The Secure Shell (SSH) Protocol

Mario Čagalj

University of Split, FESB

Introduction

What is SSH?

SSH is a protocol for secure remote login and other secure network services over an insecure network (RFC 4251).

Developed by Tatu Ylönen (Helsinki University of Finland), later on commercialized by SSH Communications Security Corp., Finland.

Two distributions are available:

- commercial version
- freeware (`www.openssh.com`)

SSH2 is specified in a set of Internet drafts (RFC 4250 - 4256).

The Secure Shell (SSH) Protocol Architecture

The SSH protocol consists of three major components:

- SSH Transport Layer Protocol - provides server authentication, confidentiality, and integrity with perfect forward secrecy
- SSH User Authentication Protocol - authenticates the client to the server
- SSH Connection Protocol - multiplexes the encrypted tunnel into several logical channels (enables secure shell session, TCP port forwarding/tunneling, etc.)

The Secure Shell (SSH) Protocol Architecture

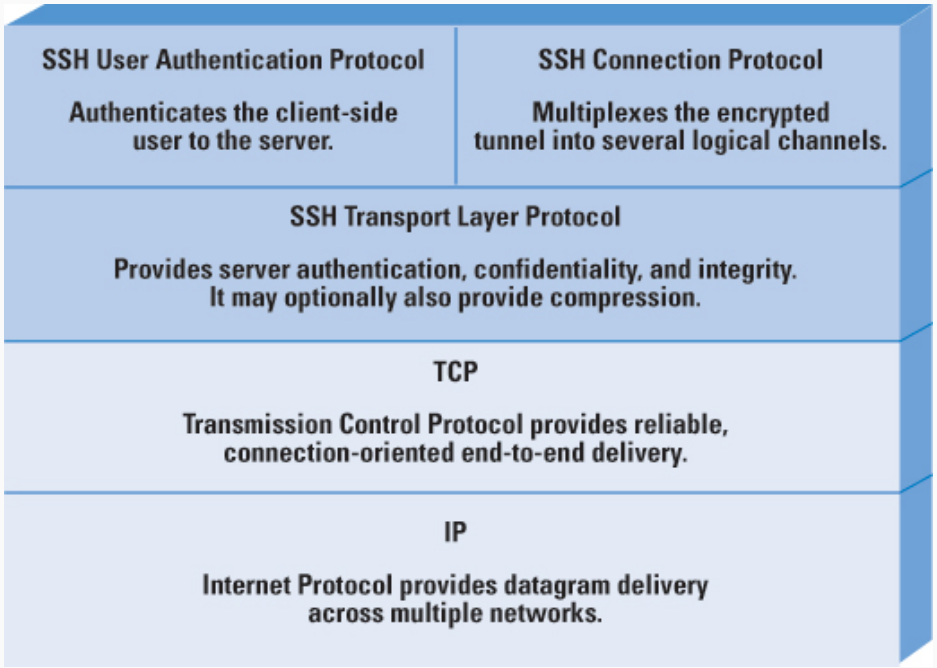

Figure 1: SSH Protocol Stack (source: cisco.com).

SSH Transport Layer Protocol

SSH Transport Layer Protocol - Overview

SSH TLP typically runs on TCP/IP; when used over TCP the server normally listens for connections on port 22. Provides:

- encryption of user data
- server authentication (based on asymmetric host key/s)
- integrity protection (origin and data)
- compression of data prior to encryption (optionally)

Key exchange method, public key algorithm, symmetric encryption algorithm, message authentication algorithm, and hash algorithm are all negotiated (source: RFC 4253).

