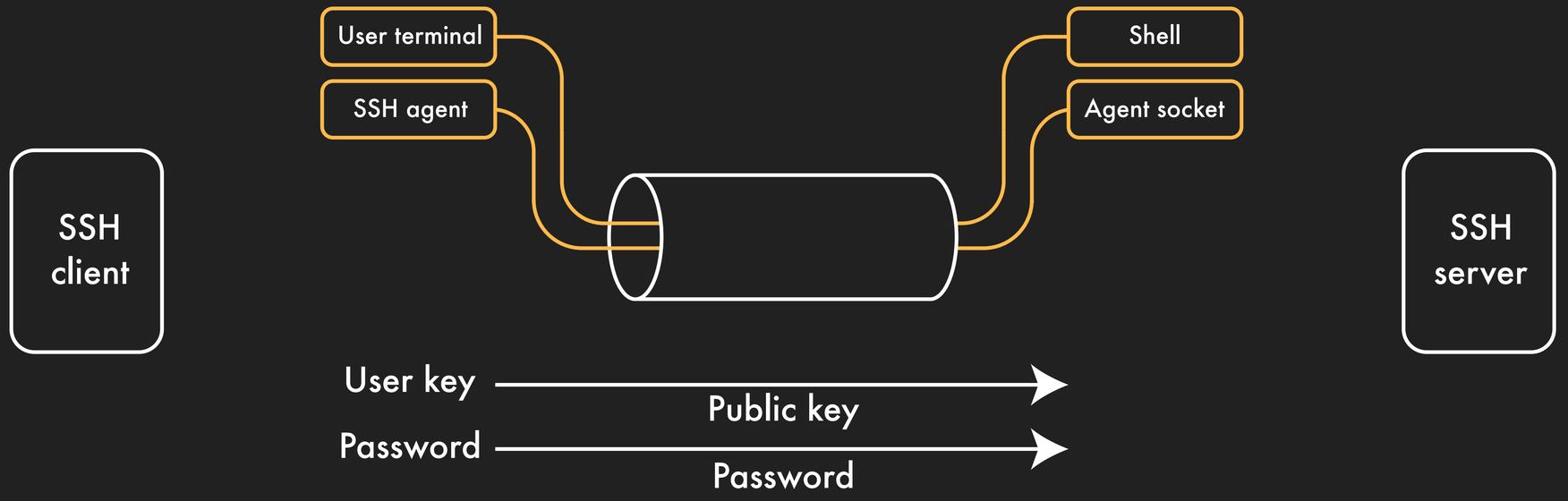
SSH Protocol

transport layer



SSH Protocol

connection layer



SSH user@host ls /etc/

What is SSH again?

