BSc in Telecommunications Engineering

TEL3214

Computer Communication Networks

Lecture 09
Applications

Eng Diarmuid O'Briain, CEng, CISSP

Department of Electrical and Computer Engineering,
College of Engineering, Design, Art and Technology,
Makerere University

Copyright © 2017 Diarmuid Ó Briain

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Table of Contents

Illustration Index

1. The Client-Server architecture

A Client-Server architecture involves two parties, a Server which is a daemon running on a computer with an open Transport Layer port. A Client requiring service connect to that port and depending on the application share information.

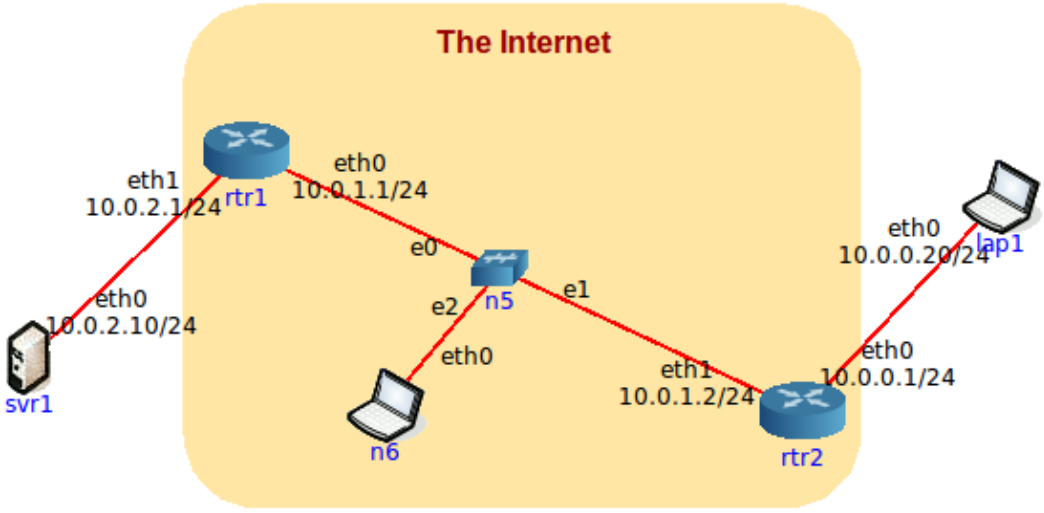

Illustration 1: Client-Server architecture

Illustration 1 Has two routers acting as the Internet. On ***svr1*** run the Server and on ***lap1*** the client.

svr1

```
root@svr1:/tmp/pycore.46202/svr1.conf# cd /home/nte/TEL-3214-exercises/code
root@svr1:/home/nte/TEL-3214-exercises/code# ./ccnd.py 5050

Welcome to the TEL-3214 Computer Communication Networks daemon (ccnd)
This program is designed to give students an understanding of the
client/server architecture through a simple TCP Socket.

Starting TCP Server on 5050

Connected by ('10.0.0.20', 49655)
Sending received data back to ccnc at 10.0.0.20
```

lap1

```
root@lap1:/tmp/pycore.46202/svr1.conf# cd /home/nte/TEL-3214-exercises/code
root@lap1:/home/nte/TEL-3214-exercises/code# ./ccnc.py 10.0.2.10 5050

Welcome to the TEL-3214 Computer Communication Networks client (ccnc)
This program is designed to give students an understanding of the
client/server architecture through a simple TCP Socket.

IP Addr:  10.0.2.10 Port# 5050

What message do you want to pass?: This is a TCP message

Sending message to server at 10.0.2.10 on port number: 5050

Received back:  This is a TCP message
```

2. TCP Flow

2.1 TCP Socket Connection

To understand the Client-Server architecture it is essential to understand the Transmission Control Protocol (TCP) flow.

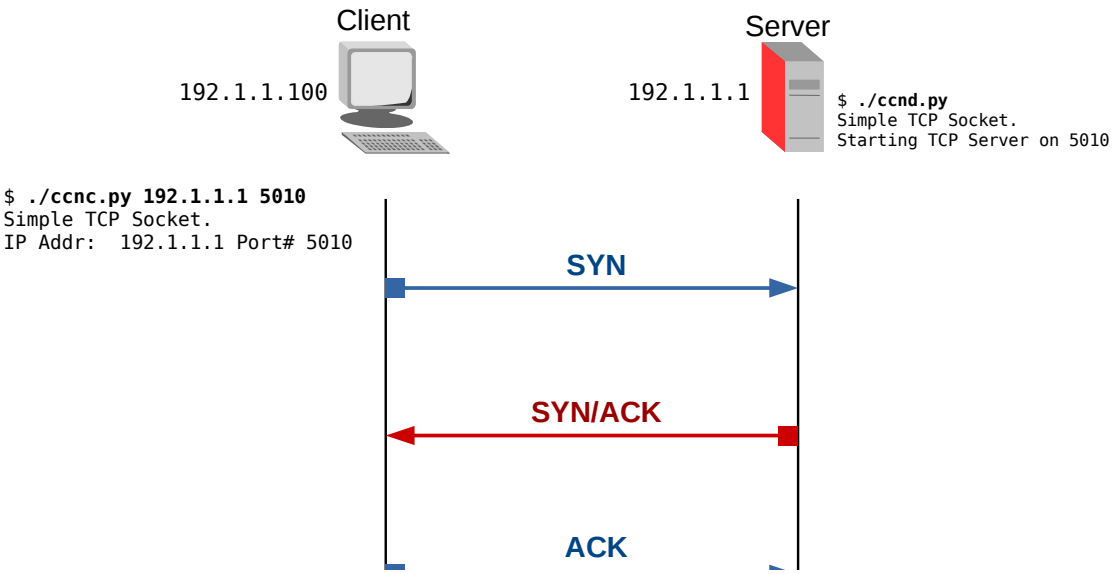

Illustration 2: TCP Socket connection

Initially a TCP SYNc (SYN) message is sent from from Client to Server.

```
Frame 1: 74 bytes on wire (592 bits) on interface 0
Ethernet II, Src: 08:00:27:9d:3b:da Dst: 34:e6:ad:05:51:bd
Internet Protocol Version 4, Src: 192.1.1.100, Dst: 192.1.1.1
Transmission Control Protocol, Src Port: 57567, Dst Port: 5560, Seq: 0, Len: 0
    Header length: 40 bytes
    Flags: 0x002 (SYN)
    Window size value: 29200
    Checksum: 0x1034
     Options: Max segment size, SACK permitted, Tstamps, No-Operation (NOP), Window
scale
```

