

SSH

SECURE SHELL

**OVERVIEW OF THE SECURE SHELL PROTOCOL
FOR SECURE REMOTE SESSIONS AND
PORT FORWARDING**

**Peter R. Egli
INDIGOO.COM**

Contents

1. SSH in a nutshell
2. SSH-1 architecture
3. SSH-1 protocol
4. SSH-1 server (host) authentication
5. SSH-1 client authentication
6. SSH-1 keys
7. SSH-1 session trace
8. SSH configuration
9. SSH port forwarding / SSH tunneling

1. SSH in a nutshell (1/2)

Why SSH?

SSH ([RFC4250](#) et.al.) is a secure replacement for TELNET, rcp, rlogin and rsh for login, remote execution of commands and file transfer.

SSH security:

Security-wise SSH provides:

1. Confidentiality (nobody can read the message content).
2. Integrity (guarantee that data is unaltered on transit).
3. Authentication (of client and server).

This provides protection against:

- IP spoofing
- IP source routing
- DNS spoofing
- Password interception
- Eavesdropping

SSH versions:

There exist 2 (incompatible) versions of SSH:

- SSH-1
- SSH-2

SSH-1.x (1.5, 1.99) are updates of version 1 and thus compatible with SSH-1.

1. SSH in a nutshell (2/2)

SSH encryption:

SSH uses symmetric and asymmetric (public/private) keys for encryption.

SSH supports various different encryption algorithms like 3DES, AES, Blowfish, IDEA.

SSH can be used for:

- Secure remote shell (secure system administration)
- Port forwarding / tunneling (securely circumvent firewalls to access a remote system)
- X11- or VNC-session tunneling (secure remote desktop connection)

SSH comes with some administrator tools:

- SSH key generation with ssh-keygen
- ssh-agent (manage private keys for client)
- ssh-add (register new key with SSH agent)
- make-ssh-known-hosts (list of known host keys of a domain)

SSH compression:

SSH optionally supports compression (gzip= GNU zip).

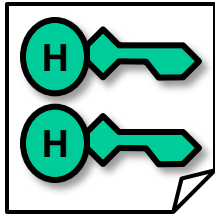
Popular SSH clients:

Windows: PuTTY, TTSsh (Teraterm), WinSCP.

Linux: OpenSSH client.

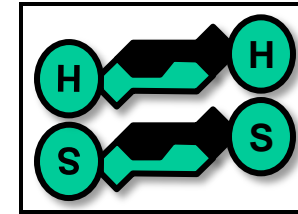
2. SSH-1 architecture

SSH Client

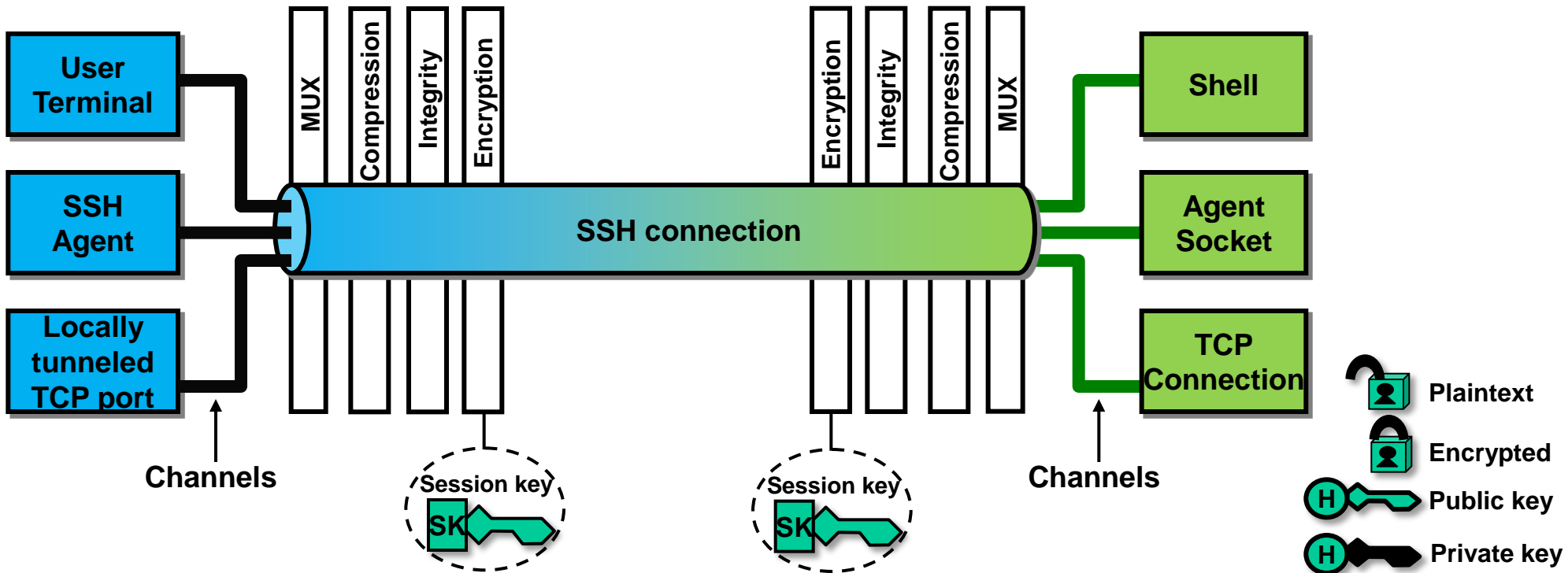
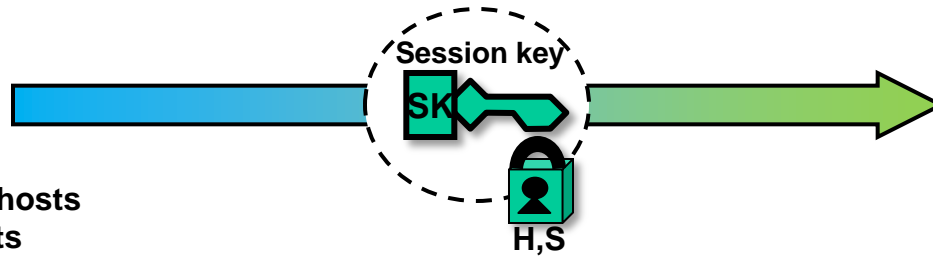