History of SSH

History of SSH

Cryptography 101

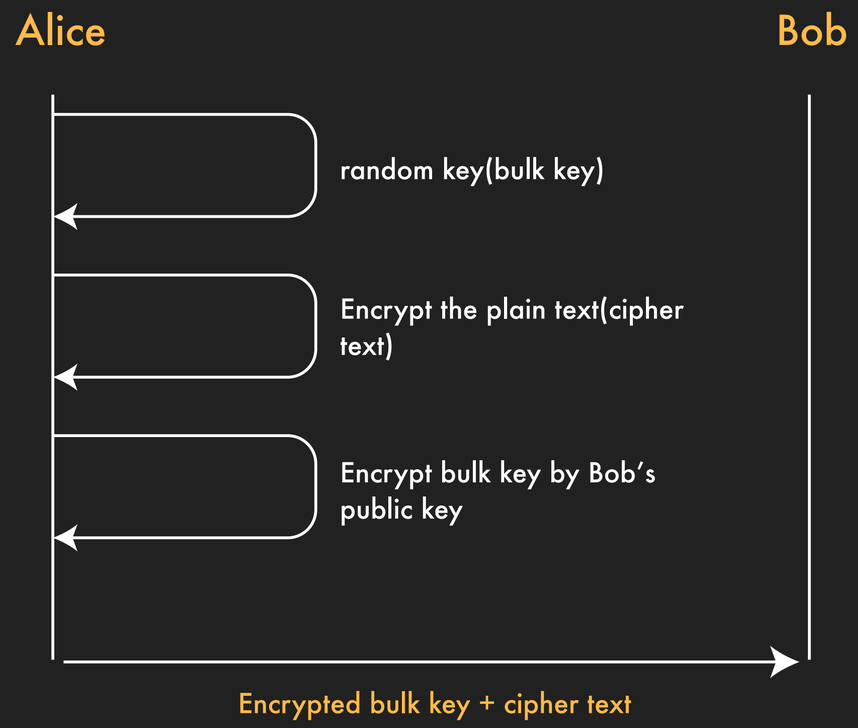
En/Decryption

Encryption algorithms

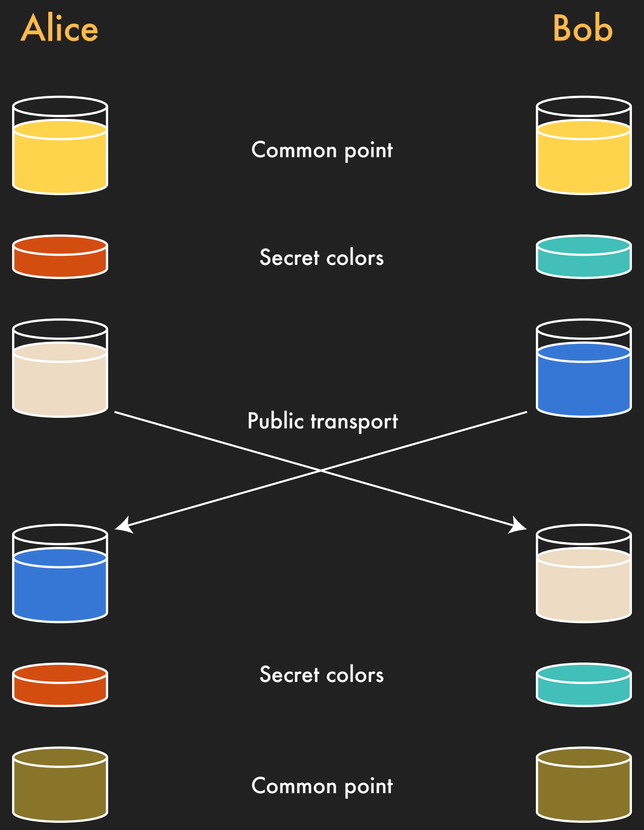
Symmetric

Asymmetric

Encryption using both technique

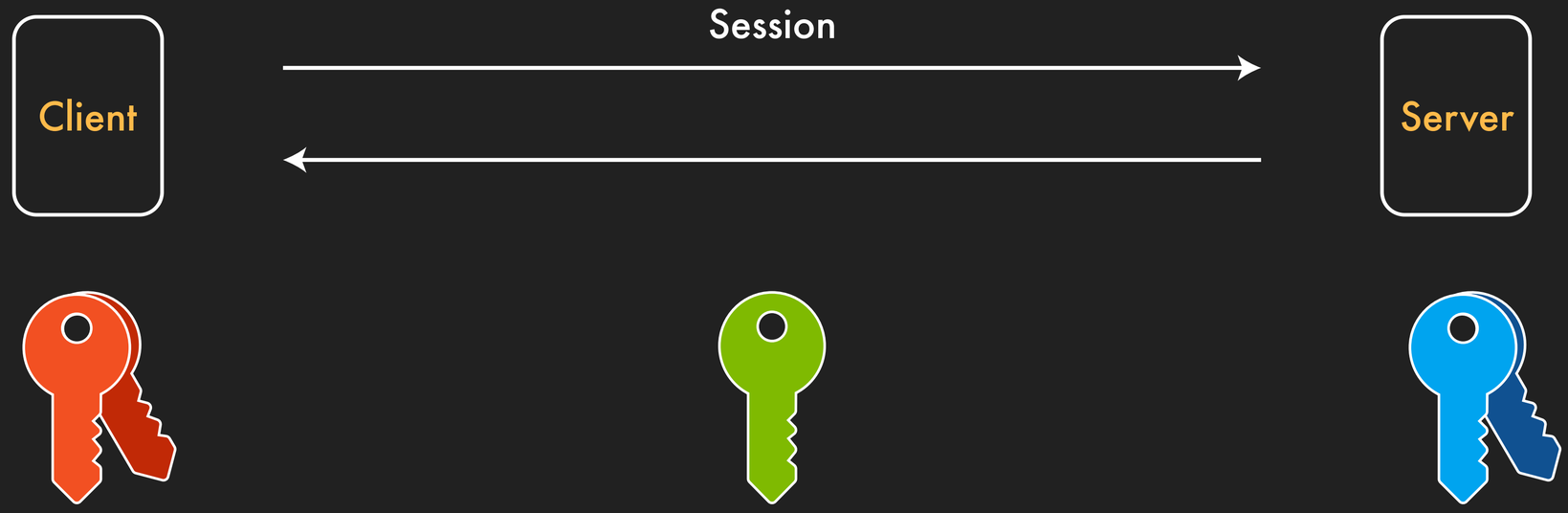


Diffie-Hellman

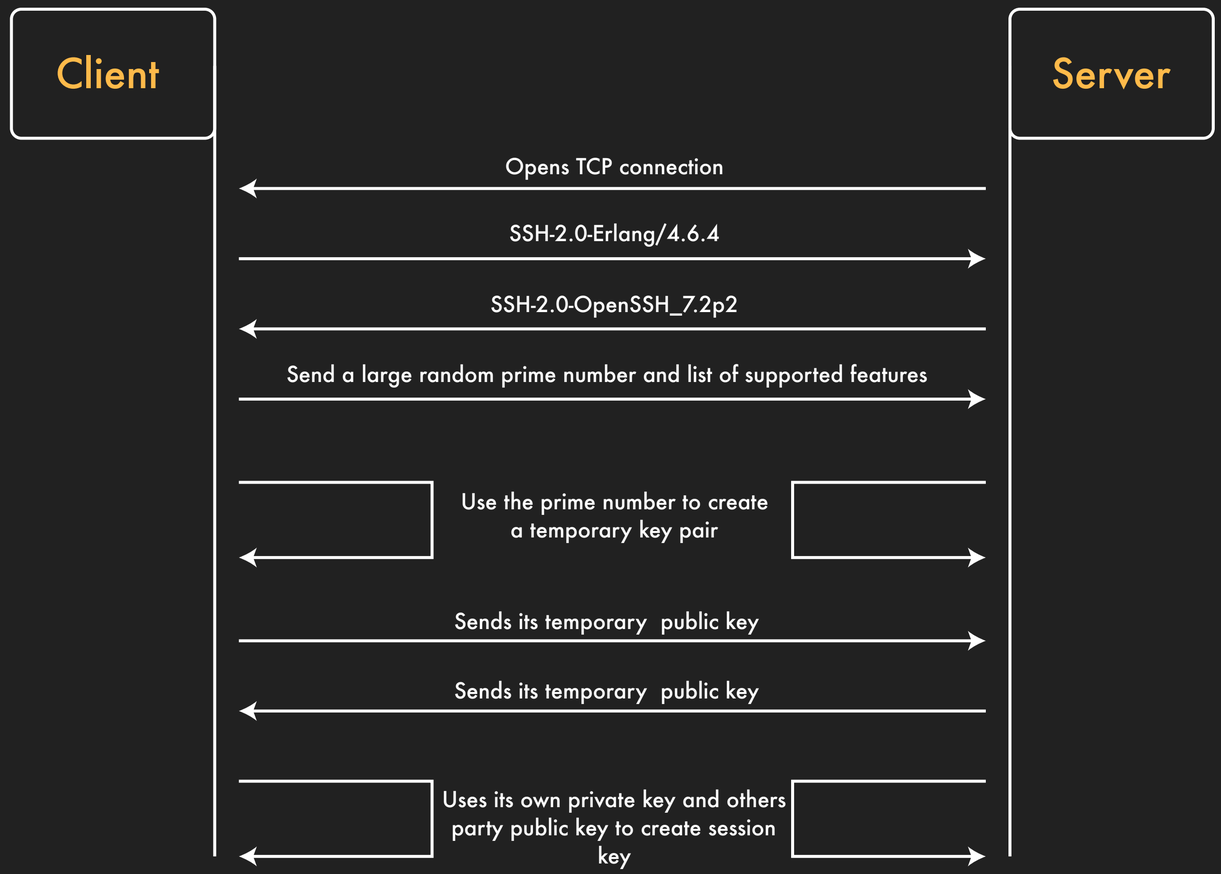


SSH Internal

The Architecture of an SSH System



Establishing the Secure Connection

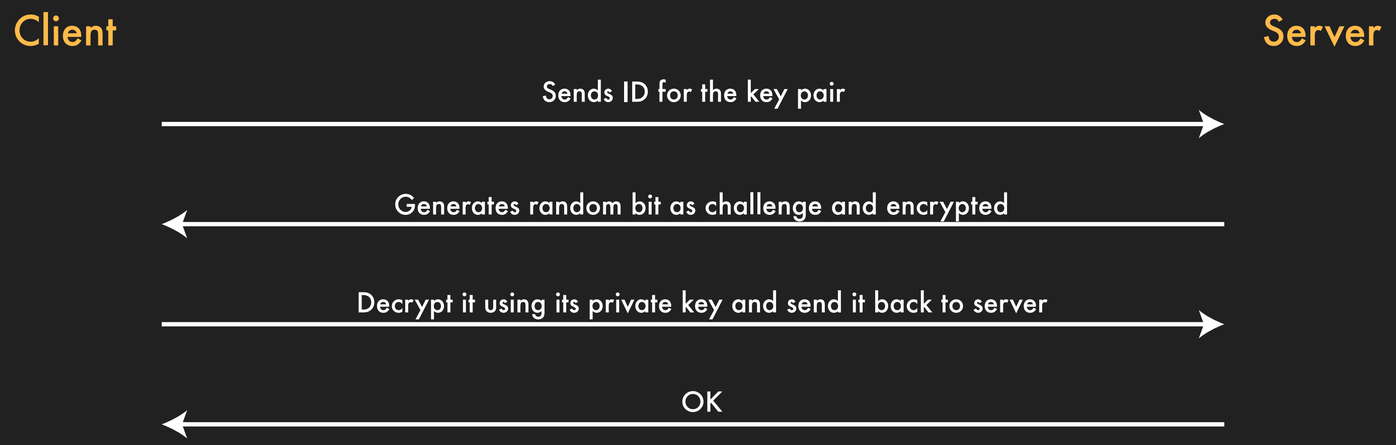


Client Authentication

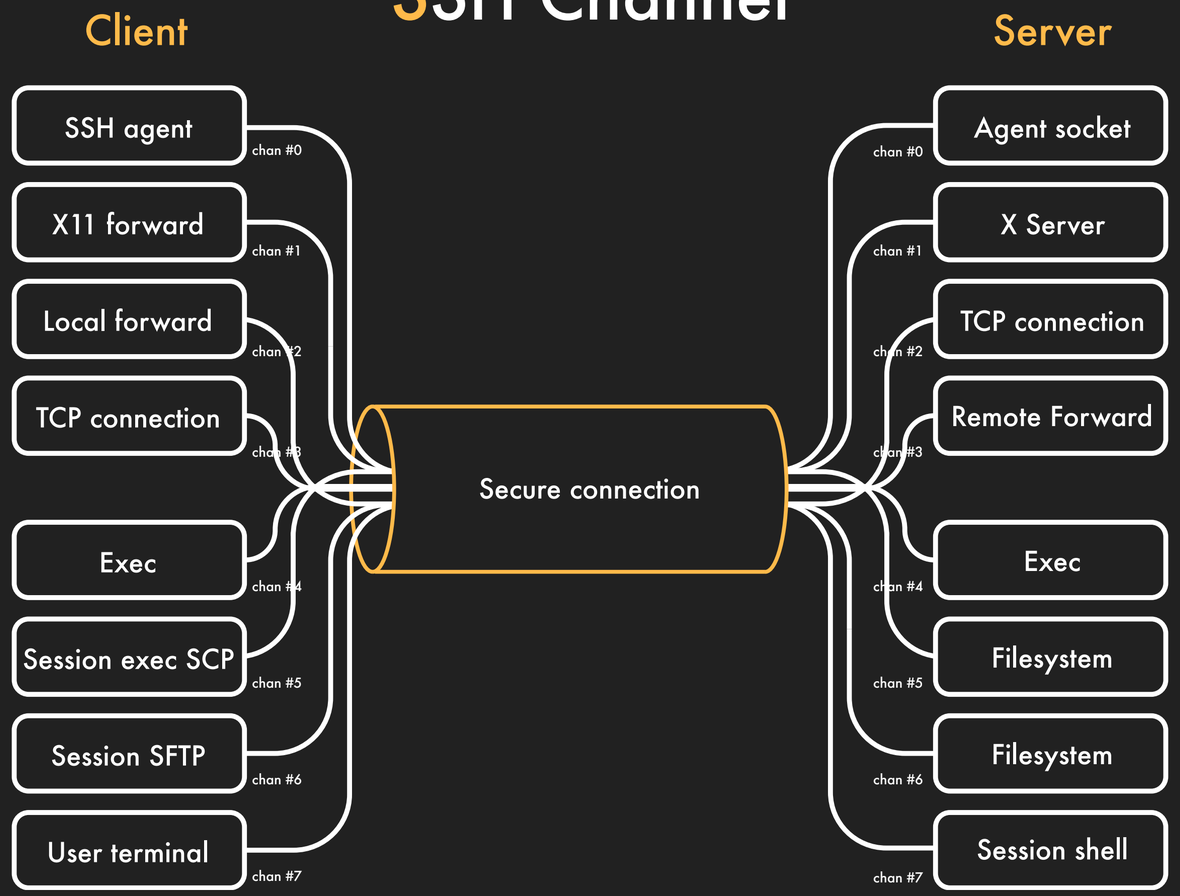
- Password authentication
- Public-Key authentication

Client Authentication

