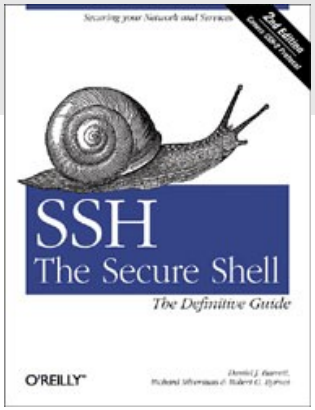
Secure Shell – SSH

Chapter 17.4

Secure Shell – SSH



- Protocol but also an implementation (a program) called SSH
  - RFC 4250-4256
  - Developed 1995 by Tatu Ylönen, University of Helsinki, Finland
  - Application level (similar to SSL)
  - Uses TCP port 22
  - Only version 2 of the protocol should be used (1997)
- Originally a secure replacement for Unix telnet, rsh and rlogin
  - Secure terminal: command-line access to remote systems (servers, routers, …)
  - Allows servers to be identified with a PKI system (“host keys”)
  - Can also multiplex TCP traffic from applications (called port forwarding)
- Many systems support SSH natively
  - Unix/Linux systems for remote access
  - Windows 10 has OpenSSH client and server (under APPS/Add a feature)
  - Routers and switches

Example of a terminal session

One channel used for the terminal, i.e. command-line session (blue text typed by user):

```
% ssh legolas
The authenticity of host 'legolas (129.16.20.40)' can't be established.
RSA key fingerprint is 28:c5:61:86:2d:90:7b:68:03:45:a8:4c:d9:4e:cf:0b.
Are you sure you want to continue connecting (yes/no)?
yes
Warning: Permanently added 'legolas,129.16.20.40' (RSA) to the list of
   known hosts.
Tomas password: *********

legolas> ls
…
```