SSH Transport Layer Protocol (TLP) - Packet Exchanges

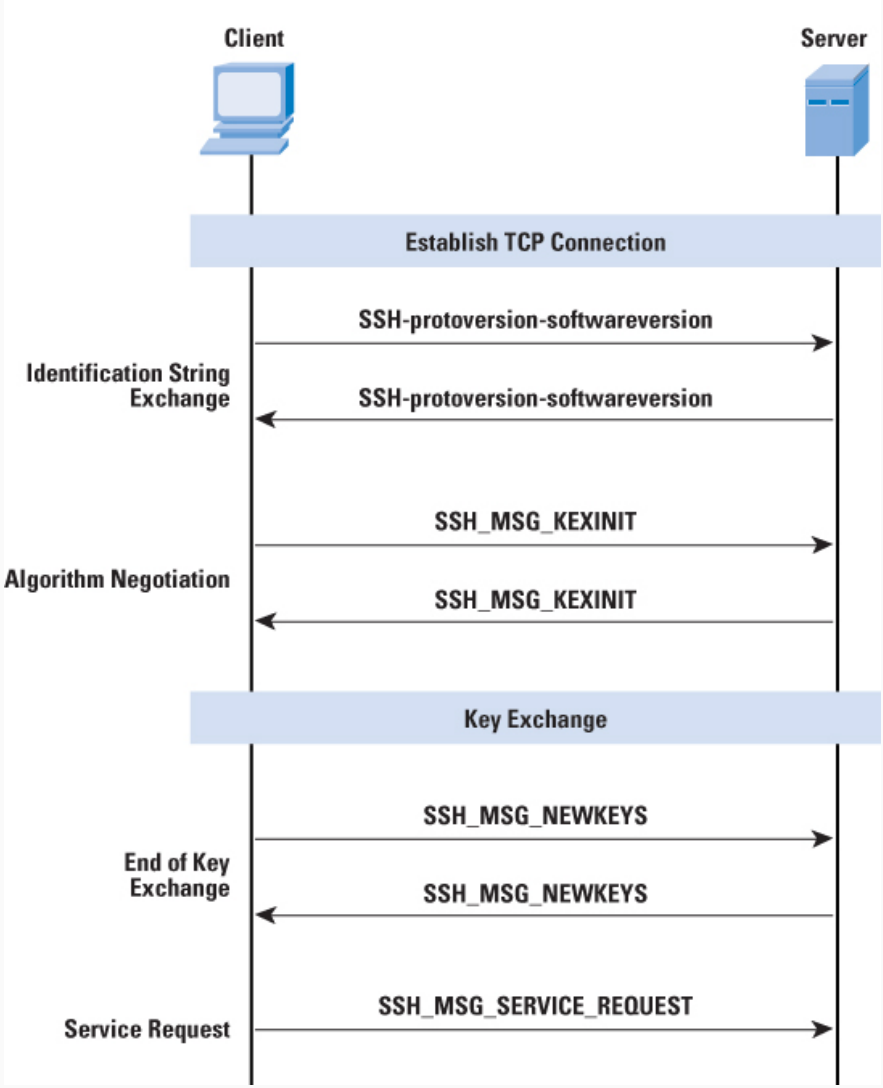

Figure 2: SSH TLP Packet Exchanges (source: cisco.com).