Public-Key authentication



SSH Channel


Client
Server
SSH agent
X11 forward
Local forward
TCP connection
Exec
Session exec SCP
Session SFTP
User terminal
chan #0
chan #1
chan #2
chan #3
chan #4
chan #5
chan #6
chan #7
Secure connection
Agent socket
X Server
TCP connection
Remote Forward
Exec
Filesystem
Filesystem
Session shell
chan #0
chan #1
chan #2
chan #3
chan #4
chan #5
chan #6
chan #7

:ssh Server

Setting up Server Keys

```
/p/t/ssh_talk $ ssh-keygen -N '' -b 256 -t ecdsa -f sys_dir/ssh_host_ecdsa_key
Generating public/private ecdsa key pair.
Your identification has been saved in sys_dir/ssh_host_ecdsa_key.
Your public key has been saved in sys_dir/ssh_host_ecdsa_key.pub.
The key fingerprint is:
SHA256:0JjV9ed0CNWFs9H+iizZyZUINHPyliW3YegjGjxZRdE milad@MBP-9919.local
The key's randomart image is:
+---[ECDSA 256]---+
|         .. +*=.+o|
|       =  B +oE.o|
|      +..+ B *oB+|
|       •= o * o+o|
|        S+ + o .o|
|        .    . o .|
|           = + .  |
|          o * .   |
|           .      |
+----[SHA256]-----+
/p/t/ssh_talk $ ssh-keygen -N '' -b 1024 -t rsa -f sys_dir/ssh_host_rsa_key
/p/t/ssh_talk $ ssh-keygen -N '' -b 1024 -t dsa -f sys_dir/ssh_host_dsa_key
```

