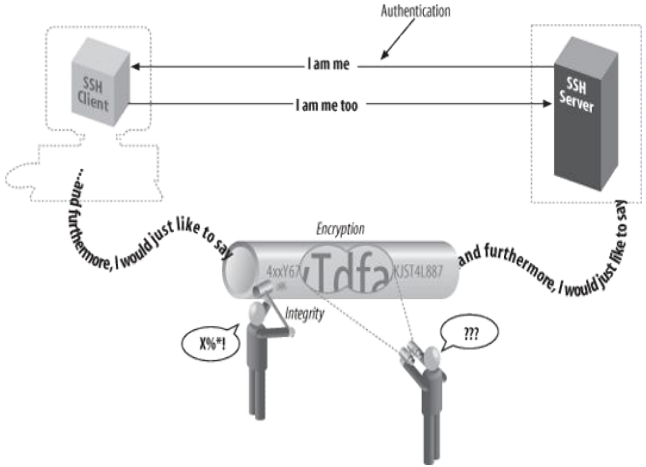
SECURE SHELL (SSH)

CYBERSECURITY

PROF. F. D'AMORE

THE SSH PROTOCOL

- Network protocol that allows data to be exchanged using a secure channel between two networked devices
- The channel is made secure through
  - One- or two-ways authentication
  - Data confidentiality (encryption)
  - Data integrity



SSH ALLOWS TO

login to a shell on a remote host (replacing Telnet and rlogin)

execute a single command on a remote host (replacing rsh)

securely transfer files

back up, copy and mirror files efficiently and securely (in combination with rsync)

forward or tunnel a TCP/IP port

arrange a full-fledged encrypted VPN

- OpenSSH only

forward X from a remote host (possible through multiple intermediate hosts)

browse the Web through an encrypted proxy connection with SSH clients that support the SOCKS protocol

securely mount a directory on a remote server as a filesystem on a local computer using SSHFS (SSH filesystem)

SSH OVERVIEW

SSH Communications Security (SCS)

- www.ssh.com
- Founded by Tatu Ylönen, writer of SSH-1
- SSH is a trademark of SCS
- https://www.ssh.com/ssh/

Open source version from OpenSSH

OpenSSH specification: https://www.openssh.com/specs.html

Long-running confusion and dispute over naming

see Wikipedia for a historical perspective
https://en.wikipedia.org/wiki/Secure_Shell#History_and_development