SSH Packet creation

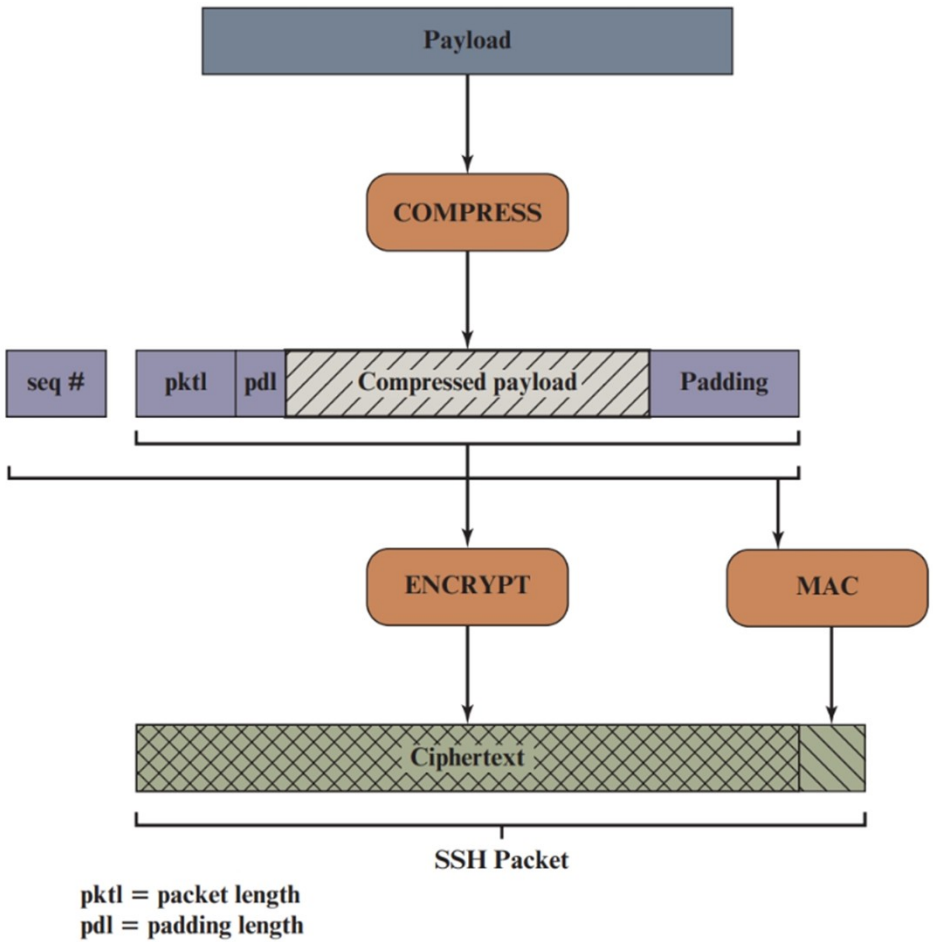


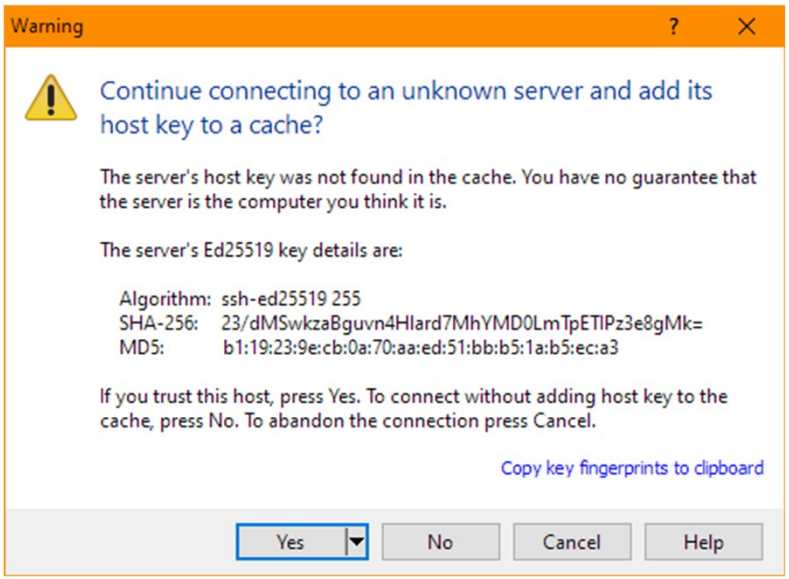
Fig. 17.10

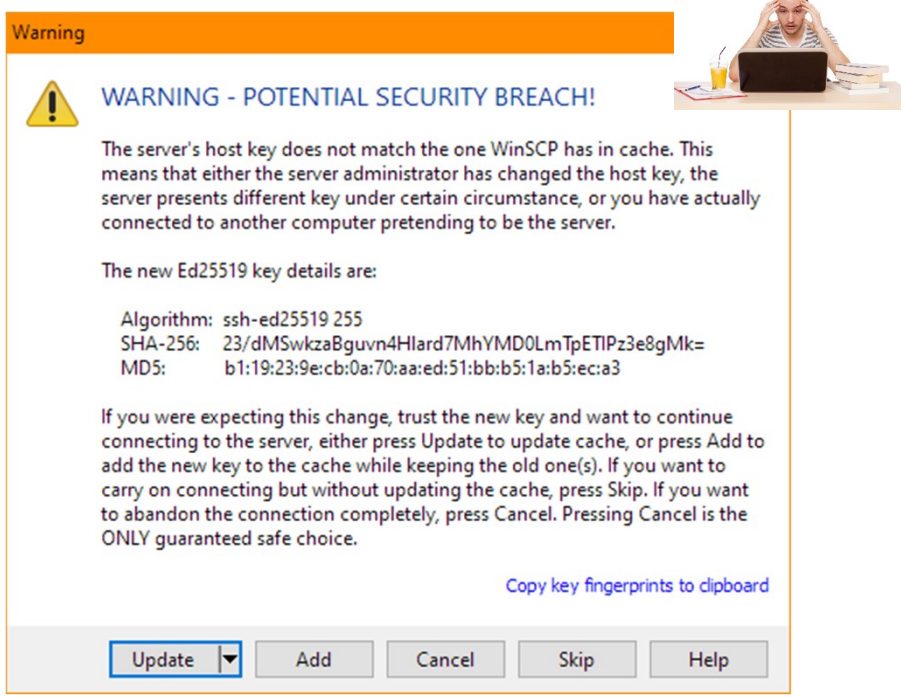
Note that MAC covers sequence number, but it is never transmitted!

Authentication in SSH

- Server host keys:
  - Default: 2048-bit RSA public/private keys
  - Clients maintain a hostname ←→ public_host_key mapping table
  - Public keys must be distributed to clients, preferable off-line to prevent MITM attacks
- Wide user authentication method support
  - Passwords, SecurID, token cards, etc.
  - Clients can also use RSA public/private keys for authentication
    - Often used for automated logins
  - Supports two-factor authentication (e.g. password + RSA keys)
- Many implementations support certificates (own format, not X.509)
  - CA signs the keys for a user or a server – but you have to create the CA
  - The public key is spread and used to authenticate the user or the server
  - Own CA infrastructure needs to be built…

If host key does not match...