http://bit.ly/why-choose-ecdsa

Application List

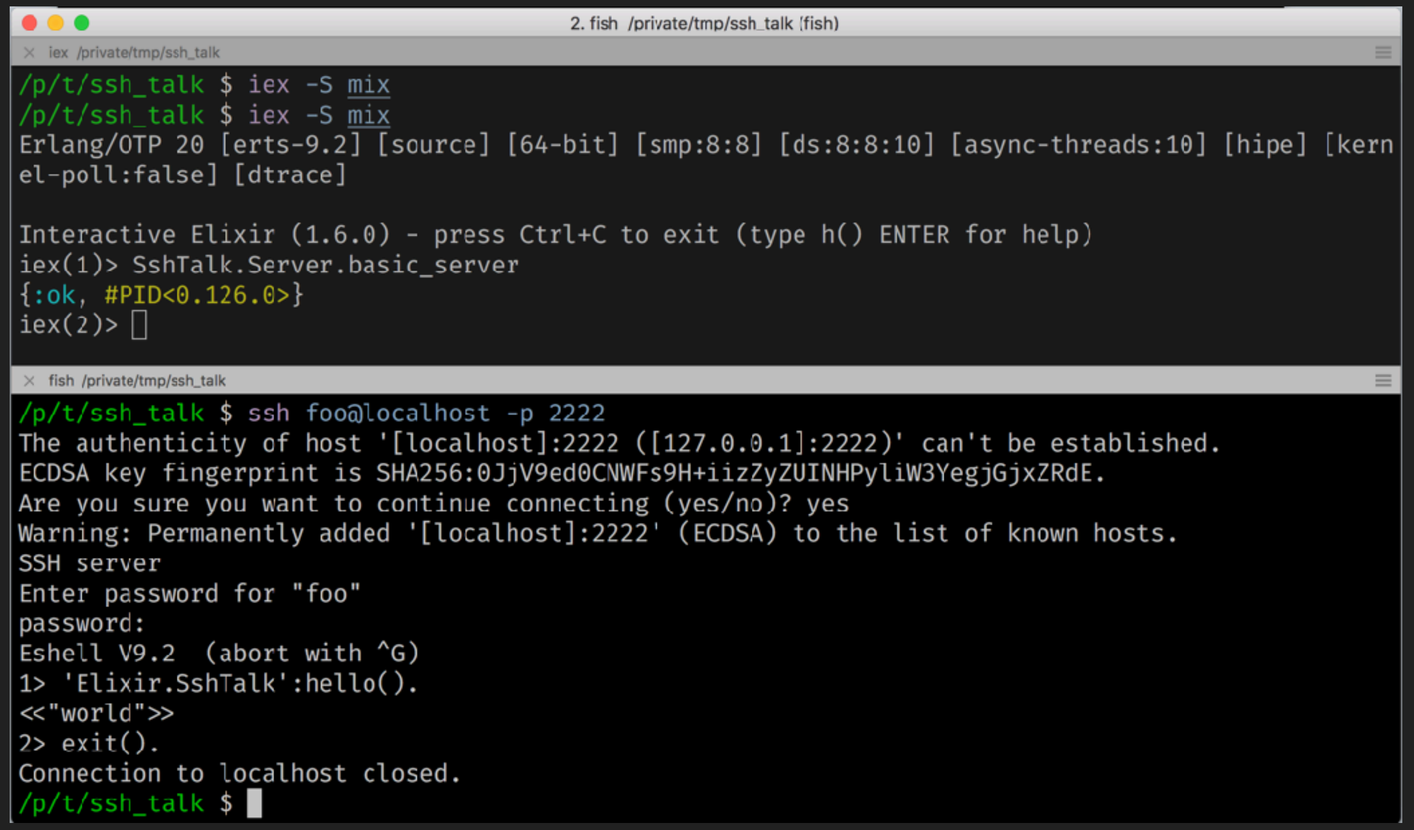
```
def application do
  [
    extra_applications: [:logger, :ssh]
  ]
end
```

NORMAL mix.exs

Basic SSH Server

```elixir
defmodule SshTalk.Server do
  @sys_dir String.to_charlist("#{Path.expand(".")}/sys_dir")

  def basic_server do
    :ssh.daemon(
      2222,
      system_dir: @sys_dir,
      user_passwords: [{'foo', 'bar'}]
    )
  end
end
```

Basic SSH Server



Basic SSH Server with Public Key

```elixir
@usr_dir String.to_charlist("#{Path.expand(".")}/usr_dir")

def basic_server_with_public_key do
  :ssh.daemon(
    2222,
    system_dir: @sys_dir,
    user_dir: @usr_dir,
    auth_methods: 'publickey'
  )
end
```