SSH TLP Binary Packet Protocol

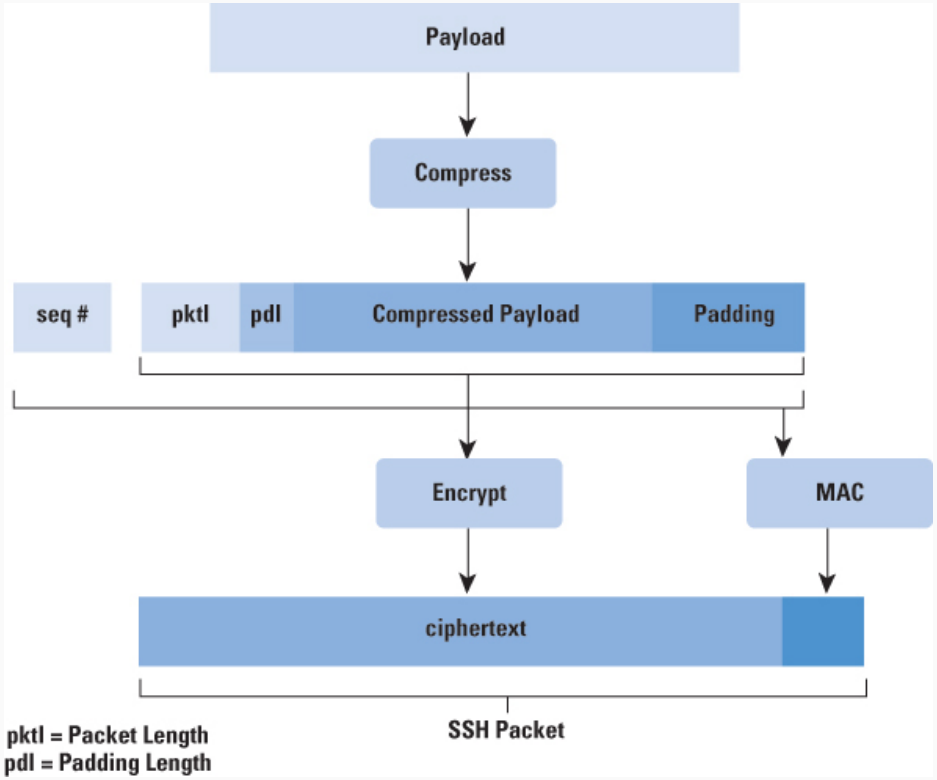

Figure 3: SSH TLP Packet Formation (source: cisco.com).

Implements encrypt and MAC/authenticate paradigm; attacks exists when used with CBC mode (prefer CTR mode).

SSH TLP Binary Packet Protocol

Encryption and authentication algorithms, encryption keys, IVs are all negotiated during the key exchange. Algorithms and modes (AES, 3DES, CBC, CTR, etc.) can be different in each direction.

Before encrypting, the packet is random padded:

```
encoded_packet = (packet_length || padding_length || payload ||
                  random_padding)
```

MAC (HMAC) is computed over the encoded plaintext packet and an implicit 32-bit sequence number (not transmitted); prevents replay attacks. MAC is not encrypted.

```
mac = MAC(key, sequence_number || encoded_packet)
```

SSH TLP Key Exchange

The key exchange method specifies how one-time session keys are generated for encryption and for authentication, and how the server authentication is done (source: RFC 4253).

The SSH specification allows for alternative methods of key exchange, but it specifies only two versions of DiffieHellman key exchange.