This is responded to with a TCP SYN/ACKnowledge (ACK) from the Server to the Client.

```
Frame 2: 74 bytes on wire (592 bits) on interface 0
Ethernet II, Src: 34:e6:ad:05:51:bd, Dst: 08:00:27:9d:3b:da
Internet Protocol Version 4, Src: 192.1.1.1, Dst: 192.1.1.100
Transmission Control Protocol, Src Port: 5560, Dst Port: 57567, Seq: 0, Ack: 1, Len: 0
    Header length: 40 bytes
    Flags: 0x012 (SYN, ACK)
    Window size value: 28960
    Checksum: 0x52b2
     Options: Max segment size, SACK permitted, Tstamps, No-Operation (NOP), Window
scale
```

The Client sends the Server an ACK and the parameters of the connection are agreed.

```
Frame 3: 66 bytes on wire (528 bits) on interface 0
Ethernet II, Src: 08:00:27:9d:3b:da Dst: 34:e6:ad:05:51:bd
Internet Protocol Version 4, Src: 192.1.1.100, Dst: 192.1.1.1
Transmission Control Protocol, Src Port: 57567, Dst Port: 5560, Seq: 1, Ack: 1, Len: 0
    Header length: 32 bytes
    Flags: 0x010 (ACK)
    Window size value: 229
    Checksum: 0xf1b9
    Options: No-Operation (NOP), No-Operation (NOP), Tstamps
```

2.2 TCP Writes and Reads

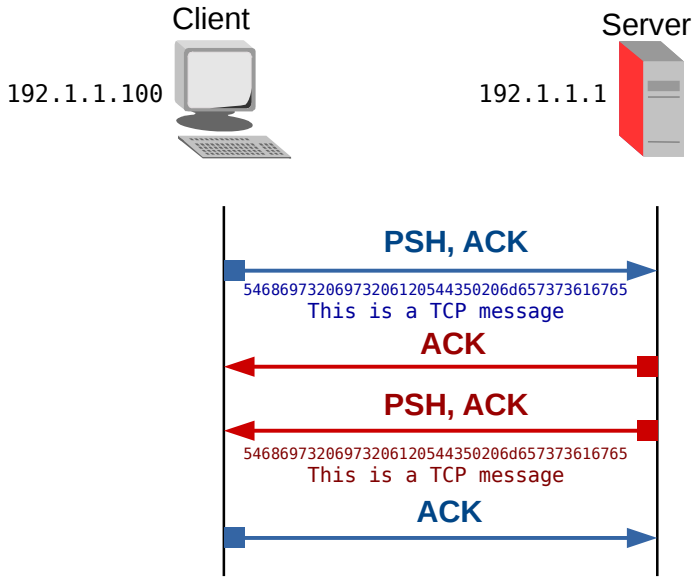

Illustration 3: TCP Writes and Reads

The Client sends the server the data as a PuSH (PSH), ACK message.

```
Frame 4: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface 0
Ethernet II, Src: 08:00:27:9d:3b:da Dst: 34:e6:ad:05:51:bd
Internet Protocol Version 4, Src: 192.1.1.100, Dst: 192.1.1.1
Transmission Control Protocol, Src Port: 57567, Dst Port: 5560, Seq: 1, Ack: 1, Len: 21
    Header length: 32 bytes
    Flags: 0x018 (PSH, ACK)
    Window size value: 229
    Checksum: 0xf372
    Options: No-Operation (NOP), No-Operation (NOP), Tstamps
Data (21 bytes)
0000  54 68 69 73 20 69 73 20 61 20 54 43 50 20 6d 65   This is a TCP me
0010  73 73 61 67 65                                    ssage
```

The Server responds with an ACK of the received data.

```
Frame 5: 66 bytes on wire (528 bits) on interface 0
Ethernet II, Src: 34:e6:ad:05:51:bd, Dst: 08:00:27:9d:3b:da
Internet Protocol Version 4, Src: 192.1.1.1, Dst: 192.1.1.100
Transmission Control Protocol, Src Port: 5560, Dst Port: 57567, Seq: 1, Ack: 22, Len: 0
    Source port: 5560
    Destination port: 57567
    Sequence number: 1
    Acknowledgment number: 22
    Header length: 32 bytes
    Flags: 0x010 (ACK)
    Window size value: 227
    Checksum: 0xf1a6
    Options: No-Operation (NOP), No-Operation (NOP), Tstamps
```

The Server then resends the data back to the Client as a PSH, ACK message.

```
Frame 6: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface 0
Ethernet II, Src: 34:e6:ad:05:51:bd, Dst: 08:00:27:9d:3b:da
Internet Protocol Version 4, Src: 192.1.1.1, Dst: 192.1.1.100
Transmission Control Protocol, Src Port: 5560, Dst Port: 57567, Seq: 1, Ack: 22, Len:
21
    Header length: 32 bytes
    Flags: 0x018 (PSH, ACK)
    Window size value: 227
    Checksum: 0xf35f
    Options: No-Operation (NOP), No-Operation (NOP), Tstamps
Data (21 bytes)
0000  54 68 69 73 20 69 73 20 61 20 54 43 50 20 6d 65   This is a TCP me
0010  73 73 61 67 65                                    ssage
```

The Client sends an ACK to confirm the received data.

```
Frame 7: 66 bytes on wire (528 bits) on interface 0
Ethernet II, Src: 08:00:27:9d:3b:da Dst: 34:e6:ad:05:51:bd
Internet Protocol Version 4, Src: 192.1.1.100, Dst: 192.1.1.1
Transmission Control Protocol, Src Port: 57567, Dst Port: 5560, Seq: 22, Ack: 22, Len:
0
    Header length: 32 bytes
    Flags: 0x011 (FIN, ACK)
    Window size value: 229
    Checksum: 0xf18e [validation disabled]
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Tstamps
```