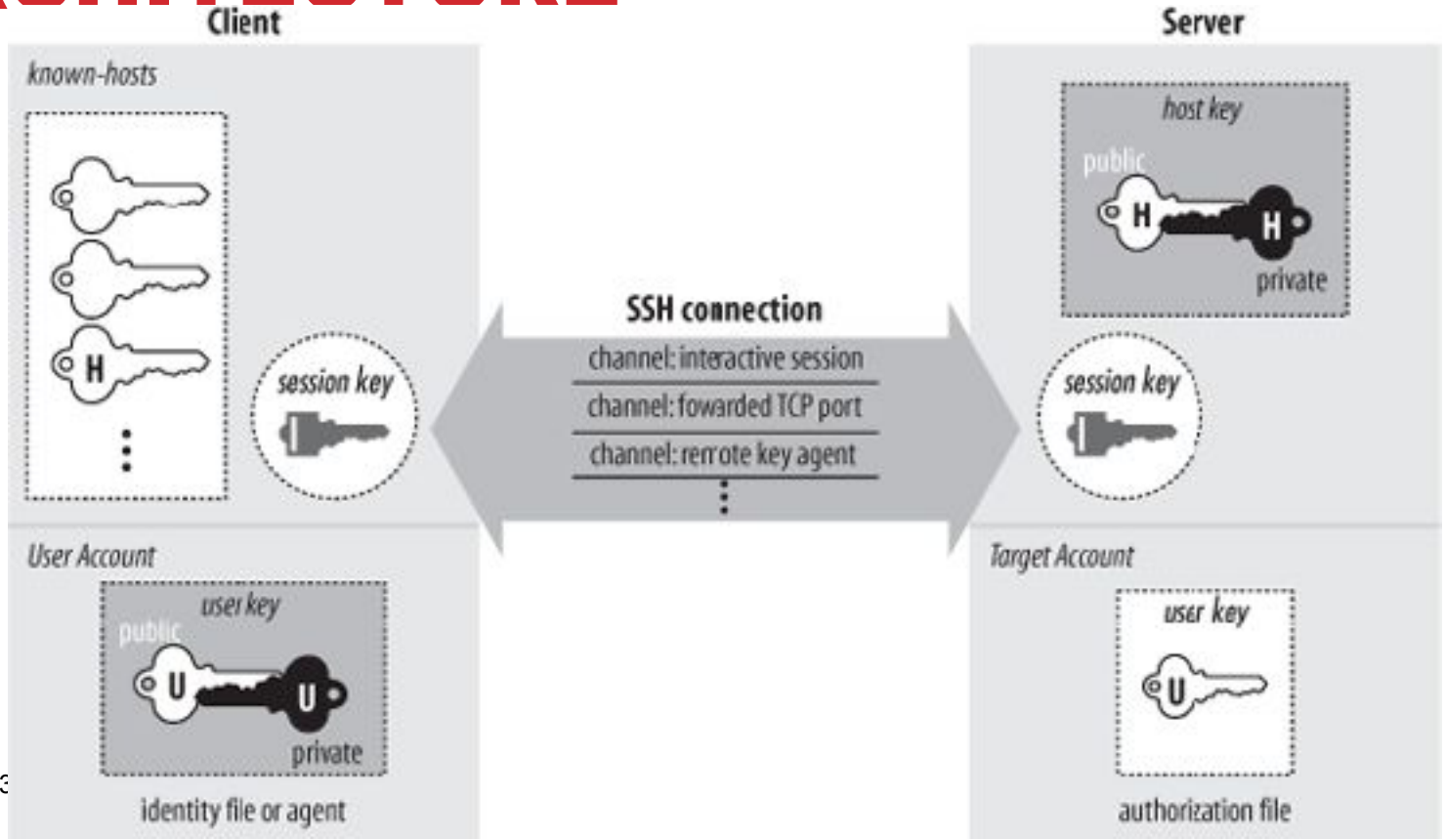
SSH VERSIONS

- **SSH1**
  - 1995 by Tatu Ylönen (Finnish researcher)
  - initial freeware, then proprietary
  - several vulnerabilities
- **SSH2**
  - 2006 (revised version)
- **OpenSSH**
  - open source project
  - started in 1999 from last free version of SSH (1.2.12)
  - supports both 1.x and 2.0 SSH versions
    - https://www.openssh.com/specs.html
  - widely used

SSH INTERNET STANDARD

- **basic set of RFCs**
  - RFC 4250, The Secure Shell (SSH) Protocol Assigned Numbers
  - RFC 4251, The Secure Shell (SSH) Protocol Architecture
  - RFC 4252, The Secure Shell (SSH) Authentication Protocol
  - RFC 4253, The Secure Shell (SSH) Transport Layer Protocol
  - RFC 4254, The Secure Shell (SSH) Connection Protocol
  - RFC 4255, Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints
  - RFC 4256, Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)
  - RFC 4335, The Secure Shell (SSH) Session Channel Break Extension
  - RFC 4344, The Secure Shell (SSH) Transport Layer Encryption Modes
  - RFC 4345, Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol
- **more recent RFCs (2006 – 2009) modify and expand the basic set of RFCs**