`$HOME/.ssh/known_hosts`
`/etc/ssh_known_hosts`

SSH Server

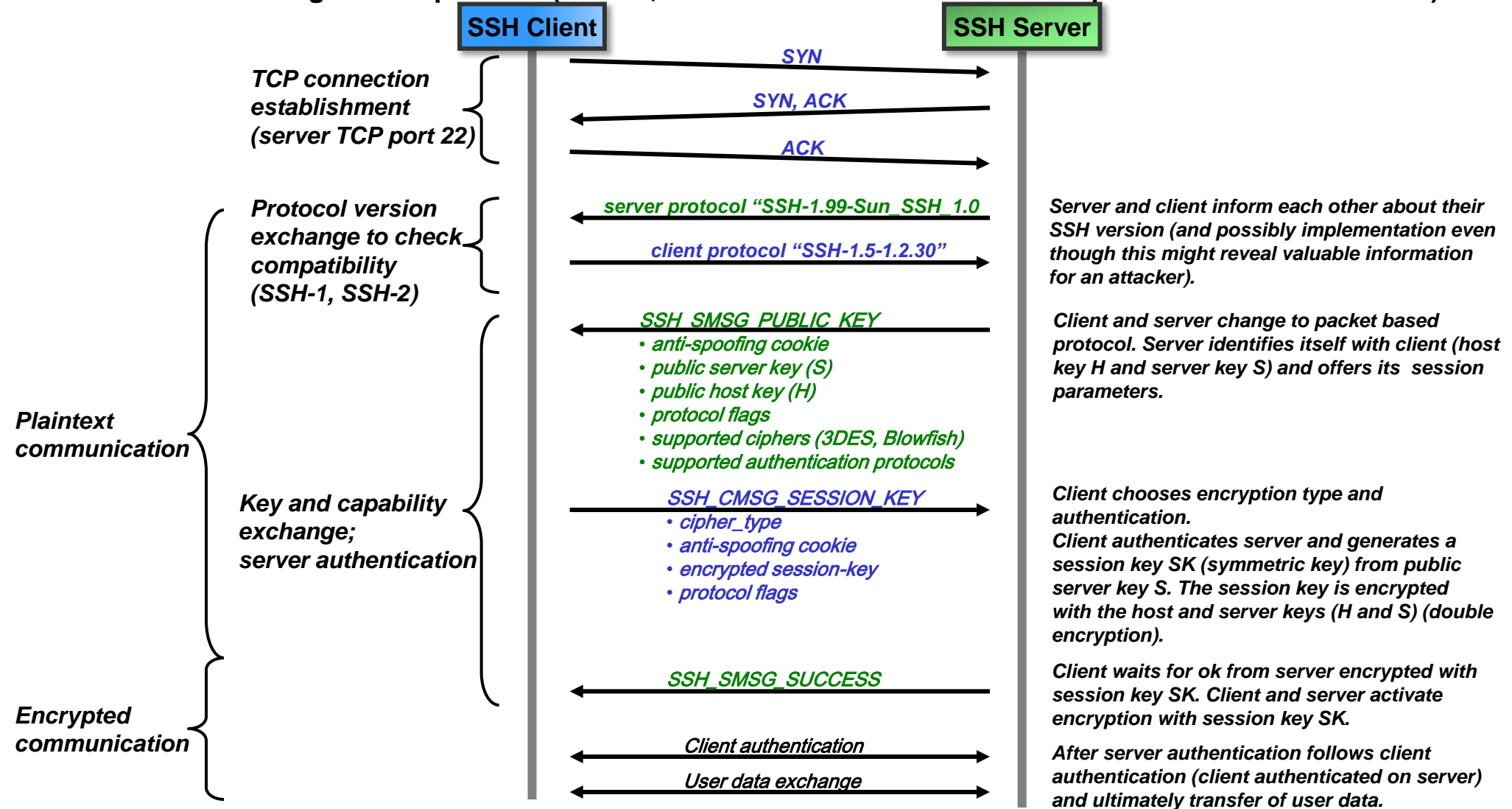


Public and private host and server keys (H and S)