Basic SSH Server with Public Key

```
/p/t/ssh_talk $ ssh foo@localhost -p 2222
foo@localhost: Permission denied (publickey).
/p/t/ssh_talk $ cat ~/.ssh/id_ecdsa.pub > usr_dir/authorized_keys
/p/t/ssh_talk $ chmod 600 usr_dir/authorized_keys
/p/t/ssh_talk $ ssh foo@localhost -p 2222
Eshell V9.2  (abort with ^G)
1> 'Elixir.SshTalk':hello().
<<"world">>
2> exit().
Connection to localhost closed.
/p/t/ssh_talk $
```

SSH Server with callbacks

```elixir
def server_with_logs do
  :ssh.daemon(
    2222,
    system_dir: @sys_dir,
    user_dir: @usr_dir,
    auth_methods: 'publickey',
    disconnectfun: &log_it/1,
    connectfun: &log_it/3,
    failfun: &log_it/3
  )
end

def log_it(a1, a2 \\ nil, a3 \\ nil) do
  require Logger
  Logger.debug("#{inspect(a1)}, #{inspect(a2)}, #{inspect(a3)}")
end
```

SSH Server with callbacks

```
iex(1)> SshTalk.Server.server_with_logs
{:ok, #PID<0.126.0>}
iex(2)>
14:46:21.215 [debug] 'foo', {{127, 0, 0, 1}, 53234}, 'publickey'

14:46:31.142 [debug] 'foo', {{127, 0, 0, 1}, 53235}, 'publickey'

14:46:32.726 [debug] 'disconnected by user', nil, nil

14:46:39.321 [debug] 'disconnected by user', nil, nil

14:47:37.272 [debug] 'foo', {{127, 0, 0, 1}, 53259}, 'publickey'

14:47:59.626 [debug] 'Connection closed', nil, nil
```

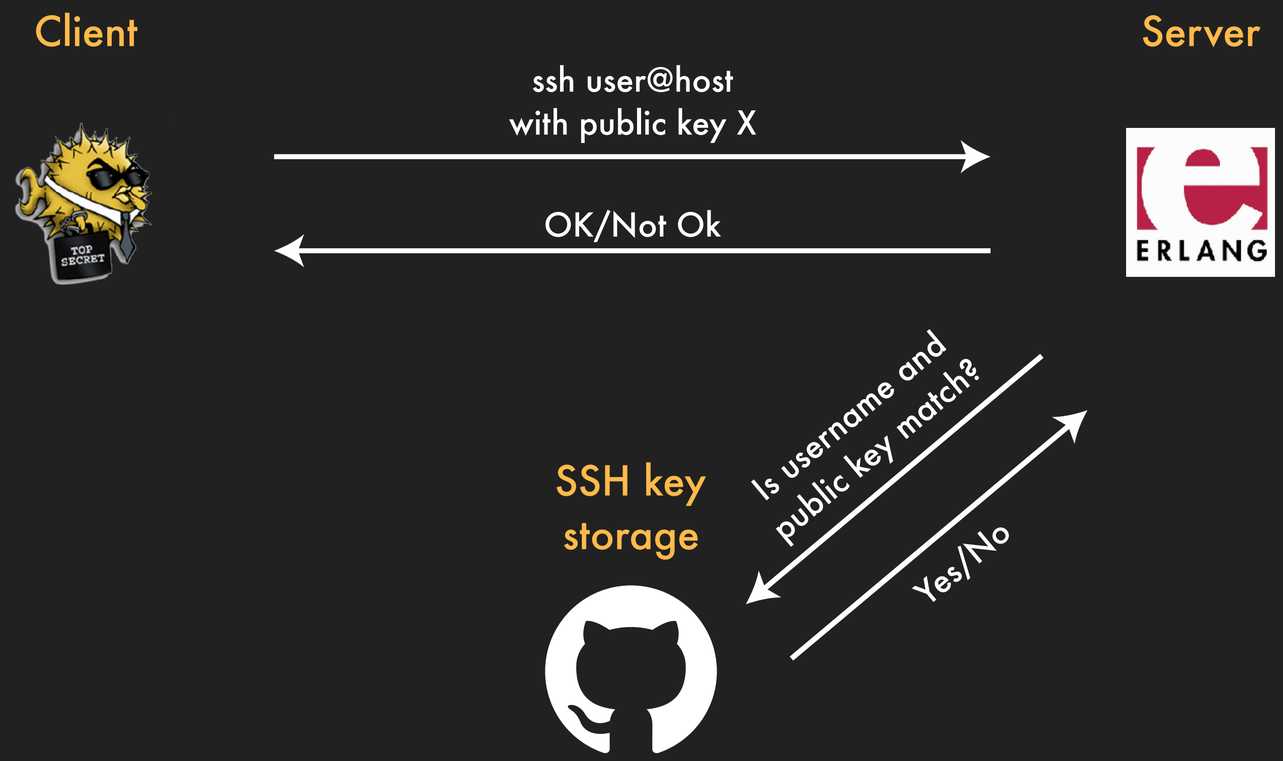
SSH Server with Elixir Shell

```elixir
def server_with_elixir_cli do
  :ssh.daemon(
    2222,
    system_dir: @sys_dir,
    user_dir: @usr_dir,
    auth_methods: 'publickey',
    disconnectfun: &log_it/1,
    connectfun: &log_it/3,
    failfun: &log_it/3,
    shell: &shell/2
  )
end

def shell(username, peer) do
  require Logger
  Logger.debug("shell #{inspect(username)}, #{inspect(peer)}")
  IEx.start([])
end
```