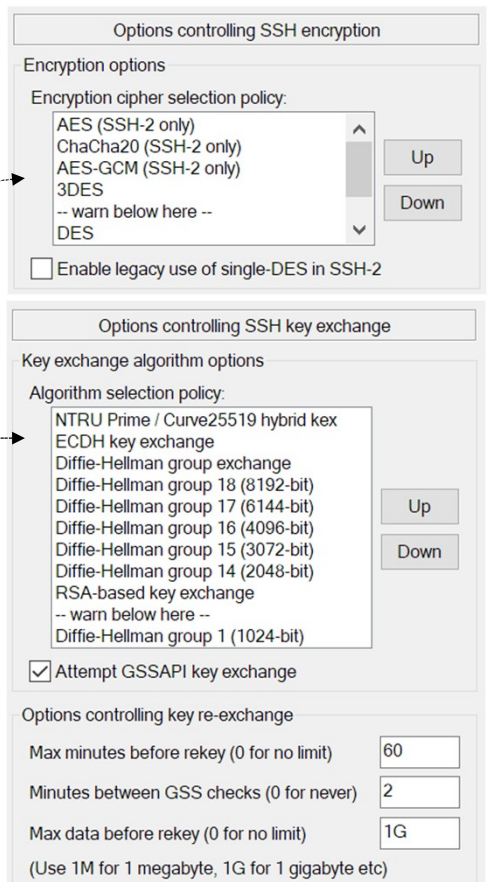


Selecting algorithms

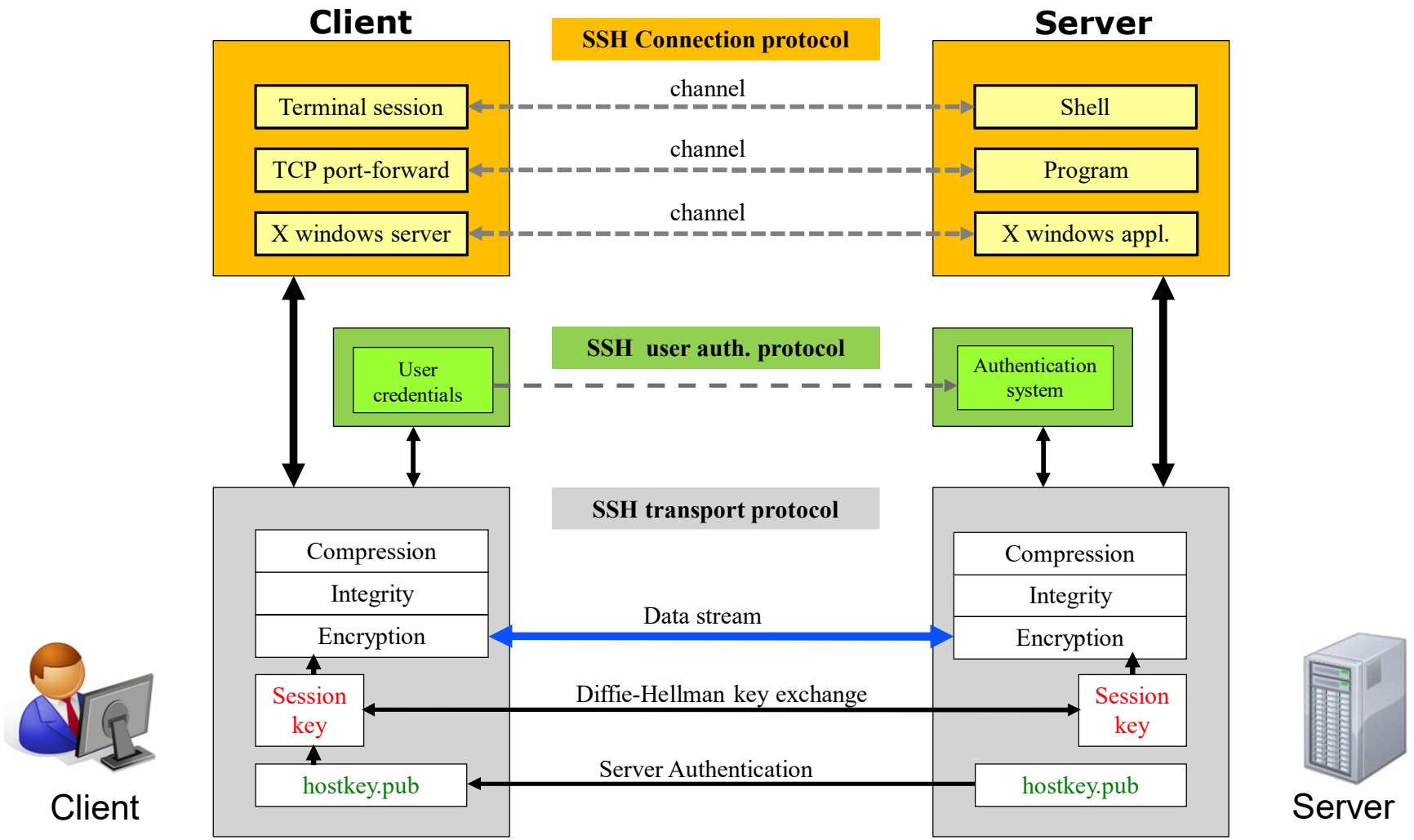
- Algorithms and ciphers are negotiated at connection time
  - Algorithms negotiated for each direction – they may be different
  - Both parties send a full list, first match is used
- Negotiated algorithms:
  - Ciphers: AES, 3-DES, etc.
  - MAC: HMAC with SHA-1 or MD5
  - Compression: none or zlib (RFC 1950, 1951)
- EC or D-H normally used for Key Exchange
  - Pre-defined D-H groups exist for efficiency and compatibility
  - Group 1: $g^x \bmod p$ can be calculated as:

    $g = 2;\quad p = 2^{1024} - 2^{960} - 1 + 2^{64} \times (\text{floor}(2^{894} \times \pi) + 129093)$
  - Standard recommends that g=2 since it gives efficiency in multiplication
  - Group 1 too short to be really secure