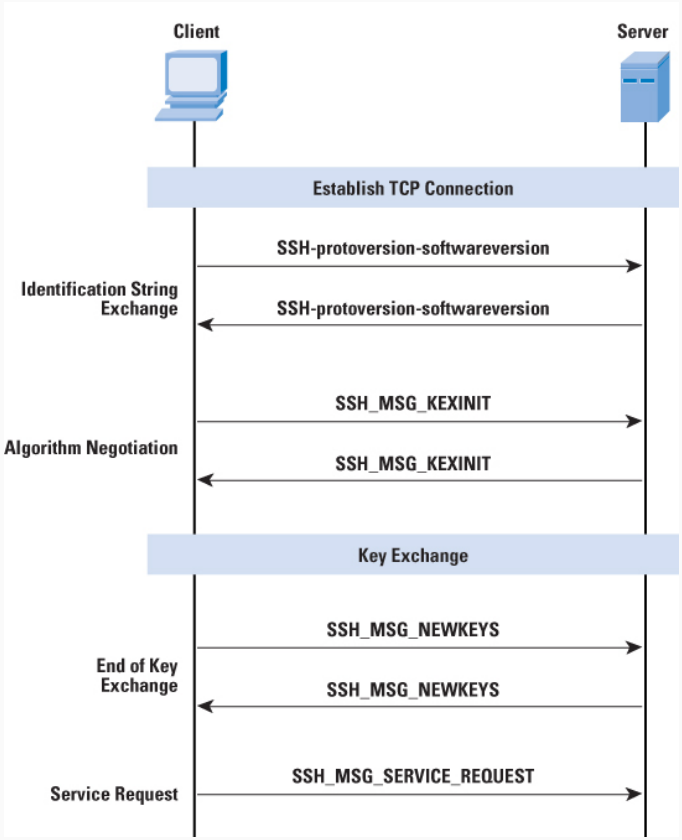

Figure 4: SSH TLP Packet Exchanges (source: cisco.com).

SSH TLP Diffie-Hellman Key Exchange (source: RFC 4253)

V_S and V_C are S's and C's id strings; K_S is S's public host key; I_S and I_S are S's and C's `SSH_MSG_KEXINIT` msgs exchanged before this part begins; these messages include a 128-bit random cookie to ensure session freshness.

1. C picks random x, computes $e = g^x \mod p$. C sends e to S.

2. S picks random y, computes $f = g^y \mod p$. S receives e, computes:
   - $K = e^y \mod p$
   - $H = hash(V_C \,||\, V_S \,||\, I_C \,||\, I_S \,||\, K_S \,||\, e \,||\, f \,||\, K)$
   - signature s on H with its private host key

   S sends $(K_S \,||\, f \,||\, s)$ to C.

3. C verifies that K_S really is the host key for S (e.g., using certificates or a local database, or accepts the key without verification). C computes:
   - $K = f^x \mod p$
   - $H = hash(V_C \,||\, V_S \,||\, I_C \,||\, I_S \,||\, K_S \,||\, e \,||\, f \,||\, K)$

   C verifies the signature s on H.

SSH Session Key Derivation

As a result of the key exchange steps:

- C and S now share a master key K
- S has been authenticated to C (S has signed C's random DH public key)
- The hash value H servers as a `session_id` for this connection
- Subsequent communication protected using the derived session keys

Session init vectors, encryption and authentication keys:

- $IV_{CS} = hash(K \mid\mid H \mid\mid \text{“}A\text{”} \mid\mid session\_id)$
- $IV_{SC} = hash(K \mid\mid H \mid\mid \text{“}B\text{”} \mid\mid session\_id)$
- $EK_{CS} = hash(K \mid\mid H \mid\mid \text{“}C\text{”} \mid\mid session\_id)$
- $EK_{SC} = hash(K \mid\mid H \mid\mid \text{“}D\text{”} \mid\mid session\_id)$
- $AK_{CS} = hash(K \mid\mid H \mid\mid \text{“}E\text{”} \mid\mid session\_id)$
- $AK_{SS} = hash(K \mid\mid H \mid\mid \text{“}F\text{”} \mid\mid session\_id)$

Key data is taken from the beginning of the hash output.

SSH User Authentication Protocol - Overview

After the key exchange, the client requests a service by sending `SSH_MSG_SERVICE_REQUEST` packet to request either the User Authentication or the Connection Protocol.