2.3 TCP Close

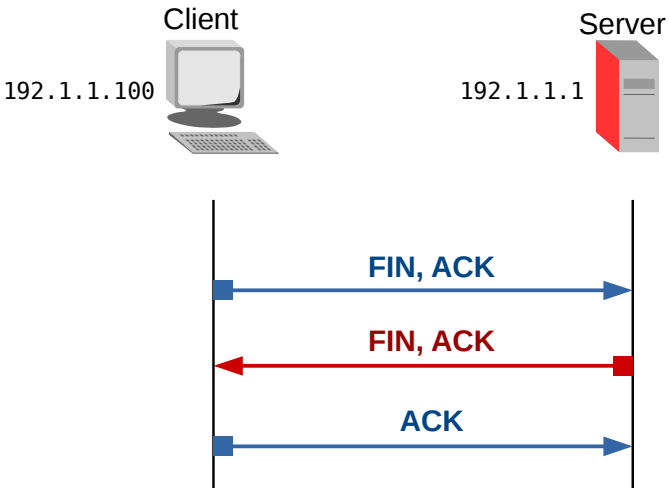

Illustration 4: TCP Close

To close the connection the Client sends a FIN, ACK message to the Server.

```
Internet Protocol Version 4, Src: 192.1.1.100, Dst: 192.1.1.1
Transmission Control Protocol, Src Port: 57567, Dst Port: 5560, Seq: 22, Ack: 22, Len:
0
    Header length: 32 bytes
    Flags: 0x011 (FIN, ACK)
    Window size value: 229
    Checksum: 0xf18e [validation disabled]
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Tstamps
```

This is responded to with a corresponding FIN, ACK message.

```
Internet Protocol Version 4, Src: 192.1.1.1, Dst: 192.1.1.100
Transmission Control Protocol, Src Port: 5560, Dst Port: 57567, Seq: 22, Ack: 23, Len:
0
    Header length: 32 bytes
    Flags: 0x011 (FIN, ACK)
    Window size value: 227
    Checksum: 0xf18f
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Tstamps
```

Finally the Client closes the connection by sending an ACK to the Server.

```
Frame 10: 66 bytes on wire (528 bits) on interface 0
Ethernet II, Src: 08:00:27:9d:3b:da Dst: 34:e6:ad:05:51:bd
Internet Protocol Version 4, Src: 192.1.1.100, Dst: 192.1.1.1
Transmission Control Protocol, Src Port: 57567, Dst Port: 5560, Seq: 23, Ack: 23, Len:
0
   Header length: 32 bytes
    Flags: 0x010 (ACK)
    Window size value: 229
    Checksum: 0xf18d [validation disabled]
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Tstamps
```

2.4 TCP Socket Daemon

The TCP Socket daemon is a demonstration tool

The program can be given a TCP port number at the beginning or can used the preprogrammed port of ***5010***.

```
try:
    tcp_port = sys.argv[1]
except:
    tcp_port = '5010'
```

It then validates the port number, i.e that it is between 5001 and 9999.

```
tcp_port = tcp_port.strip()

try:
    tcp_port = int(tcp_port)
except:
    print "\nPort number is not valid use the following format"
    print sys.argv[0], "[<port 5001-9999 | 5010>]"
    sys.exit()

if tcp_port < 5001 or tcp_port > 9999:
    print "\nPort number is not valid use the following format"
    print sys.argv[0], "[<port 5001-9999 | 5010>]"
    sys.exit()

print "Starting TCP Server on", tcp_port, "\n"
```

If all is OK the program then attempts to create a socket on the selected port number and remain open for all IP addresses on the system.

```
s = None
for sai in socket.getaddrinfo(ip_addr, tcp_port, socket.AF_UNSPEC, socket.SOCK_STREAM,
0, socket.AI_PASSIVE):
    addr_family, sock_type, proto, canonical_name, sock_addr = sai
    try:
        s = socket.socket(addr_family, sock_type, proto)
    except socket.error as msg:
        s = None
        continue
    try:
        s.bind(sock_addr)
        s.listen(1)
    except socket.error as msg:
        s.close()
        s = None
        continue
    break
if s is None:
    print 'could not open socket'
    sys.exit(1)
```

If an client binds to the socket, then a message is printed out telling of the connection. It receives until the client selects the enter key when it sends the data right back to the client. Once complete the connection is closed.

```
 conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    print "Sending received data back to ccnc at", addr[0], "\n"
    conn.send(data)
conn.close()
```

2.5 TCP Client

The client gets the IP address and Port number of the server from the initial running of the command.

```
try:
    ip_addr = sys.argv[1]
    tcp_port = sys.argv[2]
except:
    print "\nIncorrect format"
    print sys.argv[0], "<ip addr> <port 5001-9999 | 5010>"
    sys.exit()
```

It then validates the IP address.

```
def is_valid_ipv4_address(ip_addr):

    try:
        socket.inet_pton(socket.AF_INET, ip_addr)
    except socket.error:
        return False

    return True

def is_valid_ipv6_address(ip_addr):

    try:
        socket.inet_pton(socket.AF_INET6, ip_addr)
    except socket.error:
        return False

    return True

if  (is_valid_ipv4_address(ip_addr)  ==  False  and  is_valid_ipv6_address(ip_addr)  ==
False):
    print "\nIP address is not valid use the following format"
    print sys.argv[0], "<ip addr> <port 5001-9999 | 5010>"
    sys.exit()
```

After testing the IP address it validates the given Port number.

```
tcp_port = tcp_port.strip()

try:
    tcp_port = int(tcp_port)
except:
    print "\nPort number is not valid use the following format"
    print sys.argv[0], "<ip addr> <port 5001-9999 | 5010>"
    sys.exit()

if tcp_port < 5001 or tcp_port > 9999:
    print "\nPort number is not valid use the following format"
    print sys.argv[0], "<ip addr> <port 5001-9999 | 5010>"
    sys.exit()

print "IP Addr: ", ip_addr, "Port#", tcp_port, "\n"
```

The client then asks the user for a message.

```
message = raw_input('What message do you want to pass?: ')
```