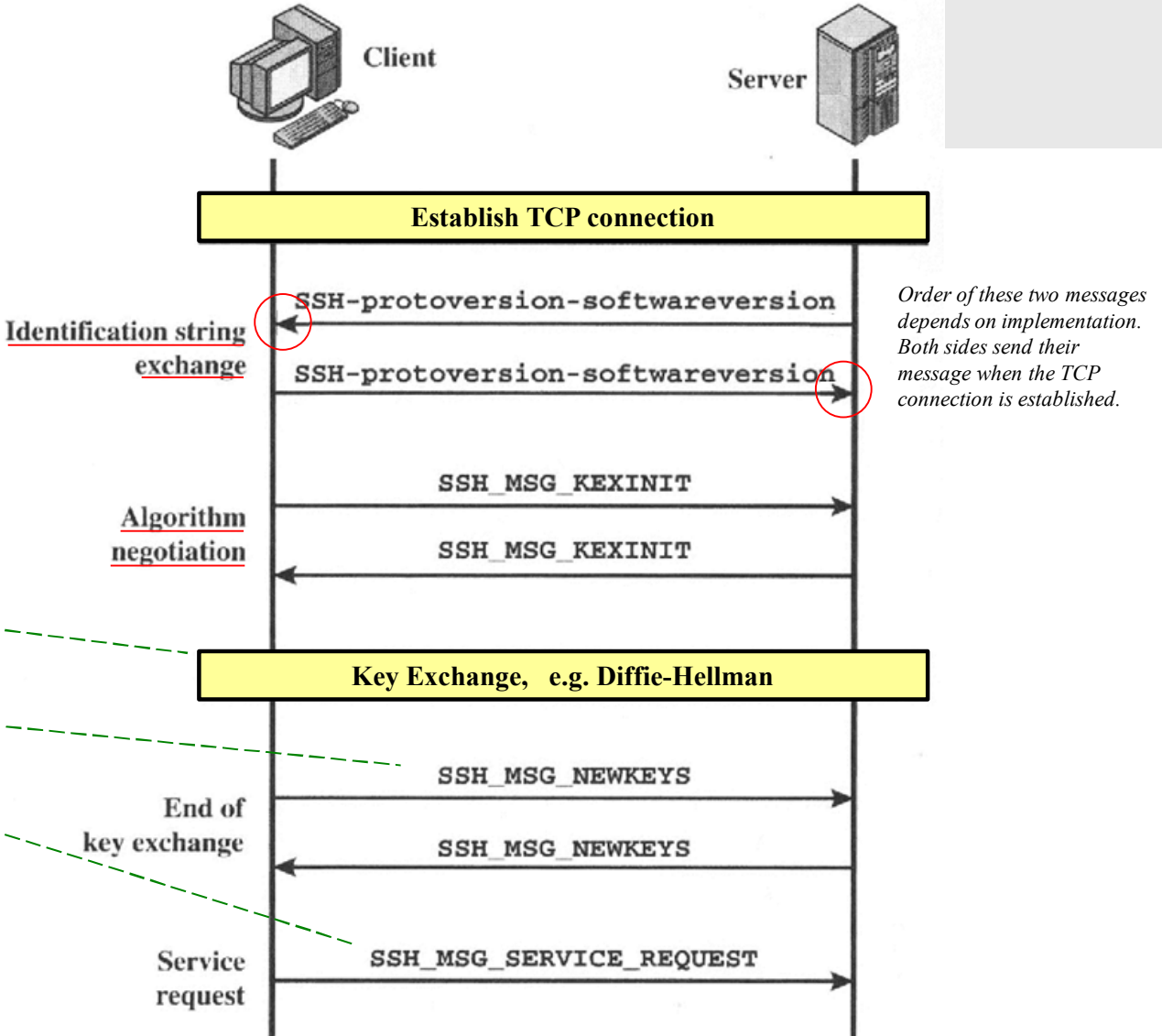
Putty:



The SSH2-Protocol Architecture



SSH Transport Protocol



- Key Exchange uses server hostkeys in D-H negotiation
- NEWKEYS = begin encryption (use new keys)
- SERVICE_REQEST = ready for user authentication or connection protocol