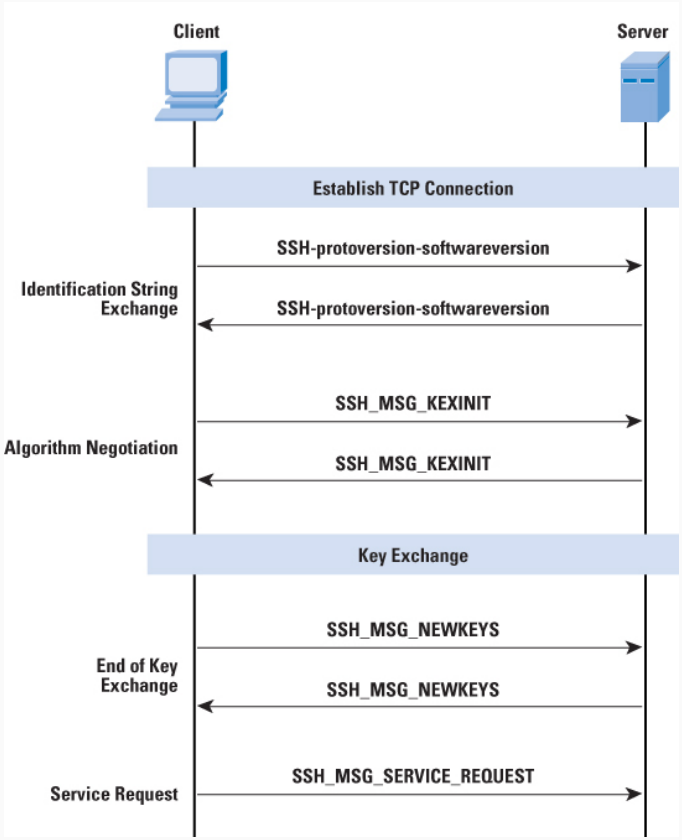

Figure 5: SSH TLP Packet Exchanges (source: cisco.com).

SSH User Authentication Protocol

SSH User Authentication Protocol - Overview

The SSH Authentication Protocol runs over the SSH Transport Layer Protocol; it assumes that the underlying protocol provides integrity and confidentiality protection. The protocol has access to the `session_id` (source: RFC 4252).

Supports three user authentication methods:

- Public key (required)
- Password (optional)
- Host-based (optional)

The server controls which authentication methods can be used.

SSH User Authentication Protocol - Authentication Methods

Public key

The client sends to the server a signed message comprising the `session_id`, the user name, the user's public key, and some other info. The signature is generated using the user's private key.

Upon message reception, the server checks whether the supplied public key is acceptable for authentication, and if so, check whether the signature is correct.

Password

The client essentially sends the user name and the password in clear; they are however protected by the SSH Transport Security Protocol.

Host-based

Analogous to the above public key method but with per-host (not per-user) public key; authentication is based on the host that the user is coming from and the user name on the remote host.

SSH Connection Protocol

SSH Connection Protocol - Overview

Provides:

- interactive login sessions
- remote execution of commands
- forwarded TCP/IP connections (port forwarding)
- forwarded X11 connections

These apps implemented as channels multiplexed into an encrypted tunnel; the tunnel is provided by the SSH Transport Layer Protocol.