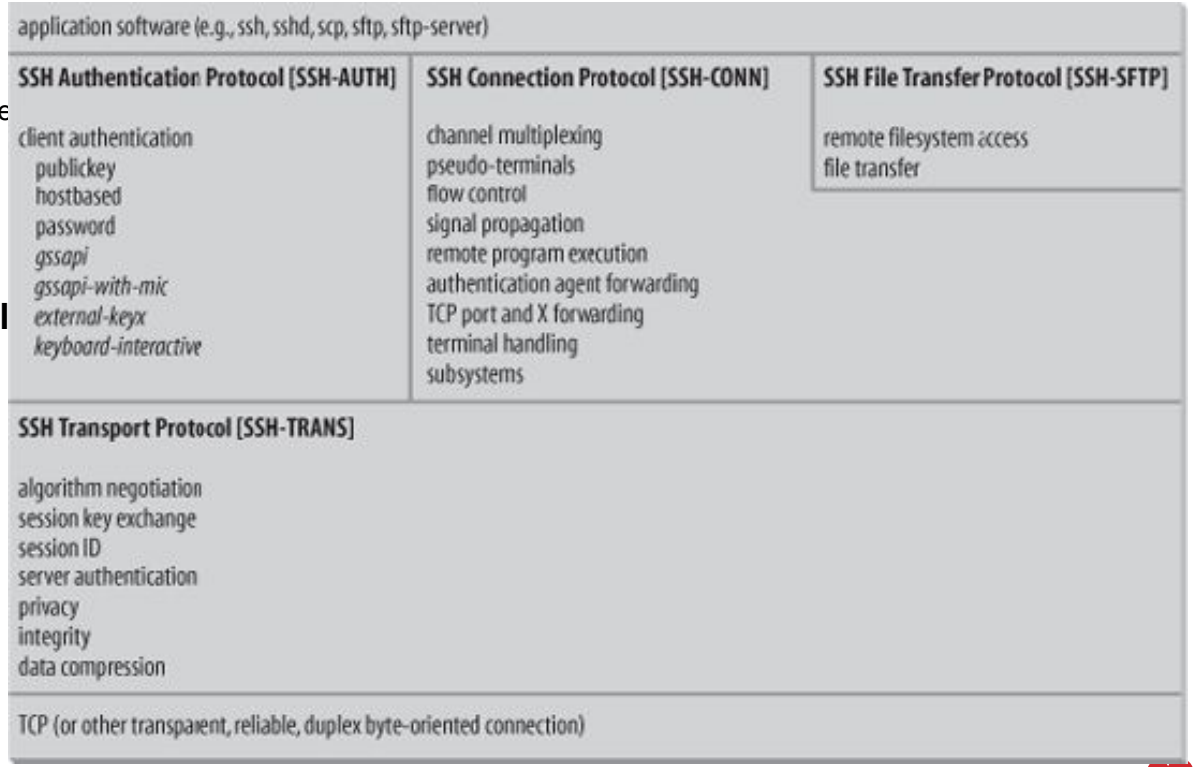
SSH BASIC ARCHITECTURE



SSH ARCHITECTURE

- **actually divided into four major pieces**
  - formally described as four separate protocols in different IETF documents
  - in principle independent of one another

- **SSH Transport Layer Protocol (SSH-TRANS)**
- **SSH Authentication Protocol (SSH-AUTH)**
- **SSH Connection Protocol (SSH-CONN)**
- **SSH File Transfer Protocol (SSH-SFTP)**



SSH SUMMARY

- **SSH-TRANS**
  - providing initial connection, server authentication, basic encryption and integrity services
  - after establishing SSH-TRANS connection, client has single, secure, full-duplex byte stream to authenticated peer
- **SSH-AUTH over the SSH-TRANS connection to authenticate client**
  - multiple authentication mechanisms may be used
  - it *requires* only one method: public key with the DSS algorithm
  - it further defines two more methods: password and hostbased
  - a number of other methods have been defined in various Internet-Drafts, and some of them have gained wide acceptance
- **after authentication SSH clients invoke the SSH-CONN protocol**
  - a variety of services over the single pipe provided by SSH-TRANS
  - support of multiple interactive and noninteractive sessions
    - multiplexing several streams (or *channels*) over the underlying connection
    - managing X, TCP, and agent forwarding
    - propagating signals across the connection (such as SIGINT, when a user types ^C to interrupt a process);
    - terminal handling, data compression, remote program execution etc.
- **finally, an application may use SSH-SFTP over an SSH-CONN channel to provide file-transfer and remote filesystem manipulation functions**

SSH-TRANS (RFC 4253)

Transport layer

- Handles initial key exchange, server authentication and sets up encryption, compression and integrity verification
- Exposes to upper layer an interface for sending/receiving plaintext packets with sizes of up to 32,768 bytes each (more can be allowed by the implementation)
- Transport layer also arranges for key re-exchange, usually after 1 GB of data or after 1 hour

SSH-AUTH (RFC 4252)

User authentication layer

- **Handles client authentication and provides a number of authentication methods**
- **Authentication is client-driven. Widely used user authentication methods include the following:**
  - password: straightforward password authentication; not implemented by all programs
  - publickey: public key-based authentication, usually supporting at least DSA or RSA keypairs, with other implementations also supporting X.509 certificates
  - keyboard-interactive (RFC 4256): server sends one or more prompts to enter information and the client displays them and sends back responses keyed-in by the user (e.g., one-time passwords)
  - GSSAPI: extensible scheme to perform SSH authentication using external mechanisms such as Kerberos 5 or NTLM, providing single sign on capability to SSH sessions
    - application programming interface for programs to access security services
    - usually implemented by commercial SSH implementations for use in organizations, though OpenSSH does have a working GSSAPI implementation

SSH-CONN (RFC 4254)

Connection layer

- **This layer defines the concept of channels, channel requests and global requests using which SSH services are provided**
- **A single SSH connection can host *multiple* channels simultaneously, each transferring data in both directions.**
- ***Channel requests* are used to relay out-of-band channel specific data, such as the changed size of a terminal window or the exit code of a server-side process.**
- **The SSH client requests a server-side port to be forwarded using a *global request*. Standard channel types include:**
  - shell for terminal shells, SFTP and exec requests (including SCP transfers)
  - direct-TCP/IP for client-to-server forwarded connections
  - forwarded-TCP/IP for server-to-client forwarded connections