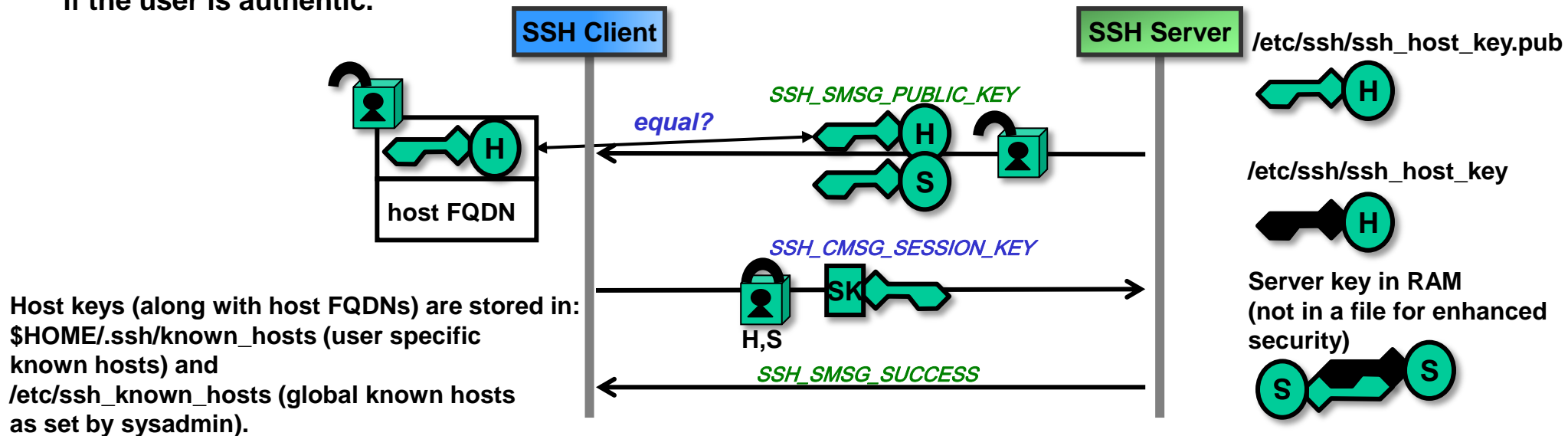
3. SSH-1 protocol

SSH uses a message based protocol (inband, same TCP connection for SSH-1 protocol and for user data).



4. SSH-1 server (host) authentication - client verifies if server is authentic (1/2)

Server (host) authentication by the client is usually employed in order to establish an SSH connection. Servers often do not authenticate the client and simply rely on the user login (username/password) to verify if the user is authentic.



*Client compares public host key from server with host key in `ssh_known_hosts`; if there is a match the client proceeds, otherwise the client issues a security warning and leaves it to user to add (new/changed) server (and host key) to `ssh_known_hosts`.
→ This thwarts man-in-the-middle attacks (but not for the first time a client connects to a host).
→ Sysadmin can control behavior with setting of 'StrictHostKeyConfig' in file `/etc/ssh/ssh_config` (values 'ask'/'no'/'yes').*

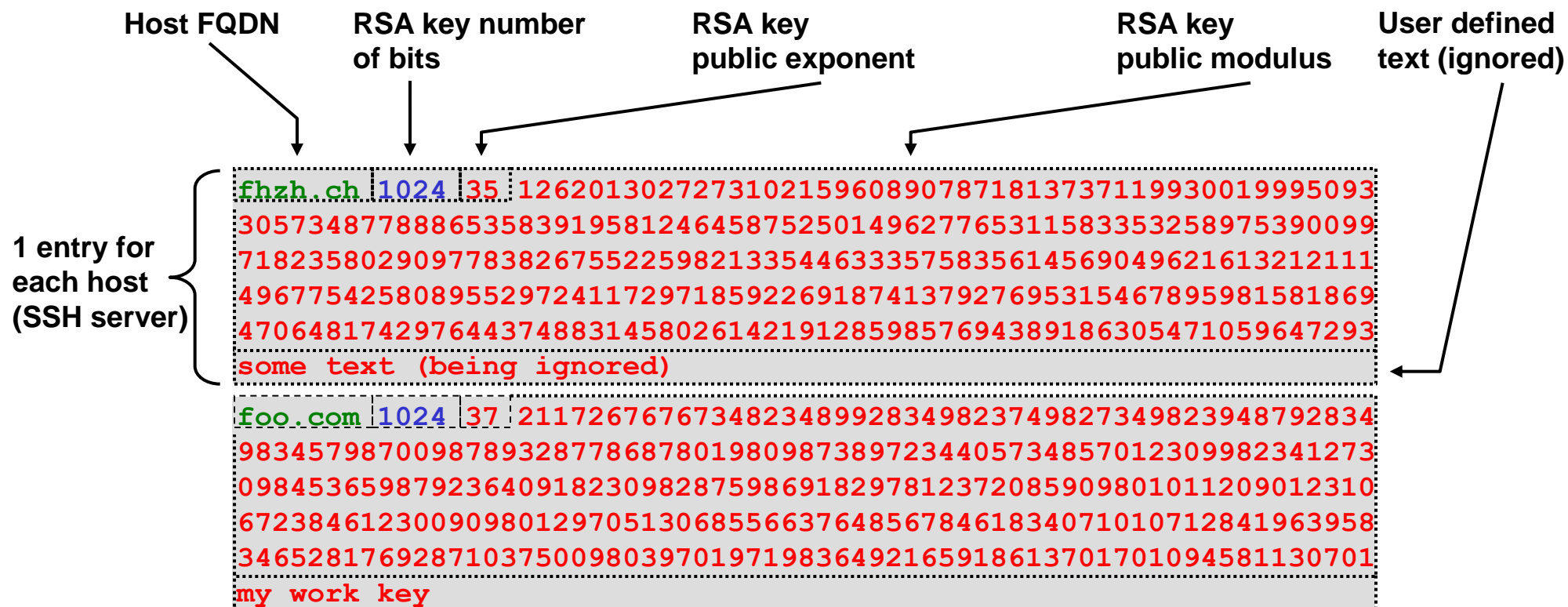
Server authentication is completed with `SSH_MSG_SUCCESS` (encrypted with session key SK). If the client can correctly decrypt the message `SSH_MSG_SUCCESS` authentication is successful because only a genuine server could have decrypted encrypted session key in `SSH_MSG_SESSION_KEY` with its private host and server keys (H and S).

- Plaintext
- Encrypted
- Public key
- Private key
- Asymmetric key(s)
- Symmetric key

4. SSH-1 server (host) authentication - client verifies if server is authentic (2/2)

Each trusted server has an entry in the file `known_hosts` (`~/.ssh/known_hosts`) on the client.