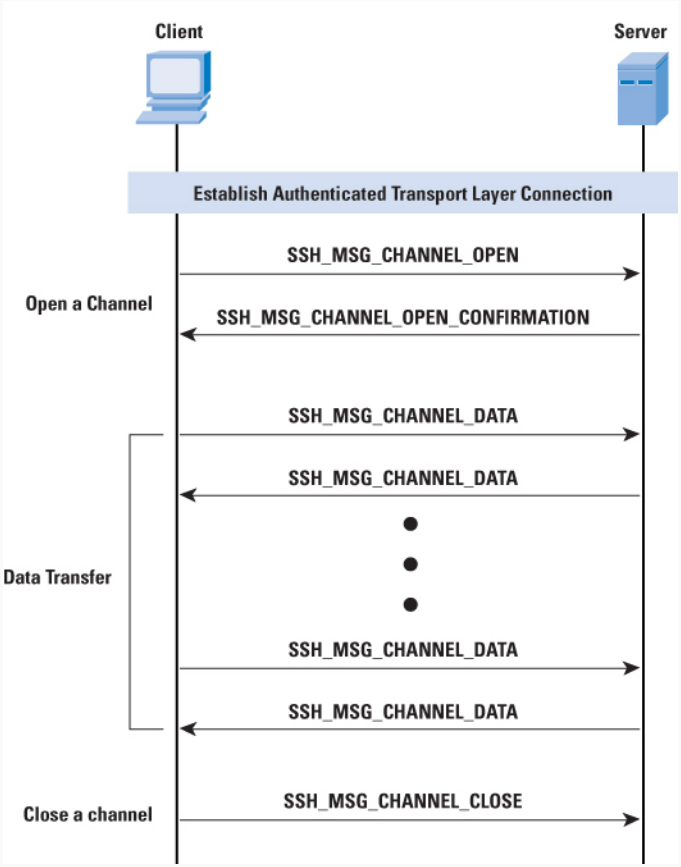

Figure 6: SSH Connection Protocol Message Exchange (source: cisco.com).

SSH Connection Protocol - TCP/IP Port Forwarding

Port forwarding or SSH tunneling is one of the most useful features of SSH. Using port forwarding one can convert any insecure TCP connection into a secure SSH connection.

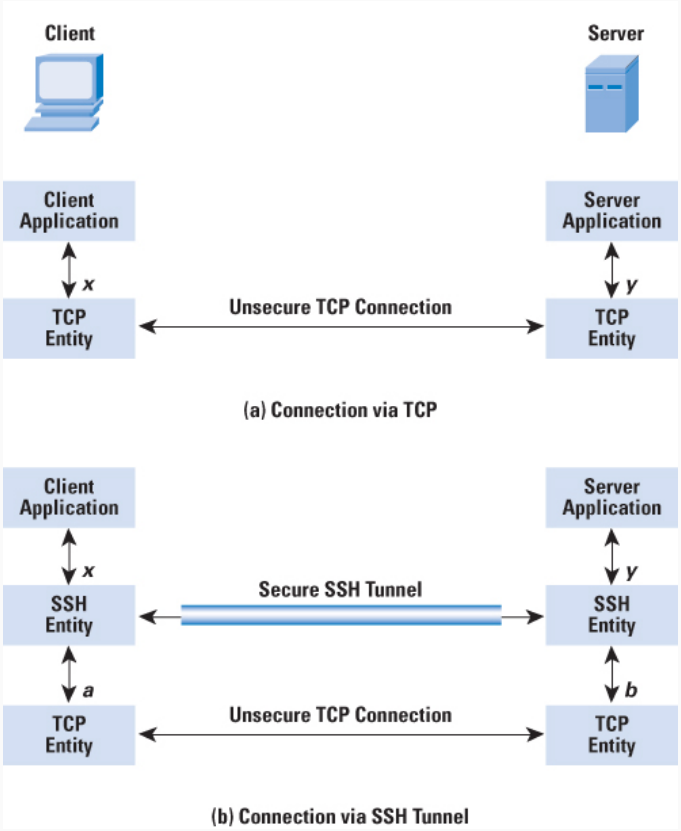

Figure 7: SSH Tunneling (source: cisco.com).

SSH Port Forwarding - Local Port Forwarding

Connections from the SSH client are forwarded via the SSH server, to a destination server.

```
ssh -L sourcePort:DEST-HOST:destPort SSH-HOST
```

Connect with SSH to the SSH-HOST, forward all TCP connection attempts to the sourcePort (on the local machine) to the destPort on the DEST-HOST, which can be reached from the SSH-HOST.

Examples:

```
ssh -L 8080:localhost:80 fesb.hr
```

```
ssh -L 8080:google.com:80 fesb.hr
```