SSH Server with Elixir Shell

```
/p/t/ssh_talk $ ssh foo@localhost -p 2222
Interactive Elixir (1.6.0) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> SshTalk.
MixProject    Server        hello/0
iex(1)> SshTalk.hello
"world"
iex(2)>
```

SSH Server with Custom Public Key backend



SSH Server with Custom Public Key backend

```elixir
defmodule SshTalk.GitHubKeyAuthentication do
  @behaviour :ssh_server_key_api

  require Logger

  def host_key(algorithm, daemon_options) do
    :ssh_file.host_key(algorithm, daemon_options)
  end

  def is_auth_key({:RSAPublicKey, _, _} = key, username, daemon_options) do
    is_auth_key({key, []}, username, daemon_options)
  end

  def is_auth_key(key, username, _daemon_options) do
    key_str =
      [key]
      |> :public_key.ssh_encode(:auth_keys)
      |> String.trim()

    key_str in fetch_github_user_pub_keys(username)
  end

  def fetch_github_user_pub_keys(username) do
    username = to_string(username)

    with {:ok, response} <- HTTPoison.get("https://api.github.com/users/#{username}/keys"),
         {:ok, body} <- Poison.decode(response.body) do
      Enum.map(body, &Map.get(&1, "key"))
    else
      reason ->
        Logger.error("failed to fetch user's keys, error: #{inspect(reason)}")
        []
    end
  end
end
```

SSH Server with Custom Public Key backend

```elixir
def server_with_users_github_public_key do
  :ssh.daemon(
    2222,
    system_dir: @sys_dir,
    user_dir: @usr_dir,
    auth_methods: 'publickey',
    disconnectfun: &log_it/1,
    connectfun: &log_it/3,
    failfun: &log_it/3,
    shell: &shell/2,
    key_cb: SshTalk.GitHubKeyAuthentication
  )
end
```

```
/p/t/ssh_talk $ ssh slashmili@localhost -p 2222
Interactive Elixir (1.6.0) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> Connection to localhost closed.
/p/t/ssh_talk $ ssh josevalim@localhost -p 2222
josevalim@localhost: Permission denied (publickey).
```

SSH <github-user>@ssh-talk.xyz

Simple Git Server

https://github.com/slashmili/gixir-server

:ssh Client

SSH Client using password

```elixir
def simple_client(host, port \\ 22) do
  :ssh.connect(
    String.to_charlist(host),
    port,
    disconnectfun: &log_it/1
  )
end
```

```
iex(1)> SshTalk.Client.simple_client("ssh-talk.xyz")
ssh password:

16:19:14.622 [debug] 'Unable to connect using the available authentication methods', nil, nil
{:error, 'Unable to connect using the available authentication methods'}
iex(2)>
```

SSH Client using Public Key

```elixir
def client_with_public_key(host, port \\ 22) do
  :ssh.connect(
    String.to_charlist(host),
    port,
    disconnectfun: &log_it/1,
    unexpectedfun: &log_it/2,
    user_dir: String.to_charlist(Path.join([System.get_env("HOME"), ".ssh"])),
    auth_methods: 'publickey',
    user: 'root'
```

```
iex(7)> {:ok, conn_ref} = SshTalk.Client.client_with_public_key("ssh-talk.xyz")
{:ok, #PID<0.200.0>}
iex(8)> {:ok, chan} = :ssh_connection.session_channel(conn_ref, :infinity)
{:ok, 0}
iex(9)> :ssh_connection.exec(conn_ref, chan, 'uptime', :infinity)
:success
iex(10)> flush
{:ssh_cm, #PID<0.200.0>,
 {:data, 0, 0,
  " 14:35:03 up 94 days,  4:10,  1 user,  load average: 0.00, 0.23, 0.62\n"}}
{:ssh_cm, #PID<0.200.0>, {:eof, 0}}
{:ssh_cm, #PID<0.200.0>, {:exit_status, 0, 0}}
{:ssh_cm, #PID<0.200.0>, {:closed, 0}}
:ok
iex(11)>
```