It then attempts to bind with the Socket.

```
s = None
for sai in socket.getaddrinfo(ip_addr, tcp_port, socket.AF_UNSPEC, socket.SOCK_STREAM,
0, socket.AI_PASSIVE):
    addr_family, sock_type, proto, canonical_name, sock_addr = sai
    try:
        s = socket.socket(addr_family, sock_type, proto)
    except socket.error as msg:
        s = None
        continue
    try:
        s.connect(sock_addr)
    except socket.error as msg:
        s.close()
        s = None
        continue
    break
if s is None:
    print 'could not open socket'
    sys.exit(1)

If this is successful then the message is sent to the server.

print "\nSending message to server at", ip_addr, "on port number:", tcp_port
s.sendall(message)
```

A reply is received from the server, it is the message repeated back and is then printed to the Client terminal.

```
data = s.recv(1024)
s.close()

print '\nReceived back: ', data
```

3. Secure Shell (SSH)

Secure Shell (SSH) is the connectivity tool for remote login. It encrypts all traffic to eliminate eavesdropping, connection hijacking, and other attacks. In addition, SSH provides a large suite of secure tunnelling capabilities, several authentication methods, and sophisticated configuration options.

SSH consists of the following tools:

- Server side consists of:
  - SSH Daemon (**sshd**)
  - Secure File Transfer Protocol (**sftp-server**)
  - **ssh-agent** creates a socket and then checks the connections from ssh.
- Remote operations are carried out using:
  - **ssh** for remote login
  - **sftp**
  - Secure Copy (**scp**).
- Key management is achieved with
  - ssh-add
  - ssh-keysign
  - ssh-keyscan
  - ssh-keygen.

3.1 SSH

Looking at an example of a connection from the Client ***lap1*** to the Server ***svr1***. Note the user gets full shell access to the Server.

```
root@lap1:/root#  ssh nte@10.0.2.10
The authenticity of host '10.0.2.10 (10.0.2.10)' can't be established.
RSA key fingerprint is 53:c2:a2:55:1d:ed:de:16:2a:33:1a:42:7c:e9:4a:84.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.10' (RSA) to the list of known hosts.
nte@10.0.2.10's password: nte

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

nte@svr1:~$ id
uid=1001(nte) gid=1001(nte) groups=1001(nte),27(sudo),128(wireshark)

nte@svr1:~$ ip -4 addr list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
31: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UP group default qlen 1000
    inet 10.0.2.10/24 scope global eth0
       valid_lft forever preferred_lft forever
```

3.2 SFTP

SFTP is a really useful tool for securely transferring files to and from Servers. In the example a 10 MB is created on the Client ***lap1*** and is transferred to the Server ***svr1***.

lap1

Create a 10 MB file as a test file and confirm its existence and size.

```
root@lap1:/tmp/pycore.54569/lap1.conf# dd if=/dev/zero of=10mb_file_test_file
bs=10485760 count=1
1+0 records in
1+0 records out
10485760 bytes (10 MB) copied, 0.0223468 s, 469 MB/s
```

Creates a file full of zeros.

```
root@lap1:/tmp/pycore.54569/lap1.conf# ls -la 10mb_file_test_file
-rw-rw-rw- 1 root root 10485760 Mar  9 19:48 10mb_file_test_file
```

Connect to the Server ***svr1***, review the available commands.

```
root@lap1:/tmp/pycore.54569/lap1.conf# sftp root@10.0.2.10
root@10.0.2.10's password:
The authenticity of host '10.0.2.10 (10.0.2.10)' can't be established.
RSA key fingerprint is 59:f3:a4:7e:f6:25:04:9f:1c:b4:6f:84:50:1a:b2:70.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.10' (RSA) to the list of known hosts.
root@10.0.2.10's password: root
Connected to 10.0.2.10.
```

```
sftp> ?
Available commands:
bye                                Quit sftp
cd path                            Change remote directory to 'path'
chgrp grp path                     Change group of file 'path' to 'grp'
chmod mode path                    Change permissions of file 'path' to 'mode'
chown own path                     Change owner of file 'path' to 'own'
df [-hi] [path]                    Display statistics for current directory or
                                   filesystem containing 'path'
exit                               Quit sftp
get [-Ppr] remote [local]          Download file
reget remote [local]               Resume download file
reput [local] remote               Resume upload file
help                               Display this help text
lcd path                           Change local directory to 'path'
lls [ls-options [path]]            Display local directory listing
lmkdir path                        Create local directory
ln [-s] oldpath newpath            Link remote file (-s for symlink)
lpwd                               Print local working directory
ls [-1afhlnrSt] [path]             Display remote directory listing
lumask umask                       Set local umask to 'umask'
mget                               As   per   get   but   wildcards   for   multiple
downloads
mkdir path                         Create remote directory
mput                               As per put but wildcards for multiple uploads
progress                           Toggle display of progress meter
put [-Ppr] local [remote]          Upload file
pwd                                Display remote working directory
quit                               Quit sftp
rename oldpath newpath             Rename remote file
rm path                            Delete remote file
rmdir path                         Remove remote directory
symlink oldpath newpath            Symlink remote file
version                            Show SFTP version
!command                           Execute 'command' in local shell
!                                  Escape to local shell
?                                  Synonym for help
```

Change to the working directory of svr1 and create a directory there.

```
sftp> cd /tmp/pycore.54569/svr1.conf
sftp> mkdir Big_File
sftp> cd Big_File
sftp> pwd
Remote working directory: /tmp/pycore.54569/svr1.conf/Big_File
```

Check the local directory and file.

```
sftp> lpwd
Local working directory: /tmp/pycore.54569/lap1.conf

sftp> lls
10mb_file_test_file  defaultroute.sh  var.log  var.run
```

Transfer the file to the new directory on ***svr1***.

```
sftp> put 10mb_file_test_file
Uploading 10mb_file_test_file to
/tmp/pycore.54569/svr1.conf/Big_File/10mb_file_test_file
10mb_file_test_file                           100%   10MB  10.0MB/s   00:00
```

```
sftp> ls
10mb_file_test_file

sftp> exit
```