Fig. 17.9

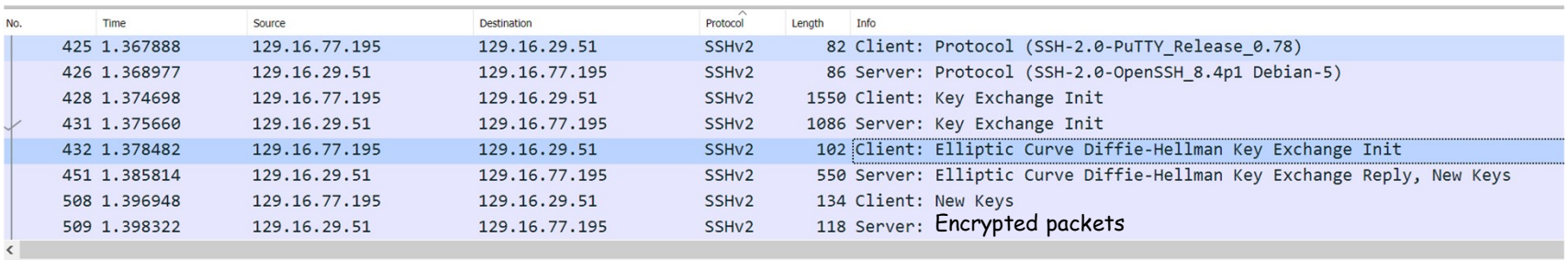
Client algorithm exchange

No.	Time	Source	Destination	Protocol	Length	Info
425	1.367888	129.16.77.195	129.16.29.51	SSHv2	82	Client: Protocol (SSH-2.0-PuTTY_Release_0.78)
426	1.368977	129.16.29.51	129.16.77.195	SSHv2	86	Server: Protocol (SSH-2.0-OpenSSH_8.4p1 Debian-5)
428	1.374698	129.16.77.195	129.16.29.51	SSHv2	1550	Client: Key Exchange Init
431	1.375660	129.16.29.51	129.16.77.195	SSHv2	1086	Server: Key Exchange Init
432	1.378482	129.16.77.195	129.16.29.51	SSHv2	102	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
451	1.385814	129.16.29.51	129.16.77.195	SSHv2	550	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
508	1.396948	129.16.77.195	129.16.29.51	SSHv2	134	Client: New Keys
509	1.398322	129.16.29.51	129.16.77.195	SSHv2	118	Server: Encrypted packets

```
> Frame 428: 1550 bytes on wire (12400 bits), 1550 bytes captured (12400 bits) on interface \Device\NPF_{54440BF1-BFE1-4607-847E-9BEE536E4147}, id 0
> Ethernet II, Src: Dell_5a:c8:16 (b0:7b:25:5a:c8:16), Dst: Cisco_ce:66:07 (28:ac:9e:ce:66:07)
> Internet Protocol Version 4, Src: 129.16.77.195, Dst: 129.16.29.51
> Transmission Control Protocol, Src Port: 58445, Dst Port: 22, Seq: 29, Ack: 33, Len: 1496
v SSH Protocol
  v SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
      Packet Length: 1492
      Padding Length: 4
    v Key Exchange (method:curve25519-sha256)
        Message Code: Key Exchange Init (20)
      v Algorithms
          Cookie: f491dfef4a033c350b68b4c8092fd412
          kex_algorithms length: 470
          kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.com,curve448-sha512,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-s
          server_host_key_algorithms length: 123
          server_host_key_algorithms string: ssh-ed25519,ssh-ed448,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,rsa-sha2-512,rsa-sha2-
          encryption_algorithms_client_to_server length: 235
          encryption_algorithms_client_to_server string [truncated]: aes256-ctr,aes256-cbc,rijndael-cbc@lysator.liu.se,aes192-ctr,aes192-cbc,aes128-c
          encryption_algorithms_server_to_client length: 235
          encryption_algorithms_server_to_client string [truncated]: aes256-ctr,aes256-cbc,rijndael-cbc@lysator.liu.se,aes192-ctr,aes192-cbc,aes128-c
          mac_algorithms_client_to_server length: 155
          mac_algorithms_client_to_server string: hmac-sha2-256,hmac-sha1,hmac-sha1-96,hmac-md5,hmac-sha2-256-etm@openssh.com,hmac-sha1-etm@openssh.c
          mac_algorithms_server_to_client length: 155
          mac_algorithms_server_to_client string: hmac-sha2-256,hmac-sha1,hmac-sha1-96,hmac-md5,hmac-sha2-256-etm@openssh.com,hmac-sha1-etm@openssh.c
          compression_algorithms_client_to_server length: 26
          compression_algorithms_client_to_server string: none,zlib,zlib@openssh.com
          compression_algorithms_server_to_client length: 26
          compression_algorithms_server_to_client string: none,zlib,zlib@openssh.com
```

Wireshark's conclusion of preferred algorithm

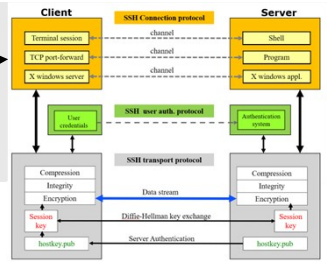
Exchange of key material



No.	Time	Source	Destination	Protocol	Length	Info
425	1.367888	129.16.77.195	129.16.29.51	SSHv2	82	Client: Protocol (SSH-2.0-PuTTY_Release_0.78)
426	1.368977	129.16.29.51	129.16.77.195	SSHv2	86	Server: Protocol (SSH-2.0-OpenSSH_8.4p1 Debian-5)
428	1.374698	129.16.77.195	129.16.29.51	SSHv2	1550	Client: Key Exchange Init
431	1.375660	129.16.29.51	129.16.77.195	SSHv2	1086	Server: Key Exchange Init
432	1.378482	129.16.77.195	129.16.29.51	SSHv2	102	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
451	1.385814	129.16.29.51	129.16.77.195	SSHv2	550	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
508	1.396948	129.16.77.195	129.16.29.51	SSHv2	134	Client: New Keys
509	1.398322	129.16.29.51	129.16.77.195	SSHv2	118	Server: Encrypted packets

```
> Frame 432: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface \Device\NPF_{54440BF1-BFE1-4607-847E-9BEE536E4147}, id 0
> Ethernet II, Src: Dell_5a:c8:16 (b0:7b:25:5a:c8:16), Dst: Cisco_ce:66:07 (28:ac:9e:ce:66:07)
> Internet Protocol Version 4, Src: 129.16.77.195, Dst: 129.16.29.51
> Transmission Control Protocol, Src Port: 58445, Dst Port: 22, Seq: 1525, Ack: 1065, Len: 48
v SSH Protocol
  v SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
      Packet Length: 44
      Padding Length: 6
    v Key Exchange (method:curve25519-sha256)
        Message Code: Elliptic Curve Diffie-Hellman Key Exchange Init (30)
        ECDH client's ephemeral public key length: 32
        ECDH client's ephemeral public key (Q_C): ac7858f3d95cafd31cabbd39faa06d6e10064fcd9ed2fdf9546a71844a437648
      Padding String: 666ca3a75cc2
      Sequence number: 1
```

SSH Connection Protocol



- All communication uses channels
- Either side can open a channel
- Channel types:
  - Terminal session (remote commands)
  - X11 (X-windows apps)
  - Port forwarding (TCP tunneling)