Contents of `~/.ssh/known_hosts` file (public host keys):

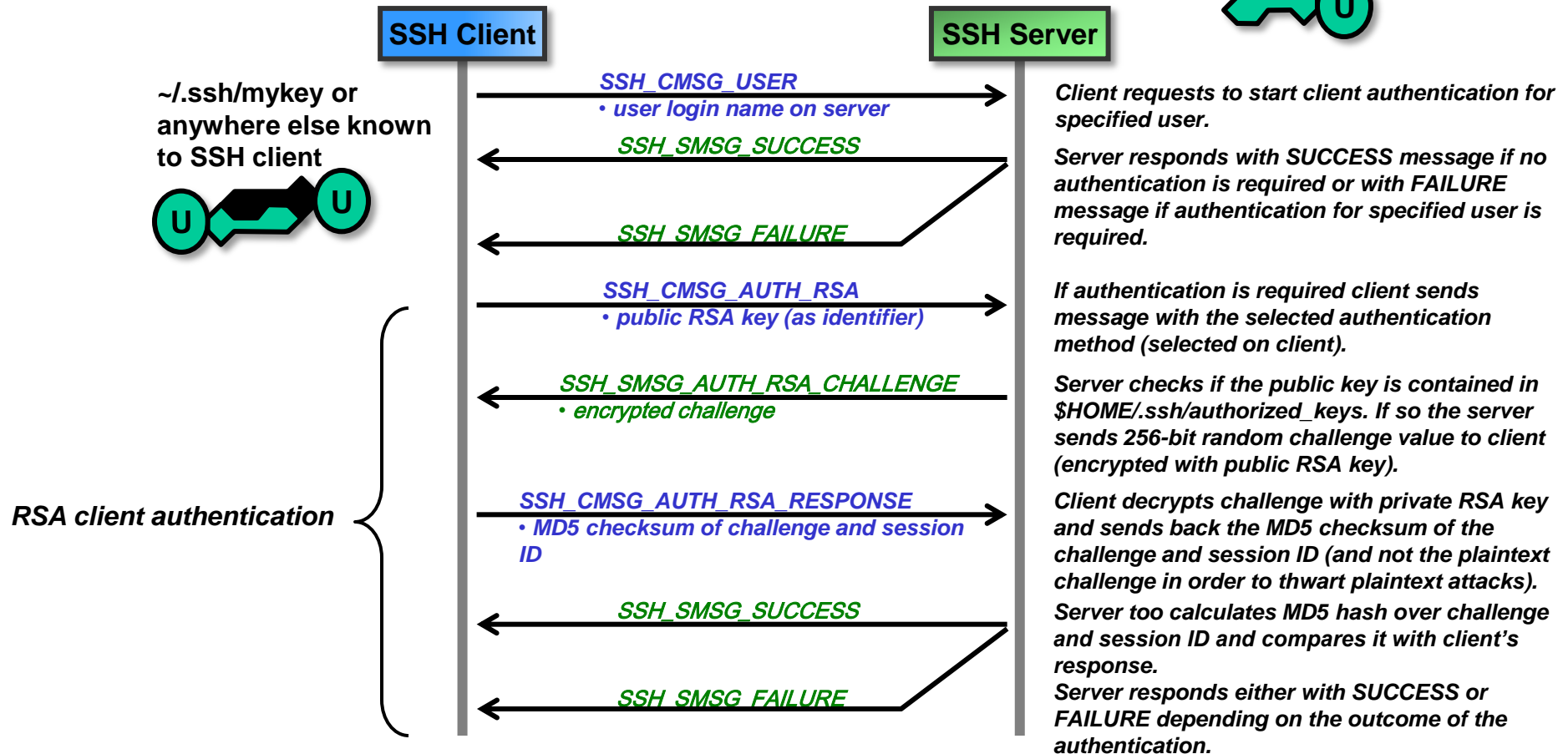


5. SSH-1 client authentication – server verifies if client is authentic (1/3)

Option 1 - RSA public key client authentication:

- 😊 Most secure client authentication method for SSH-1 (private key is protected by passphrase).
- 😊 No secret authentication data is stored on server (only public RSA key).
- 😞 Authentication is independent of client machine (IP address, FQDN).
- 😞 Users must generate and manage keys and authorization files.
- 😞 More difficult to debug if something does not work.