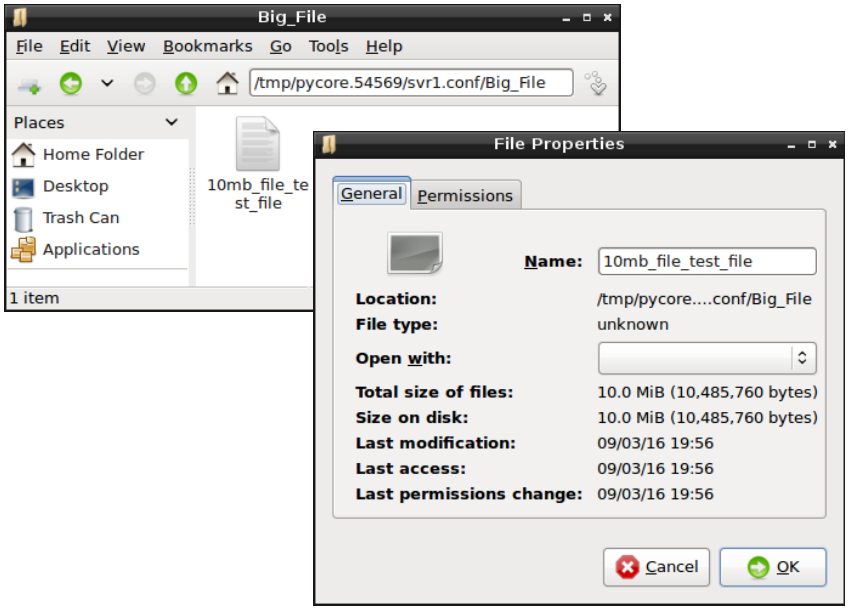

Illustration 5: File transferred to user root on 10.0.2.10

4. Archiving and compressing files and directories

4.1 Tape Archive (TAR) archiving

GNU **tar** is the GNU version of the **tar** archiving utility. Originally that was the **t**ape **ar**chive. It is useful to **tar up** a directory and all the directories and file therein as a single file, the tar archive file. The GNU tar program can do this. The resultant file is generally called a **tarball**. Using the file **10mb_file_test_file** in the **Big_File** directory on ***svr1***.

```
root@svr1:/tmp/pycore.54569/svr1.conf# ls Big_File/
10mb_file_test_file

root@svr1:/tmp/pycore.54569/svr1.conf# tar -cf Big_File.tar Big_File

root@svr1:/tmp/pycore.54569/svr1.conf# file Big_File.tar
Big_File.tar: POSIX tar archive (GNU)
```

Review a tar archive with the **-t** or **--list** option to see a table of contents for the archive.

```
root@svr1:/tmp/pycore.54569/svr1.conf# tar -tf Big_File.tar
Big_File/
Big_File/10mb_file_test_file
```

Remove the original directory.

```
root@svr1:/tmp/pycore.54569/svr1.conf# rm -r Big_File
```

Extract the archive and confirm the directory is recovered.

```
root@svr1:/tmp/pycore.54569/svr1.conf# tar -xf Big_File.tar

root@svr1:/tmp/pycore.54569/svr1.conf# ls Big_File
10mb_file_test_file
```

4.2 Compression

4.2.1 GNU ZIP (GZIP)

The tar archive can be compressed to reduce file size. For example **gzip** which reduces the size of files using Lempel-Ziv coding (LZ77) can be applied to the tarball. tar has the ability to incorporate compression functions as well as archiving and perform both functions with the same command.

```
root@svr1:/tmp/pycore.54569/svr1.conf# gzip Big_File.tar

root@svr1:/tmp/pycore.54569/svr1.conf# ls -l | grep Big_File
drwxr-xr-x 2 root     root  4096 Mar  9 19:56 Big_File
-rw-rw-rw- 1 root     root 10346 Mar  9 20:08 Big_File.tar.gz
```

To reverse this process use the **gunzip** command.

```
root@svr1:/tmp/pycore.54569/svr1.conf# gunzip Big_File.tar.gz

root@svr1:/tmp/pycore.54569/svr1.conf# ls -l | grep Big_File
drwxr-xr-x 2 root     root     4096 Mar  9 19:56 Big_File
-rw-rw-rw- 1 root     root 10496000 Mar  9 20:08 Big_File.tar
```

4.2.2 BZIP2

An alternative approach is to use the **bzip2** utility which uses the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. **bzip2** compression is generally considerably better that the more conventional LZ77/LZ78-based compressors.

```
root@svr1:/tmp/pycore.54569/svr1.conf# bzip2 Big_File.tar

root@svr1:/tmp/pycore.54569/svr1.conf# ls -l | grep Big_File
drwxr-xr-x 2 root     root 4096 Mar  9 19:56 Big_File
-rw-rw-rw- 1 root     root  180 Mar  9 20:08 Big_File.tar.bz2
```

The reverse process is similar to what has been seen for **gunzip**.

```
root@svr1:/tmp/pycore.54569/svr1.conf# bunzip2 Big_File.tar.bz2

root@svr1:/tmp/pycore.54569/svr1.conf# ls -l | grep Big_File
drwxr-xr-x 2 root     root     4096 Mar  9 19:56 Big_File
-rw-rw-rw- 1 root     root 10496000 Mar  9 20:08 Big_File.tar
```

4.2.3 XZ

Another approach is to use the **xz** utility which is a lossless data compression program and file format based on the Lempel–Ziv–Markov chain algorithm (**LZMA**).

```
root@svr1:/tmp/pycore.54569/svr1.conf# xz Big_File.tar

root@svr1:/tmp/pycore.54569/svr1.conf# ls -l | grep Big_File
drwxr-xr-x 2 root     root 4096 Mar  9 19:56 Big_File
-rw-rw-rw- 1 root     root 1764 Mar  9 20:08 Big_File.tar.xz
```

The reverse process is similar to what has been seen for **gunzip**.

```
root@svr1:/tmp/pycore.54569/svr1.conf# unxz Big_File.tar.xz

root@svr1:/tmp/pycore.54569/svr1.conf# ls -l | grep Big_File
drwxr-xr-x 2 root     root     4096 Mar  9 19:56 Big_File
-rw-rw-rw- 1 root     root 10496000 Mar  9 20:08 Big_File.tar
```