SSH Port Forwarding - Local Port Forwarding

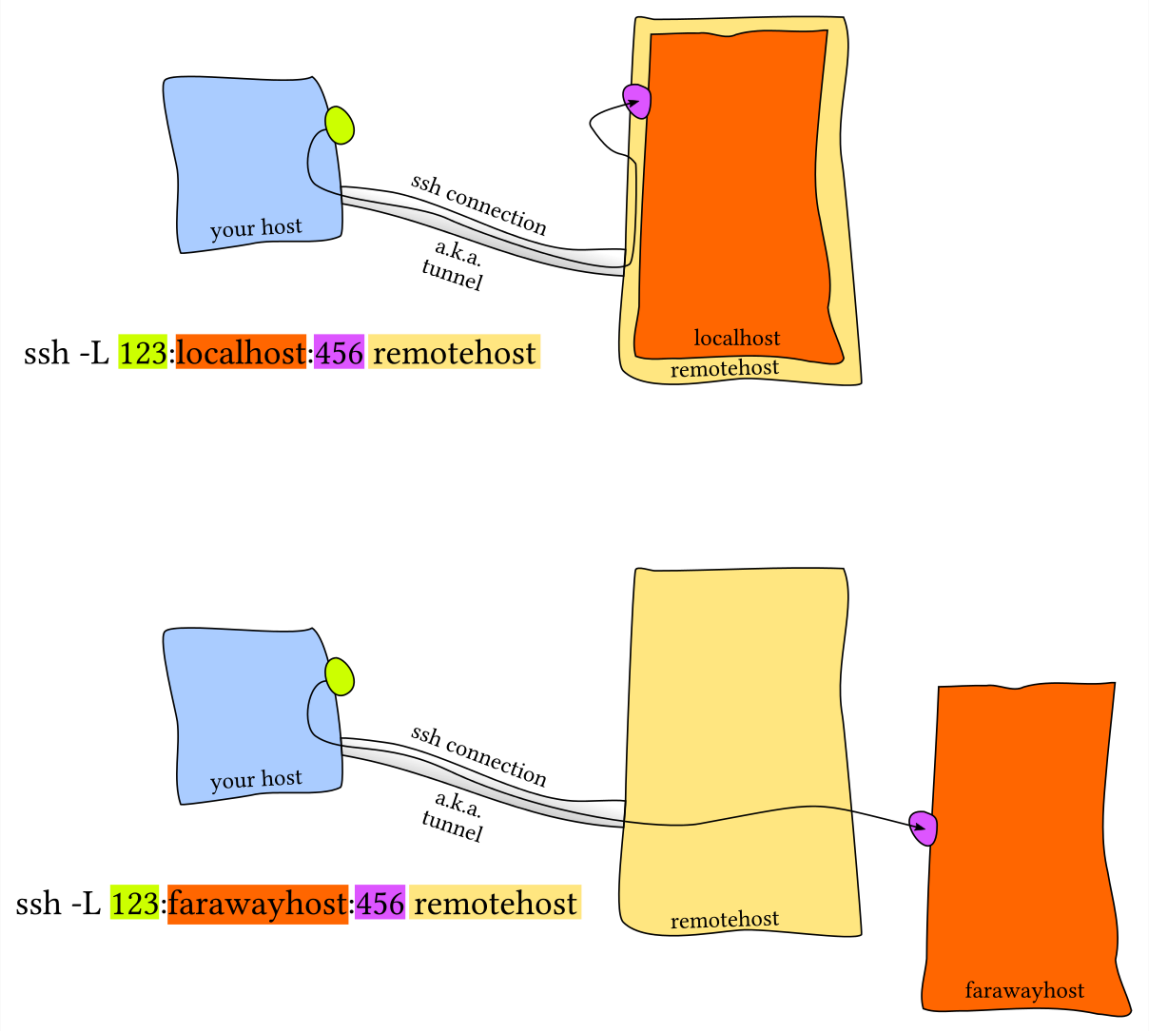

Figure 8: Local port forwarding (from: unix.stackexchange.com).

SSH Port Forwarding - Remote Port Forwarding

Connections from the SSH server are forwarded via the SSH client, then to a destination server.

```
ssh -R sourcePort:DEST-HOST:destPort SSH-HOST
```

Connect with SSH to the SSH-HOST, forward all TCP connection attempts to the sourcePort (on the SSH-HOST) to the destPort on the DEST-HOST, which can be reached from the local host.