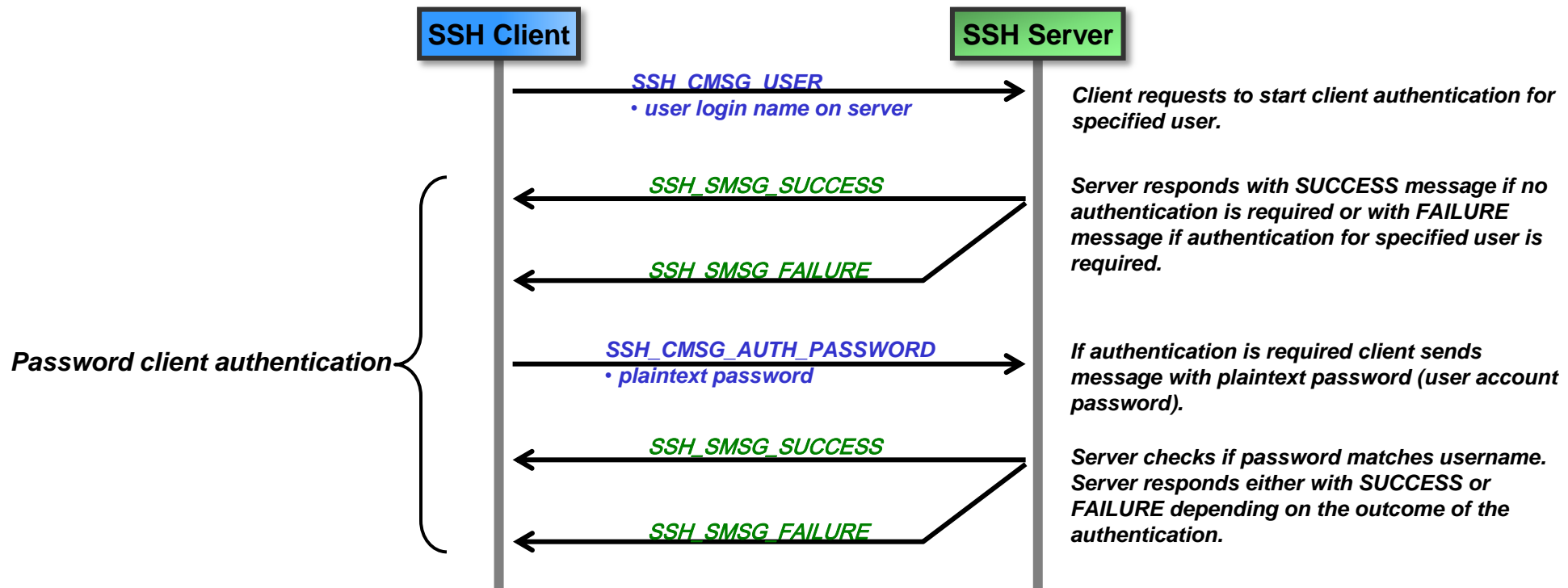
~/.ssh/mykey.pub



5. SSH-1 client authentication – server verifies if client is authentic (2/3)

Option 2 - Plaintext password client authentication:

- 😊 Simple (no configuration required, no need to carry around a private key).
- 😊 No secret authentication data is stored on server (only public RSA key).
- 😊 Authentication is independent of client machine (IP address, FQDN).
- 😊 Plaintext password is naturally encrypted with session key SK.
- 😞 Less secure than public-key authentication (RSA) since password could be cracked on a compromised server.



5. SSH-1 client authentication – server verifies if client is authentic (3/3)

Option 3 - Trusted host client authentication (Rhosts and RhostsRSA):

- 😊 Simple, no need for user to enter passwords or passphrases; this makes it suited for automation (scripts, cron jobs etc.).
- 😞 Authentication is bound to client machine and not user.
- 😞 Usually this authentication method is disabled on servers since it is deemed insecure.





Option 4 - Trusted Information Systems (TIS) client authentication:

TIS improves password authentication in that it uses a password only once (OTP One-Time Passwords). In TIS the server sends a challenge to the client which presents challenge to user. The user is requested to enter a password that he reads from a device (SecurID etc.) or software. The client sends the response back to the server for authentication.

Option 5 - Kerberos authentication

Option 6 - S/Key (one-time password scheme for Unix-type systems).

6. SSH-1 keys and keys and keys...

Name	Lifetime	Generated by	Type	Description
User key U 	Persistent	User	Public (asymmetric)	Identification of a user on a server. This key is used by the SSH client to authenticate a user to the server. (key generated with <code>/etc/ssh/ssh-keygen</code>).
Session key SK 	One session	Client (and server)	Private (symmetric)	Used for encrypting the entire session. The session key is randomly generated and is thus independent of host and server keys (H, S). Both client and server use the same session key for encryption/decryption in both directions of the communication.
Host key H 	Persistent	Sysadmin	Public (asymmetric)	Used for identification / authentication of a server (host) on a client (server authentication). In case of multiple SSH servers on the same machine each instance of server may have its own host key. Also used for session key encryption (together with server key for double encryption).
Server key S 	Limited (default 1 hour)	Server	Public (asymmetric)	Used for encrypting the session key before transmission. Also used for server authentication on client. The server key is never stored in the file system but rather is kept in RAM (in running instance of server).

7. SSH-1 session trace

SSH trace with password client authentication:

```
C: 1 0.000000 192.168.1.15 -> 193.5.54.112 TCP 1962 > 22 [SYN] Seq=4121554192 Ack=0 Win=16384 Len=0
S: 2 0.020510 193.5.54.112 -> 192.168.1.15 TCP 22 > 1962 [SYN, ACK] Seq=608128475 Ack=4121554193 Win=1460 Len=0
C: 3 0.020545 192.168.1.15 -> 193.5.54.112 TCP 1962 > 22 [ACK] Seq=4121554193 Ack=608128476 Win=17280 Len=0
S: 4 0.065252 193.5.54.112 -> 192.168.1.15 SSH Server Protocol: SSH-1.99-Sun_SSH_1.0
C: 5 0.073510 192.168.1.15 -> 193.5.54.112 SSH Client Protocol: SSH-1.5-TTSSH/1.5.4 Win32
S: 6 0.093569 193.5.54.112 -> 192.168.1.15 TCP 22 > 1962 [ACK] Seq=608128497 Ack=4121554219 Win=50400 Len=0
S: 7 0.099686 193.5.54.112 -> 192.168.1.15 SSHv1 Server: Public Key
    Packet Length: 267
    Padding Length: 5
    Msg code: Public Key (2)
    Payload: B41F1D0A80F036F30000030000062303...
C: 8 0.126817 192.168.1.15 -> 193.5.54.112 SSHv1 Client: Session Key
    Packet Length: 148
    Padding Length: 4
    Msg code: Session Key (3)
    Payload: 03B41F1D0A80F036F3040053F168B3B7...
S: 9 0.163857 193.5.54.112 -> 192.168.1.15 TCP 22 > 1962 [ACK] Seq=608128773 Ack=4121554375 Win=50244 Len=0
S: 10 0.229181 193.5.54.112 -> 192.168.1.15 SSHv1 Server: Encrypted packet len=5 (SSH_MSG_SUCCESS)
C: 11 0.356903 192.168.1.15 -> 193.5.54.112 TCP 1962 > 22 [ACK] Seq=4121554375 Ack=608128785 Win=16971 Len=0
C: 12 4.622742 192.168.1.15 -> 193.5.54.112 SSHv1 Client: Encrypted packet len=41 (SSH_MSG_USER)
S: 13 4.683739 193.5.54.112 -> 192.168.1.15 SSHv1 Server: Encrypted packet len=5 (SSH_MSG_FAILURE)
C: 14 4.684010 192.168.1.15 -> 193.5.54.112 SSHv1 Client: Encrypted packet len=41 (SSH_MSG_AUTH_PASSWORD)
S: 15 4.760309 193.5.54.112 -> 192.168.1.15 SSHv1 Server: Encrypted packet len=5 (SSH_MSG_SUCCESS)
C: 16 4.760570 192.168.1.15 -> 193.5.54.112 SSHv1 Client: Encrypted packet len=41 (user data)
...
```