4.2.4 TAR/GZIP/BZ/XZ

Fortunately the **tar** utility offers the ability to both archive and compress in one operation, here is an example using **gzip**. Note the file extension for a gzipped archives is either **.tar.gz** or simply **.tgz**. The **z** switch in the command instructs that the directory be archived and gzipped.

```
root@svr1:/tmp/pycore.54569/svr1.conf# tar -czvf Big_File.tar.gz Big_File
Big_File/
Big_File/10mb_file_test_file

root@svr1:/tmp/pycore.54569/svr1.conf# file Big_File.tar.gz
Big_File.tar.gz: gzip compressed data, last modified: Wed Mar  9 20:23:18
2016, from Unix
```

A similar process can be achieved for **bzip2**, the end extension being **.tar.bz2** or **.tbz2** by convention. The **j** switch is used to archive and **bzip2**.

```
root@svr1:/tmp/pycore.54569/svr1.conf# tar -cjvf Big_File.tar.bz2 Big_File
Big_File/
Big_File/10mb_file_test_file

root@svr1:/tmp/pycore.54569/svr1.conf# file Big_File.tar.bz2
Big_File.tar.bz2: bzip2 compressed data, block size = 900k
```

To compress with **xz** can also be achieved within the **tar** utility, the end extension being **.tar.xz** or **.txz** with the **.J** switch.

```
root@svr1:/tmp/pycore.54569/svr1.conf# tar -cJvf Big_File.tar.xz Big_File
Big_File/
Big_File/10mb_file_test_file

root@svr1:/tmp/pycore.54569/svr1.conf# file Big_File.tar.xz
Big_File.tar.xz: XZ compressed data
```

4.2.5 ZIP

zip/unzip is a compression and file packaging utility found in Unix, VMS, MSDOS, OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and Acorn RISC OS. It is analogous to a combination of the Unix commands tar and compress and is compatible with Phil Katz's ZIP for Windows systems called PKZIP or WinZIP. ZIP can also archive a group of files directly. Here **zip** recursively archives and compresses the directory Big_File and its contents.

```
root@svr1:/tmp/pycore.54569/svr1.conf# zip -r Big_File.zip Big_File
updating: Big_File/ (stored 0%)
updating: Big_File/10mb_file_test_file (deflated 100%)
```

To recover the files from the compressed archive use **unzip**.

```
root@svr1:/tmp/pycore.54569/svr1.conf# unzip Big_File.zip
Archive:  Big_File.zip
   creating: Big_File/
  inflating: Big_File/10mb_file_test_file
```

4.2.6 Comparing compression tools

Comparing the relative sizes of the archive and the three compressed versions. When the requirement is very fast compression, the **gzip** was the best option, it has also very small memory footprint, making it ideal for systems with limited memory. **bzip2** creates about 15% smaller files than **gzip** on average however it compresses at a slower rate than **gzip**. For decompression a similar picture emerges with **gzip** the fastest. **bzip2** is a lot slower taking four to twelve times more time to decompress than **gzip**. The newer **xz** is now showing to be slightly better performance in terms of compression than the others. **zip** exhibits similar performance to **gzip**.

```
root@svr1:/tmp/pycore.54569/svr1.conf# ls -la | grep Big_File.
-rw-rw-rw-  1 root     root 10496000 Mar  9 20:08 Big_File.tar
-rw-rw-rw-  1 root     root      180 Mar  9 20:24 Big_File.tar.bz2
-rw-rw-rw-  1 root     root    10333 Mar  9 20:23 Big_File.tar.gz
-rw-rw-rw-  1 root     root     1752 Mar  9 20:26 Big_File.tar.xz
-rw-rw-rw-  1 root     root    10542 Mar  9 20:28 Big_File.zip
```

Download the **Big_File.tar** files from ***svr1*** to ***lap1***. Note sizes and download times.

```
root@lap1:/tmp/pycore.54569/lap1.conf# sftp root@10.0.2.10
root@10.0.2.10's password: root
Connected to 10.0.2.10.

sftp> mget Big_File.*
Fetching Big_File.tar to /tmp/pycore.54569/svr1.conf/Big_File.tar
Big_File.tar                                  100%   10MB  10.0MB/s   00:00
Fetching Big_File.tar.bz2 to /tmp/pycore.54569/svr1.conf/Big_File.tar.bz2
Big_File.tar.bz2                              100% 1764     1.7KB/s   00:00
Fetching Big_File.tar.gz to /tmp/pycore.54569/svr1.conf/Big_File.tar.gz
Big_File.tar.gz                               100%   10KB  10.1KB/s   00:00
Fetching Big_File.tar.xz to /tmp/pycore.54569/svr1.conf/Big_File.tar.xz
Big_File.tar.xz                               100% 1764     1.7KB/s   00:00
Fetching Big_File.zip to /tmp/pycore.54569/svr1.conf/Big_File.zip
Big_File.zip                                  100%   10KB  10.3KB/s   00:00
```

5. Hyper Text Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. It's most common application is the mechanism of data communication for the World Wide Web.

Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

Th coordination for HTTP standards and development is the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C). It is standardised as Requests for Comments (RFC). The first definition of HTTP/1.1, the version of HTTP in common use, occurred in RFC 2068 in 1997. This RFC was superseded by RFC 2616 in 1999.

HTTP functions as a request–response protocol in the client–server computing model. A typical client is a web browser a HTTP daemon (apache2 for example) running on a hosting server.

The client submits an HTTP GET request message to the server. The server returns a HTTP 200/OK response message to the client. The 200/OK message contains completion status information about the request as well as the requested content in the message body.

5.1.1 HTTP User Agent (UA)

A web browser is an example of a user agent (UA). However it is not the only user agent type. Other types include web crawlers, the indexing software used by search providers, mobile apps to name just a few.

5.1.2 HTTP example

For the NTE hosts there is only shell access. However GNU/Linux has a general purpose distributed information browser for the World Wide Web called lynx the operated on the shell without resorting to a graphical interface.