:ssh Module

:ssh Subsystem

:ssh Subsystem

SSH Subsystem

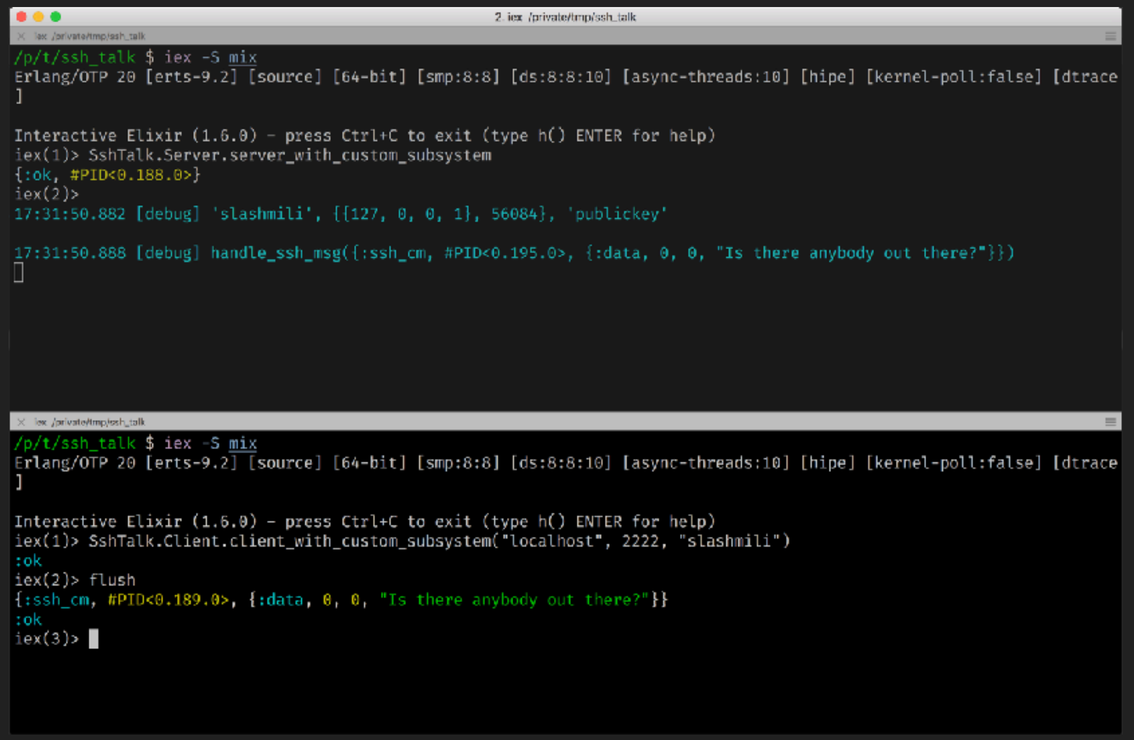
```elixir
defmodule SshTalk.EchoSubsystem do
  @behaviour :ssh_daemon_channel
  require Logger

  def init(opt) do
    {:ok, opt}
  end

  def handle_msg({:ssh_channel_up, _channel_id, _connection_manager}, state) do
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:data, channel_id, 0, data}}, state) do
    Logger.debug("handle_ssh_msg(#{inspect({:ssh_cm, cm, {:data, channel_id, 0, data}})})")
    :ssh_connection.send(cm, channel_id, data)
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:data, channel_id, 1, data}}, state) do
    Logger.debug("handle_ssh_msg(#{inspect({:ssh_cm, cm, {:data, channel_id, 1, data}})})")
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:eof, channel_id}}, state) do
    Logger.debug("handle_ssh_msg(#{inspect({:ssh_cm, cm, {:eof, channel_id}})})")
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:signal, channel_id}}, state) do
    Logger.debug("handle_ssh_msg(#{inspect({:ssh_cm, cm, {:signal, channel_id}})})")
    # Ignore signals according to RFC 4254 section 6.9.
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:exit_signal, channel_id, error}}, state) do
    Logger.debug("handle_ssh_msg(#{inspect({:ssh_cm, cm, {:exit_signal, channel_id, error}})})")
    {:stop, channel_id, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:exit_status, channel_id, status}}, state) do
    Logger.debug("handle_ssh_msg(#{inspect({:ssh_cm, cm, {:exit_status, channel_id, status}})})")
    {:stop, channel_id, state}
  end
```

SSH Server/Client with subsystem

```elixir
def server_with_custom_subsystem do
  :ssh.daemon(
    2222,
    system_dir: @sys_dir,
    user_dir: @usr_dir,
    auth_methods: 'publickey',
    disconnectfun: &log_it/1,
    connectfun: &log_it/3,
    failfun: &log_it/3,
    shell: &shell/2,
    key_cb: SshTalk.GitHubKeyAuthentication,
    subsystems: [{'echo', {SshTalk.EchoSubsystem, []}}]
  )
end
```

```elixir
def client_with_custom_subsystem(hostname, port, username) do
  {:ok, conn_ref} = client_with_public_key(hostname, port, username)
  {:ok, chan} = :ssh_connection.session_channel(conn_ref, :infinity)
  :success = :ssh_connection.subsystem(conn_ref, chan, 'echo', :infinity)
  :ssh_connection.send(conn_ref, chan, "Is there anybody out there?", :infinity)
end
```

SSH Server/Client with subsystem



Reference / Further reading

- SSH, The Secure Shell: The Definitive Guide
- http://erlang.org/doc/man/ssh.html
- https://github.com/jbenden/esshd
- https://github.com/bitcrowd/sshkit.ex
- https://github.com/drowzy/ssh_tunnel

THANK YOU

Milad Rastian
twitter/github: @slashmili