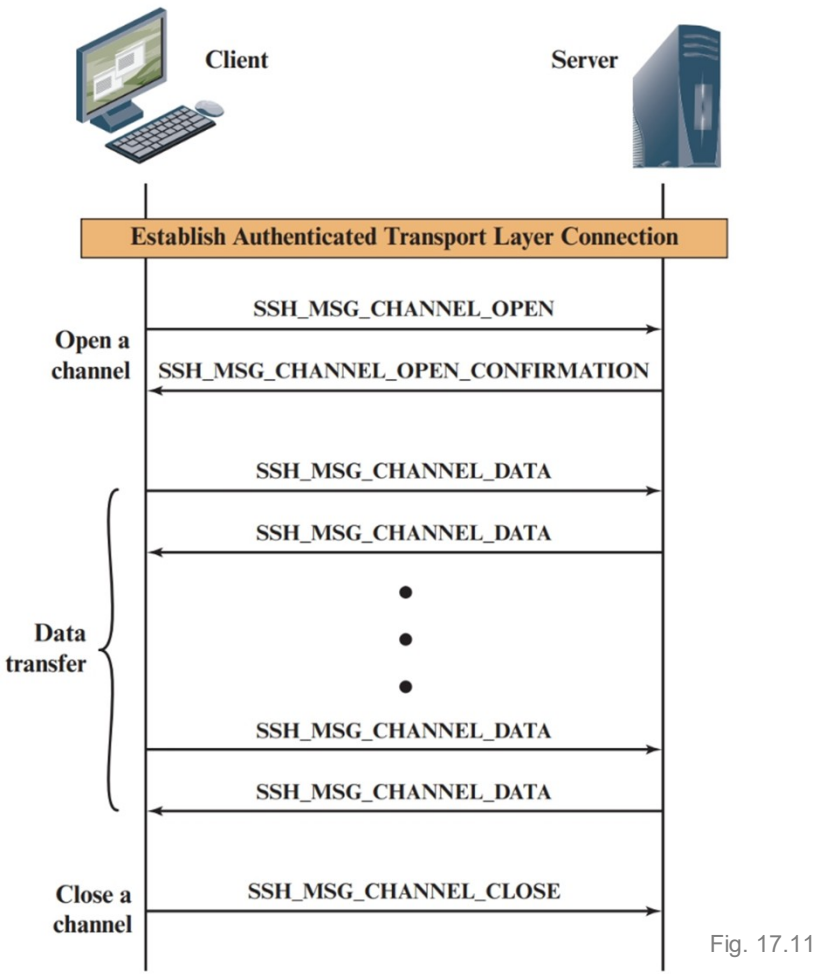


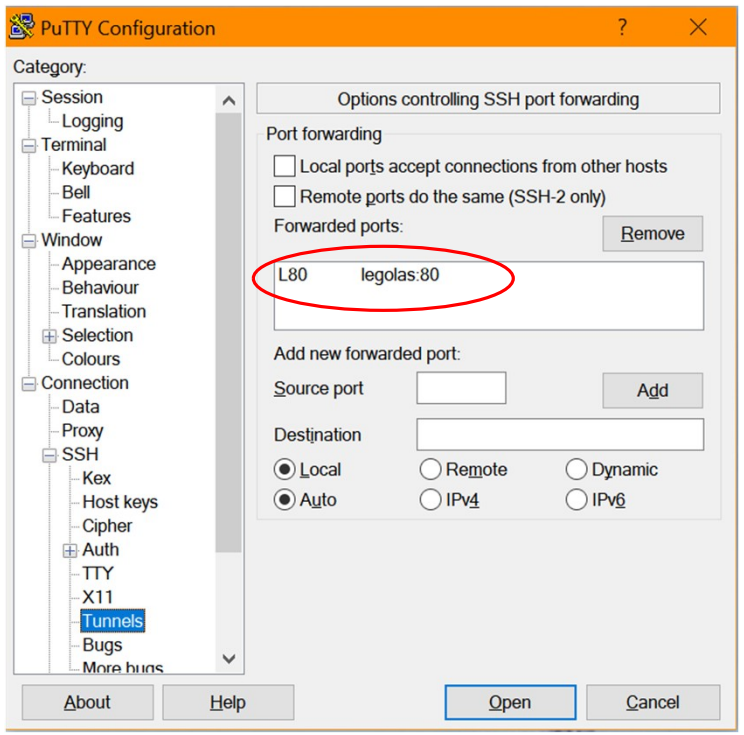
Fig. 17.11

Using channels to tunnel traffic

- SSH channels can be used to transparently tunnel traffic from other applications
- Mechanism is called port forwarding
- Many channels can be active at the same time
- SSH creates a local socket that an application can connect to:

```
% ssh legolas -L80:legolas:80
```