SSH-2 ALGORITHMS

Key establishment through Diffie-Hellman key exchange

- Variety of groups supported

Server authentication via RSA or DSS signatures on nonces (and other fields)

HMAC for MAC algorithm

3DES, RC4, or AES

Pseudo-random function for key derivation

SSH SECURITY - UPDATE

On 6 July 2017 WikiLeaks confirmed that CIA had developed tools that can be installed on computers running Microsoft Windows or GNU/Linux operating systems to intercept SSH connections started by SSH clients on the compromised systems

Cfr. BothanSpy (Windows) and Gyrfalcon (linux)
https://www.wikileaks.org/vault7/#BothanSpy

(see Archive)

SSH (OPENSSH) BASIC CLIENT USAGE

ssh [options] <userid>@<hostaddress> [command]

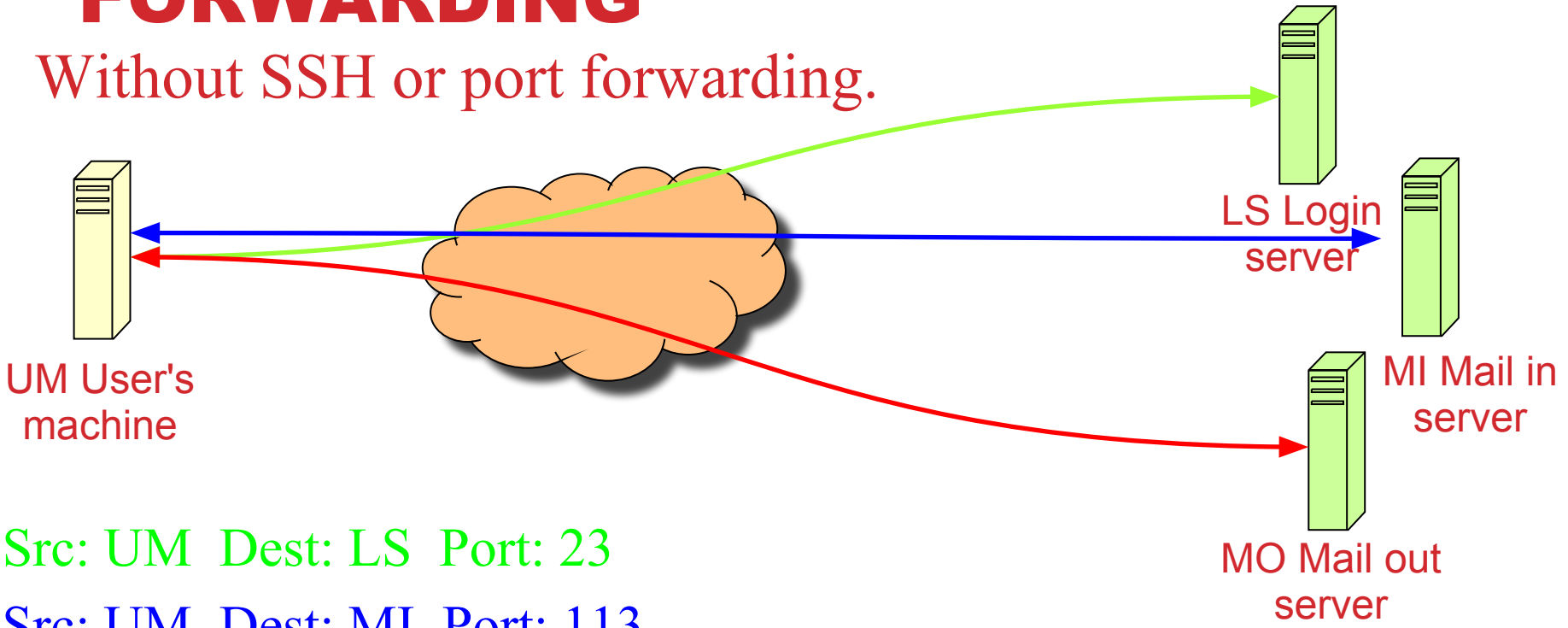
some options

- **-v [-v [-v]] enables verbose modes**
- **-C enables gzip compression**
- **-1,-2 forces version 1, 2**
- **-4,-6 forces addresses for IPv4, IPv6**

if just ssh <userid>@<hostaddress> then an interactive terminal session is established

SSH PORT FORWARDING

Without SSH or port forwarding.



Src: UM Dest: LS Port: 23

Src: UM Dest: MI Port: 113

Src: UM Dest: MO Port: 25

SSH PORT FORWARDING

Recall: TCP port number ‘identifies’ application.