C: Client → server traffic
S: Server → client traffic

8. SSH configuration (1/2)

SSH configuration with password client authentication (RSA keyset):

1. Add server (host) to client's list of known hosts:

TTSSH (Teraterm SSH): menu Setup->TCP/IP

The entry will be added to the file

~/.ssh/known_hosts (Unix) or ssh_known_hosts (TTSSH).

2. Configure client authentication and credentials:

Select RSA key authentication along with proper (private) RSA key file (copied from server in step 4.).

TTSSH: menu Setup->SSH Authentication

3. Save configuration:

TTSSH: menu Setup->Save setup...

8. SSH configuration (2/2)

SSH configuration with public key client authentication (RSA keyset):

1. Configure a public key on server:

Create key on server (or wherever ssh-keygen is available):

```
/etc/ssh/ssh-keygen
```

(execute program on server, generates key pair (public, private) in \$HOME/.ssh/identity.pub)

(choose a pass-phrase for private key encryption)

2. Add created public key to key file:

```
cd $HOME/.ssh
```

```
cat identity.pub >> authorized_keys
```

(this file is examined by SSH server (=sshd))

3. Copy the private key in file 'identity' to PC where TTSSH runs:

→ do this through a SSH session with password authentication

4. Delete private key on server machine:

```
rm identity
```

5. Add server (host) to client's list of known hosts:

TTSSH (Teraterm SSH): menu Setup->TCP/IP

The entry will be added to the file ~/.ssh/known_hosts (Unix) or ssh_known_hosts (TTSSH).

6. Configure client authentication and credentials:

Select RSA key authentication along with proper (private) RSA key file (copied from server in step 3.).

TTSSH: menu Setup->SSH Authentication

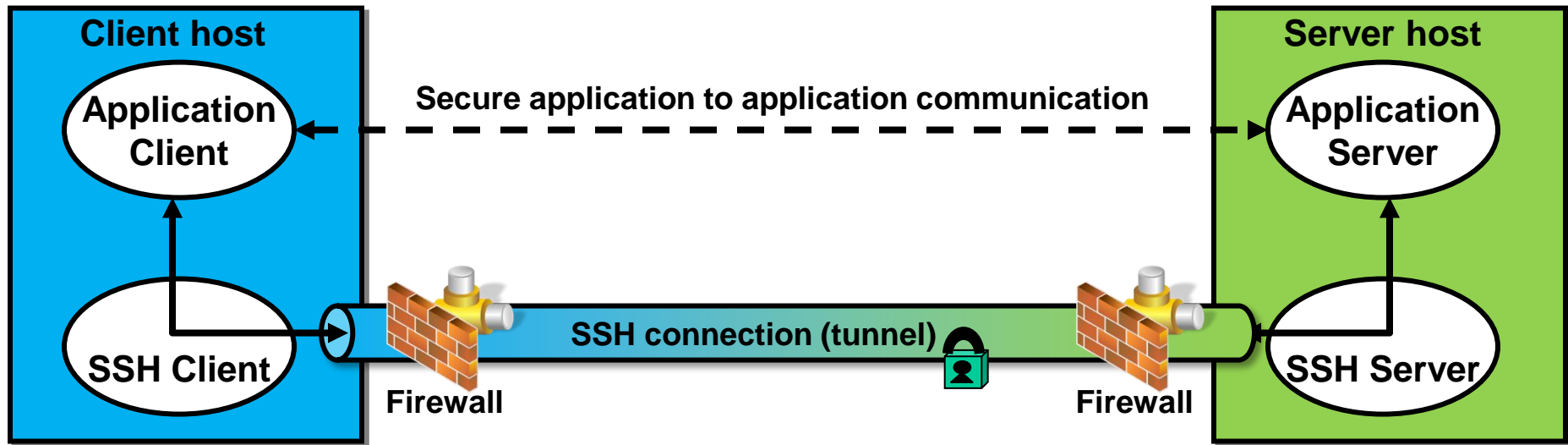
7. Save configuration:

TTSSH: menu Setup->Save setup...

9. SSH port forwarding / SSH tunneling (1/5)

Pass protocols (applications) securely through an unsecure network.