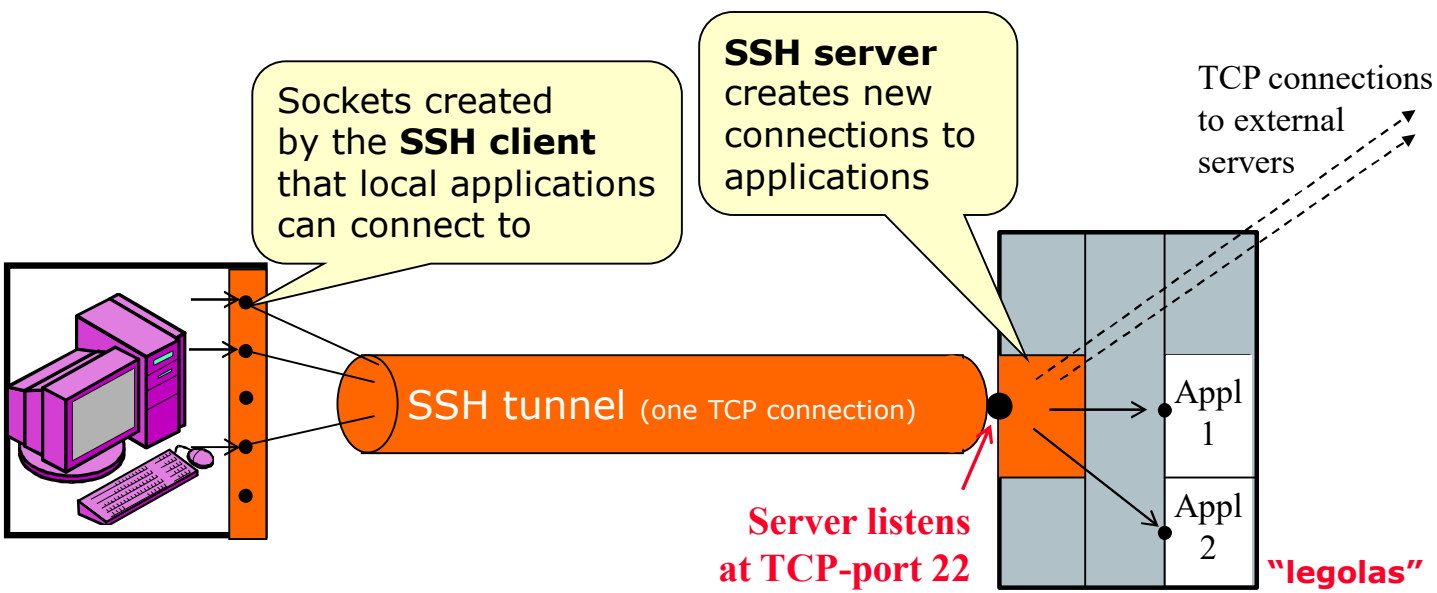
  - The SSH client starts listening to port 80 on the user's machine
  - Each time an application connects to it, the SSH server in the other end creates a connection to legolas, port 80
  - All data is forwarded through the secure encrypted tunnel between the SSH client and SSH server
  - A web browser can , for example, connect to the local SSH socket: http://localhost/index.html
  - The web browser will now get data from Legolas (port 80)



Port forwarding (tunneling)

A socket is the interface an application uses to communicate (similar to a telephone)
Example of a client connecting to a server:

```
Socket mysocket = getSocket(type = "TCP")
connect(mysocket, address = "1.2.3.4", port = "80")
send(mysocket, "Hello world!")
receive(mysocket, buffer)
```



Port forwarding (tunneling)

- Can be used to securely access remote services
  - Port 80 – Internal web servers
  - Port 139 – Windows file sharing
  - Port nnn – Any other TCP service
- Limitations
  - Port numbers must be known in advance by SSH client
  - Only one listener (server/service) per port (but multiple clients)
  - Applications must be configured to connect to the local machine: “myhost.chalmers.se”, “localhost” or 127.0.0.1 (not to www.remotewebserver.com)
- SSH also supports reverse tunnels
  - “Reverse connections” back to client:

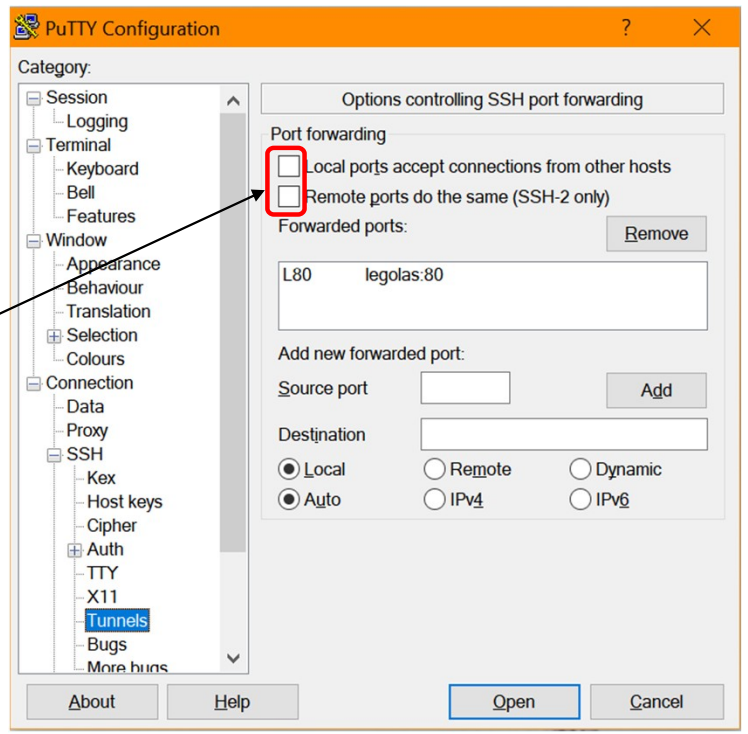
```
% ssh legolas -R80:localhost:80
```

  - Creates a listener on server side instead

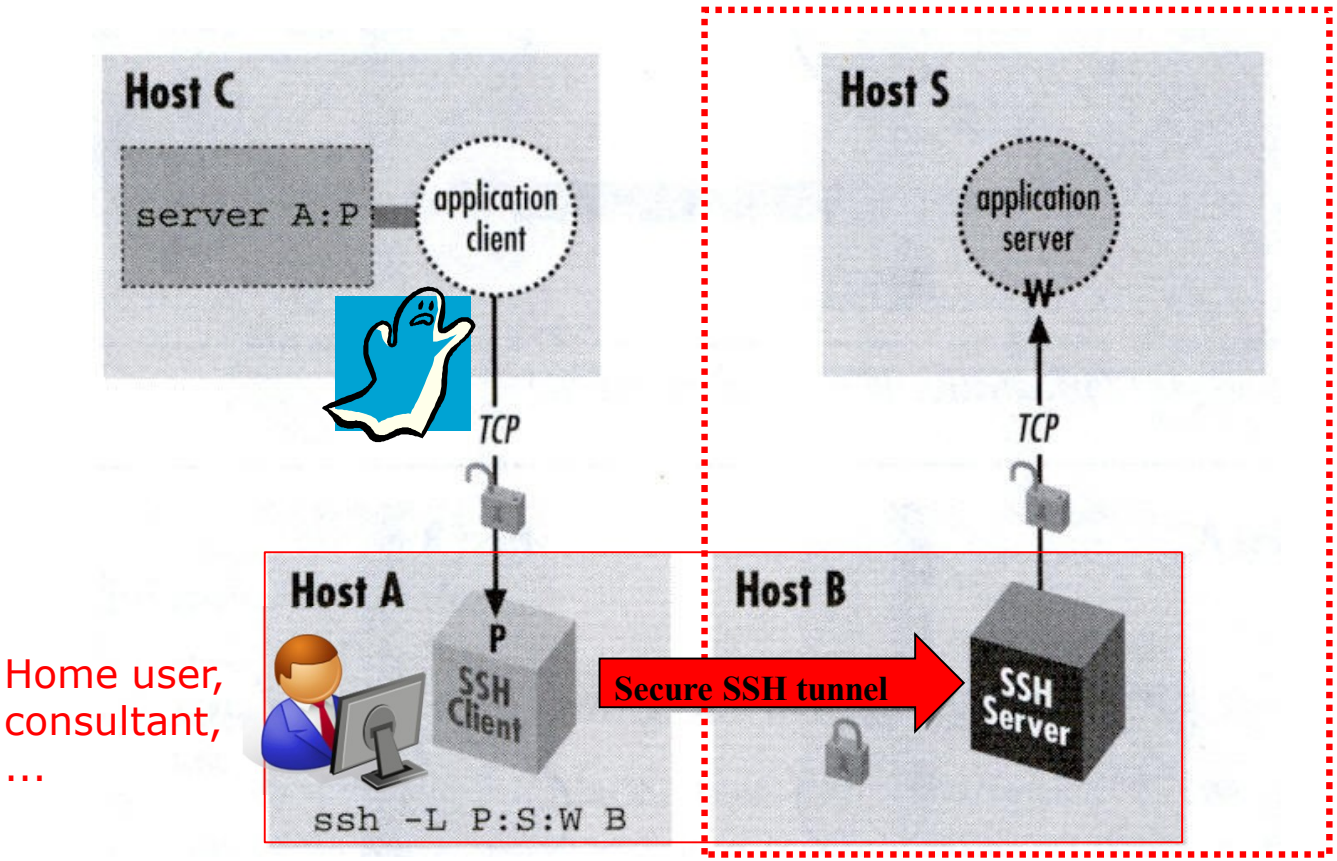
A short guide to SSH port forwarding
https://www.bitvise.com/ssh2

Security considerations

- Be careful on multi-user systems
  - All sockets on a machine are normally shared with all applications
  - Other users on the same computer can therefore connect to them
  - Problem on multi-user systems: Linux, Unix, MS terminal servers, Citrix
- Even remote users may be able to connect to the socket
  - May require a special option (-g flag) in some ssh implementations (use it!)
  - For example, if port 80 is accessible from the local netwok: http://my-IP-address/index.html becomes valid!
  - A personal firewall can also protect against external access



Remote users connecting to an SSH socket



Security changes in version 2

- Version 2 introduced 1997
- Replaced old linear CRC integrity check with hash !
  - WEP was standardized 1999…
- Added negotiation about MAC and public key algorithms, not just encryption algorithm
- Support for different keys and ciphers in each direction
- Session keys changed regularly
- Diffie-Hellman key agreement added for forward secrecy
- Allows connections without server keys using only Diffie-Hellman key negotiation (but opens up for MITM attacks)

Bugs exist in all software

Name	Description
CVE-2023-28531	ssh-add in OpenSSH before 9.3 adds smartcard keys to ssh-agent without the intended per-hop destination constraints. The earliest affected version is 8.9.
CVE-2023-25136	OpenSSH server (sshd) 9.1 introduced a double-free vulnerability during options.kex_algorithms handling. This is fixed in OpenSSH 9.2. The double free can be leveraged, by an unauthenticated remote attacker in the default configuration, to jump to any location in the sshd address space. One third-party report states "remote code execution is theoretically possible."
CVE-2022-31124	openssh_key_parser is an open source Python package providing utilities to parse and pack OpenSSH private and public key files. In versions prior to 0.0.6 if a field of a key is shorter than it is declared to be, the parser raises an error with a message containing the raw field value. An attacker able to modify the declared length of a key's sensitive field can thus expose the raw value of that field. Users are advised to upgrade to version 0.0.6, which no longer includes the raw field value in the error message. There are no known workarounds for this issue.
CVE-2021-41617	sshd in OpenSSH 6.2 through 8.x before 8.8, when certain non-default configurations are used, allows privilege escalation because supplemental groups are not initialized as expected. Helper programs for AuthorizedKeysCommand and AuthorizedPrincipalsCommand may run with privileges associated with group memberships of the sshd process, if the configuration specifies running the command as a different user.

Summary

- SSH is a good way to secure existing applications without rewriting them
  - SSL requires the application to be rewritten
- SSH is regarded as a very secure protocol
- Three protocols: Transport, User authentication and Connection protocol
- Guard your host keys
  - On Unix systems, host keys are stored ~/.ssh/known_hosts
  - SSH FAQ: ”If somebody has access to your home directory (~), then security is nonexistent“
- Can tunnel traffic (channels): terminal, X-Windows and port forwarding