SSH on local machine:

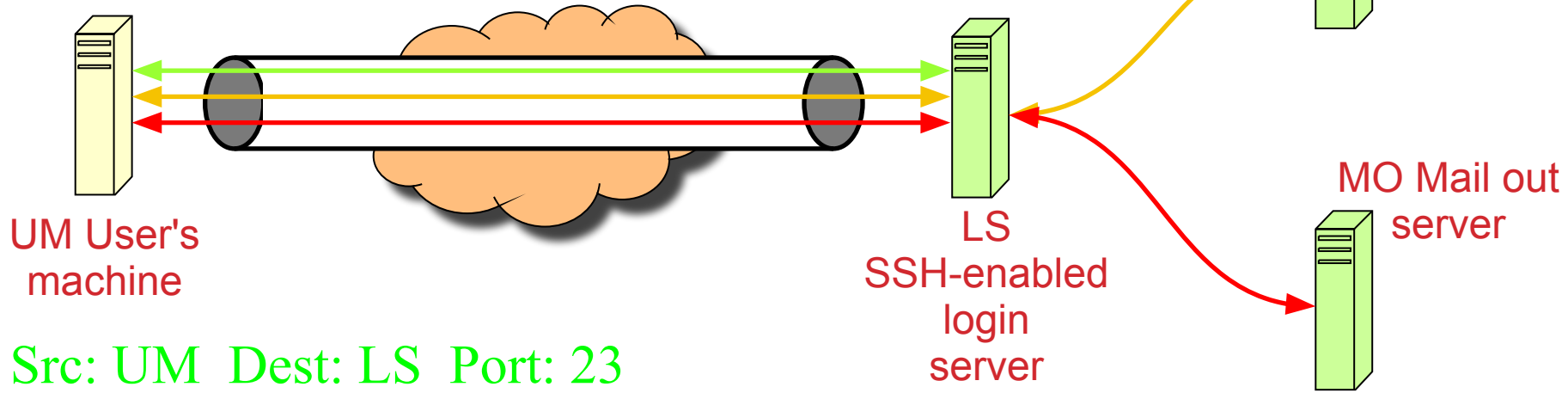
- Intercepts traffic bound for server.
- Translates standard TCP port numbers.
    - E.g. port 113 → port 5113.
- Sends packets to SSH-enabled server through SSH secure channel.

SSH-enabled server:

- Receives traffic.
- Re-translates port numbers.
    - E.g. port 5113 → port 113.
- Forwards traffic to appropriate server using internal network.

SSH PORT FORWARDING

With SSH and port forwarding.



Src: UM Dest: LS Port: 23

Src: UM Dest: MI Port: 113
Src: UM Dest: LS Port: 5113
Src: LS Dest: MI Port: 113

Src: UM Dest: MO Port: 25
Src: UM Dest: LS Port: 5025
Src: LS Dest: MO Port: 25

SSH PORT FORWARDING

how to connect to a “hidden” hostB, same network as the SSH server hostA

ssh -L portC:hostB:portB user@hostA

The given portC on client is to be forwarded to the given hostB and portB on the remote side, through hostA (hostB should be reachable from hostA). Whenever a connection is made to portC, the connection is forwarded over the secure channel, and a connection is made to portB of hostB from the remote machine (localhost). Port forwardings can also be specified in the configuration file.

Only the superuser can forward privileged ports.

Example

ssh -L 54321:10.0.2.92:22 damore@linuxserv.dis.uniroma1.it

makes it available at port 54321 of localhost port 22 of the (unreachable) host 10.0.2.92, providing a tunnel between (localhost, 54321) and (10.0.2.92, 22)

option -N disables execution of remote commands (forwarding only)

ssh -N -L 54321:10.0.2.92:22 …

option –f runs the ssh command in the background

ssh –f -N -L 54321:10.0.2.92:22 …

good for two-hops sshfs! (now: sshfs –p 54321 user@localhost: mountdir)

using -R instead of -L switches the roles of client and server

X11 FORWARDING

-X Enables X11 forwarding.

X11 forwarding should be enabled with caution. Users with the ability to bypass file permissions on the remote host (for the user's X authorization database) can access the local X11 display through the forwarded connection. An attacker may then be able to perform activities such as keystroke monitoring.

For this reason, X11 forwarding is subjected to X11 SECURITY extension restrictions by default. Please refer to the ssh -Y option and the ForwardX11Trusted directive in ssh_config(5) for more information.

-x Disables X11 forwarding.

-Y Enables trusted X11 forwarding. Trusted X11 forwardings are not subjected to the X11 SECURITY extension controls.