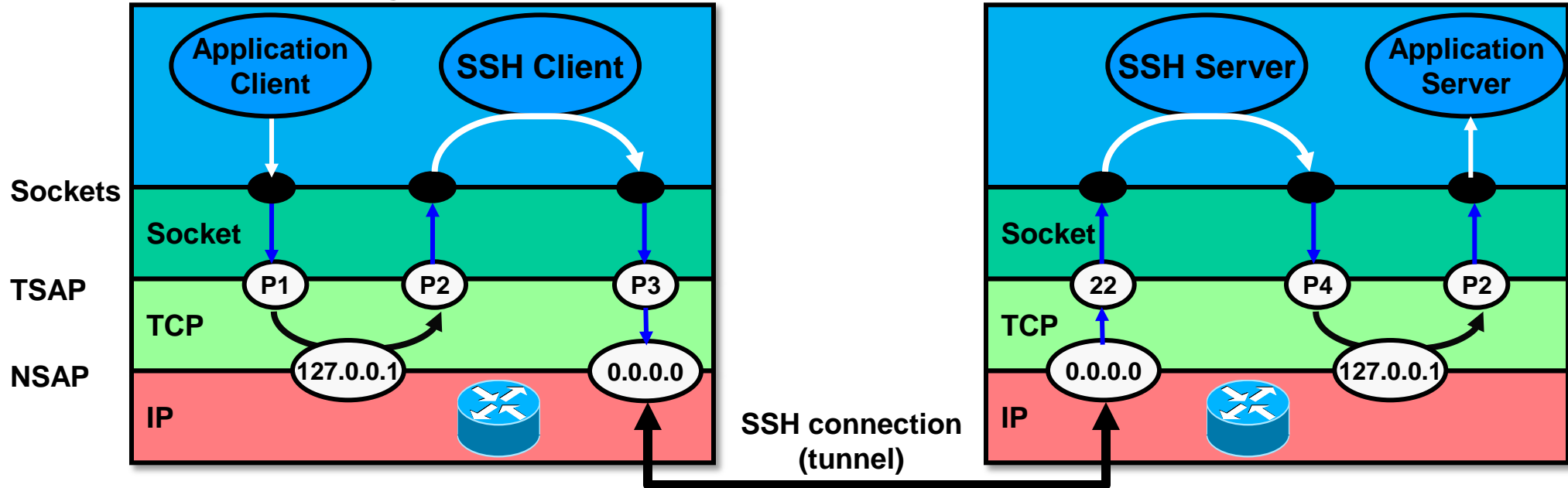
SSH port forwarding allows applications to pass traffic through firewalls that would normally block the application protocol (but not SSH), e.g. when NATP (Network Address Port Translation) is used on the firewall.



- The Connection between application client and server is relayed through a pair of SSH client and server.
- The connection between client and server host is protected (but not the connection between SSH client and application client and between SSH server and application server!).

9. SSH port forwarding / SSH tunneling (2/5)

Local port forwarding:



- Application client and SSH client are on same machine (and accordingly application server and SSH server).
- Local forwarding configuration command:

```
$ ssh -L<local port>:<local host>:<remote port> <remote host>
```

Example: `$ ssh -L 2001:localhost:143 pop.fhzh.ch`

- Forwarding is initiated by client sending `SSH_CMSG_PORT_FORWARD_REQUEST` message to SSH server (along with remote port number). SSH client and server establish a channel for this forwarding within the existing SSH connection. Client application is bound to localhost/P1 and talks to localhost/P2.

P1: source port number of application client's TCP connection (ephemeral port number).

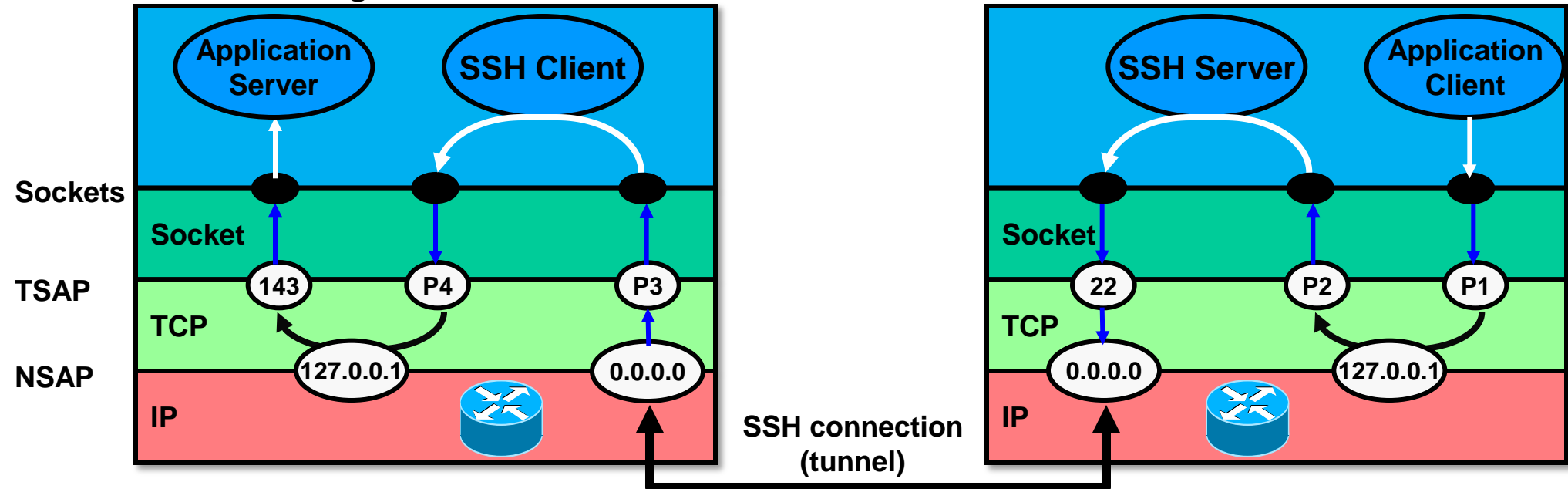
P2: port number to be forwarded; can be well-known port, e.g. 21 for FTP (2001 in example above).

P3: source port number of SSH client's SSH TCP connection (ephemeral port).

P4: source port number of SSH server's TCP connection to application server (ephemeral port).