Examples:

```
ssh -R 80:localhost:80 fesb.hr
```

```
ssh -R 80:google.com:80 fesb.hr
```

SSH Port Forwarding - Remote Port Forwarding

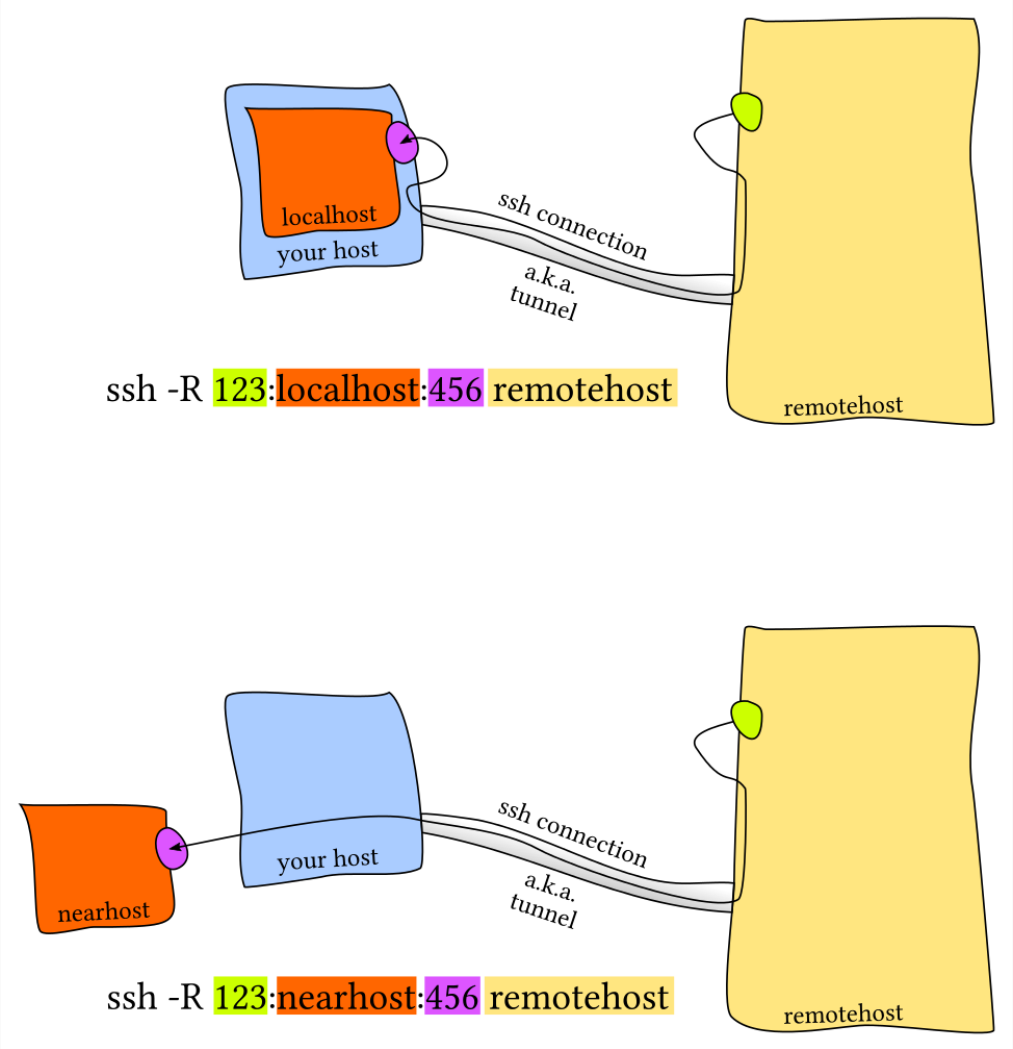

Figure 9: Remote port forwarding (from: unix.stackexchange.com).