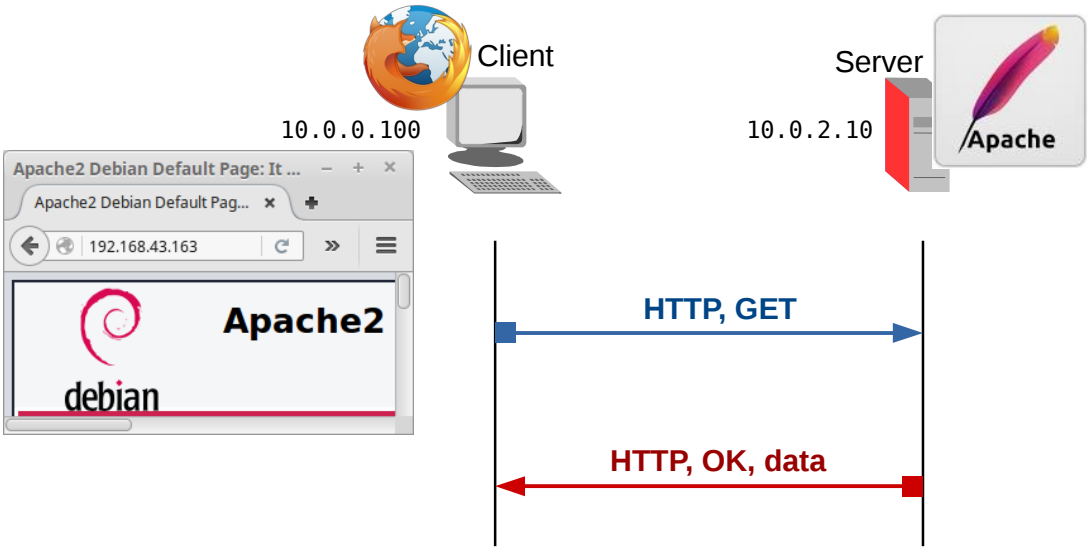

Illustration 6: HTTP message flow

Connect from ***lap1*** to ***svr1*** using ***lynx***. The default ***apache2*** page is presented to the browser.

```
root@lap1:/tmp/pycore.46202/lap1.conf# lynx 10.0.2.10
Looking up  '10.0.2.10' first

  Your Terminal type is unknown!
  Enter a terminal type: [vt100] <CR>


                                svr1 web server

   This is the default web page for this server.

   The web server software is running but no content has been added, yet.

     Eth0 - ['10.0.2.10/24', '2001:2::10/64']
```

5.1.3 HTTP on the wire

Now look at the traffic on the Internet corresponding to this connection. There is an initial **GET** / **HTTP** from ***lap1*** at 10.0.0.20 to ***svr1*** at 10.0.2.10.

```
Internet Protocol Version 4, Src: 10.0.0.20, Dst: 10.0.2.10
Transmission Control Protocol, Src Port: 54117, Dst Port: 80, Seq: 1, Ack: 1, Len: 253
Hypertext Transfer Protocol
 GET / HTTP/1.0\r\n
 Host: 10.0.2.10\r\n
       Accept: text/html, text/plain, text/sgml, text/css, app/xhtml+xml,
*/*;q=0.01\r\n
   Accept-Encoding: gzip, compress, bzip2\r\n
   Accept-Language: en\r\n
   User-Agent: Lynx/2.8.9dev.1 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/3.3.8\r\n
   \r\n
```

The webserver responds with a HTTP/1.1 200 OK message containing the requested webpage.

```
Internet Protocol Version 4, Src: 10.0.2.10, Dst: 10.0.0.20
Transmission Control Protocol, Src Port: 80, Dst Port: 54117, Seq: 1, Ack: 254, Len:
500
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Fri, 26 Feb 2016 18:33:12 GMT\r\n
    Server: Apache/2.4.10 (Debian)\r\n
    Last-Modified: Fri, 26 Feb 2016 18:32:10 GMT\r\n
    ETag: "115-52cb08464d460"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 277\r\n
    Connection: close\r\n
    \r\n
    Data (277 bytes)

0000  3c 68 74 6d 6c 3e 3c 62 6f 64 79 3e 3c 21 2d 2d   <html><body><!--
0010  20 67 65 6e 65 72 61 74 65 64 20 62 79 20 75 74    generated by ut
0020  69 6c 69 74 79 2e 70 79 3a 48 74 74 70 53 65 72   ility.py:HttpSer
0030  76 69 63 65 20 2d 2d 3e 0a 3c 68 31 3e 73 76 72   vice -->.<h1>svr
0040  31 20 77 65 62 20 73 65 72 76 65 72 3c 2f 68 31   1 web server</h1
0050  3e 0a 3c 70 3e 54 68 69 73 20 69 73 20 74 68 65   >.<p>This is the
0060  20 64 65 66 61 75 6c 74 20 77 65 62 20 70 61 67    default web pag
0070  65 20 66 6f 72 20 74 68 69 73 20 73 65 72 76 65   e for this serve
0080  72 2e 3c 2f 70 3e 0a 3c 70 3e 54 68 65 20 77 65   r.</p>.<p>The we
0090  62 20 73 65 72 76 65 72 20 73 6f 66 74 77 61 72   b server softwar
00a0  65 20 69 73 20 72 75 6e 6e 69 6e 67 20 62 75 74   e is running but
00b0  20 6e 6f 20 63 6f 6e 74 65 6e 74 20 68 61 73 20    no content has
00c0  62 65 65 6e 20 61 64 64 65 64 2c 20 79 65 74 2e   been added, yet.
00d0  3c 2f 70 3e 0a 3c 6c 69 3e 65 74 68 30 20 2d 20   </p>.<li>eth0 -
00e0  5b 27 31 30 2e 30 2e 32 2e 31 30 2f 32 34 27 2c   ['10.0.2.10/24',
00f0  20 27 32 30 30 31 3a 32 3a 3a 31 30 2f 36 34 27    '2001:2::10/64'
0100  5d 3c 2f 6c 69 3e 0a 3c 2f 62 6f 64 79 3e 3c 2f   ]</li>.</body></
0110  68 74 6d 6c 3e                                    html>
```

6. Asymmetric Key Cryptography

Asymmetric key cryptography or public key cryptography is a relatively new cryptographic approach where the use of asymmetric key algorithms instead of or in addition to symmetric key algorithms is used as an enhancement to security.