9. SSH port forwarding / SSH tunneling (3/5)

Remote forwarding:



- Application client and SSH client are on different machines (and accordingly application server and SSH server).
- Remote forwarding configuration command:

```
$ ssh -R<remote port>:<local host>:<local port> <remote host>
```

Example: `$ ssh -R 2001:localhost:143 pop.fhzh.ch`

P1: source port number of application client's TCP connection (ephemeral port number)

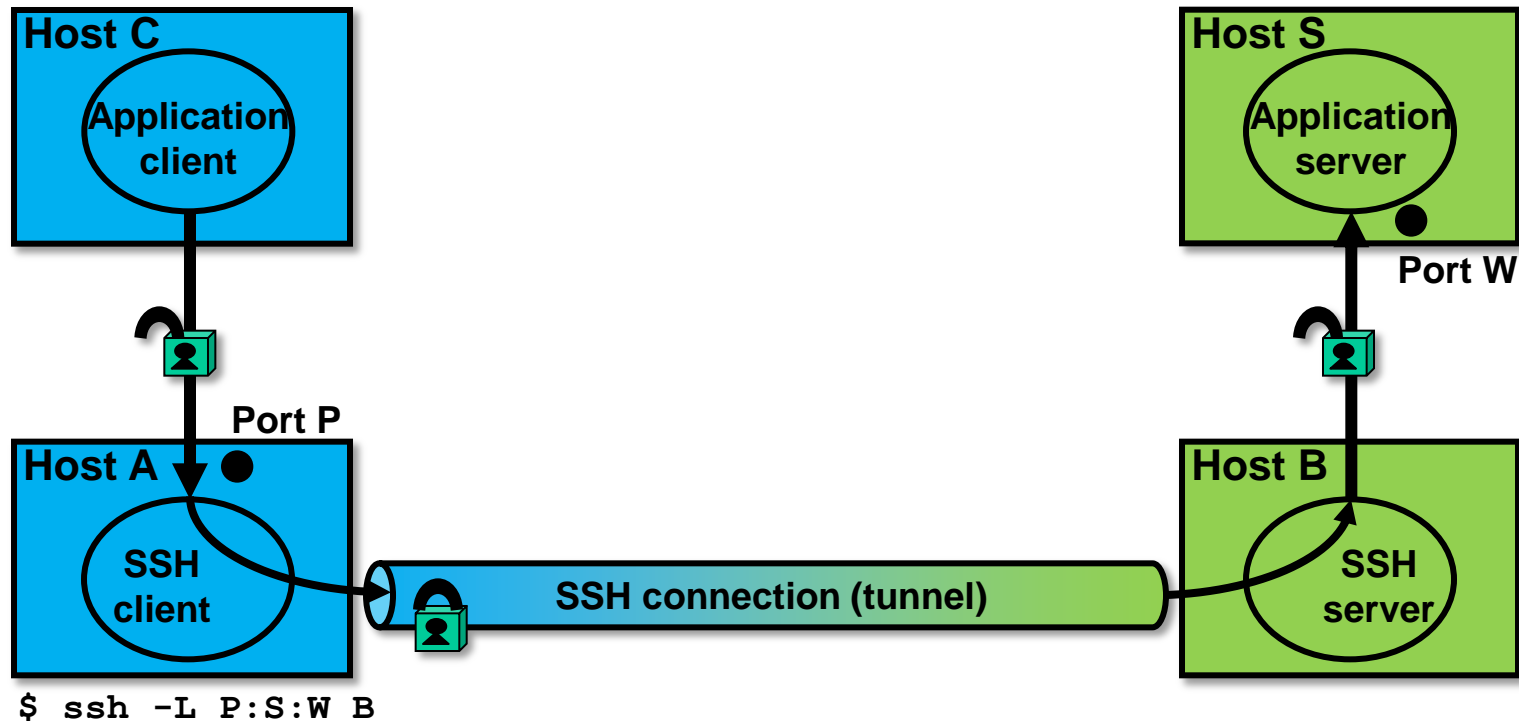
P2: port number to be forwarded; can be well-known port, e.g. 21 for FTP (2001 in example above)

P3: source port number of SSH client's SSH TCP connection (ephemeral port)

P4: source port number of SSH server's TCP connection to application server (ephemeral port)

9. SSH port forwarding / SSH tunneling (4/5)

Off-host-port forwarding:



- Application client/server and SSH client/server are on different machines.
- Local forwarding configuration command (on host A):
 \$ ssh -L<local port>:<server host>:<remote port> <remote host>
Example: \$ ssh -L P:S:W B

9. SSH port forwarding / SSH tunneling (5/5)

FTP forwarding (let local FTP client use SSH):

A. Configuration:

1. Configure port forwarding on TTSSH:

Menu 'Setup'->'SSH Forwarding...'

'Forward local port' = 21 (for FTP).

'to remote machine' = server / host to which to connect through FTP in SSH tunnel.

2. Configure FTP client to enable use of SSH:

Configure FTP client such that it connects to 127.0.0.1 (localhost) and 21 as remote port.

Enable passive mode (PASV=on); in active mode the server will not be able to open a data connection since the client's IP that the server 'sees' is localhost (127.0.0.1).

B. Usage:

1. Log on to desired server through SSH (with TTSSH) thus establishing an SSH secured TCP connection.

2. Start FTP client (will connect to localhost which is forwarded to FTP server through SSH server).

3. Normal file upload/download operation.

→ But: some servers (e.g. FHZH FTP server) will rant if the client wants to open an FTP-Data connection with a different source IP address (public client IP address since FTP-Data does not pass through SSH tunnel) than the FTP control connection (localhost on server=172.10.0.1 when using SSH tunnel); the warning will be something like "425 Possible PASV port theft, can not open data connection" (this is a security feature of the FTP server).

The problem is that FTP clients, when in passive mode, try to open a data connection to the address provided by the server after sending the PASV command (FTP server communicates IP/port number encoded as n1,n2,n3,n4,n5,n6) because opening a connection to localhost/20 would not be possible since server port is dynamic and there is no way to let the SSH client dynamically open (listen) a TCP port on localhost.