Public key cryptography unlike symmetric key algorithms does not require a secure initial exchange of one or more secret keys to both sender and receiver. Instead a mathematically related key pair is created, a secret private key and a public key the latter which is published. These keys allow protection of the authenticity of a message by creating a digital signature of a message using the private key, which can be validated using the public key. It also allows for the protection of the messages confidentiality and integrity, by public key encryption, encrypting the message using the public key, which can only be decrypted using the private key.

Public key cryptography is employed by many cryptographic algorithms and cryptosystems. It is used in standards such as Transport Layer Security (TLS)/Secure Sockets Layer (SSL), Pretty Good Privacy (PGP), and GNU Privacy Guard (GnuPG).

6.1 Key pairs

The generation of key pairs requires the use of intractable problems called trapdoor functions which are functions that are easy to compute in one direction, yet believed to be difficult to compute in the opposite direction without special information, called the "trapdoor".

An intractable problem is a problem for which there is no efficient means of solving. These aren't necessarily problems for which there is no solution. Instead, these are problems that take too long to analyse all the options. The public key cryptographic intractable problems used to date are based either on factoring prime numbers or discrete logarithms.

Looking at an example:

Take a prime number $m = 29$ as the modulus (public key). The primitive roots of 29 are: 2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27.

So taking and a base $b = 10$ (the trapdoor)

Alice chooses a secret $y = 8$ (Private Key).

Alice sends Bob $w = by \bmod m = 10^8 \bmod 29 = 25$

Bob chooses a secret $z = 11$ (Private Key).

Bob sends Alice $x = bz \bmod m = 10^{11} \bmod 29 = 2$

Alice computes $s = xy \bmod m = 2^8 \bmod 29 = 24$

Bob computes $s = wz \bmod m = 25^{11} \bmod 29 = 24$

Alice and Bob now share a secret, in this case 24 without it being transferred across the transmission path and without either Alice or Bob sharing their private keys.

6.1.1 Diffie-Hellman key protocol

In 1976 Whitfield Diffie and Martin Hellman, who, influenced by Ralph Merkle's work on public-key distribution went down the discrete log route when developing what became known as Diffie-Hellman key exchange method.

6.1.2 El Gamal

El Gamal is based on Diffie-Hellman method. It was described by Taher Elgamal in 1985. It is used in the free GNU Privacy Guard software, recent versions of PGP, and other cryptosystems.

6.1.3 RSA

In 1977 Ronald Rivest, Adi Shamir and Len Adleman developed an algorithm using factoring of prime numbers. This algorithm became known as RSA.

Taking two large prime numbers we will call ‘B’ and ‘Q’. Multiply these numbers to generate ‘N’:

$$N = B * Q$$

Select another number ‘e’ such that:

1. $e < N$
2. e and $(N-1)(Q-1)$ are relatively prime (no common factors except 1)

Find a number ‘p’ such that:

$$(ep - 1) \bmod (B - 1)(Q - 1) = 0$$

Distribute ‘e’ and ‘N’ as the public key and keep ‘p’ as the private key.

For Alice to send an encrypted message she sends:

$$\{CT\} = \{PT\}^e \bmod N$$

Bob receives and retrieves the message by:

$$\{PT\} = \{CT\}^p \bmod N$$

6.1.4 Elliptic curve cryptography (ECC)

Another intractable problem that is used is the assumption that finding the discrete logarithm of an elliptic curve element is infeasible. The size of the elliptic curve determines the difficulty of the problem. It is believed that a smaller group can be used to obtain the same level of security as RSA-based systems. Using a small group reduces storage and transmission requirements.

6.2 Asymmetric Key Protocol summary

Algorithm	Name	Mode	Block size	Keys	Other
RSA	Ron Rivest, Adi Shamir & Len Adleman	Factoring	Variable	1024 – 2048	
Diffie Hellamn	Whitfield Diffie & Martin Hellman	Discrete Log	Variable	Variable	Only used for key exchange
ECC	Elliptical Curve Cryptography	Discrete Log	Variable	80 → 512	160 bits key is equivalent to 1024 bits in RSA

6.3 How Asymmetric Key Cryptography works

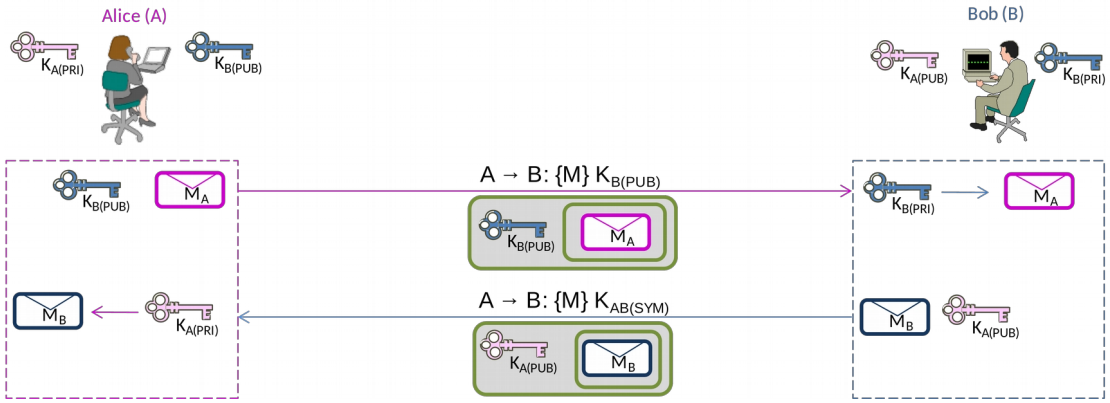

Illustration 7: Asymmetric Key Cryptography

Alice and Bob wish to communicate with each other so they each have a public key and a private key. Alice has a copy of Bob's public and Bob has a copy of Alice's public key.

Alice wishes to send a message to Bob so she encrypts it with Bob's public key $K_{B(PUB)}$ and forwards it to him. Bob extracts the message using his private key $K_{B(PRI)}$.

Bob wishes to respond with a message of his own to Alice so he encrypts it with Alice's public key $K_{A(PUB)}$ and forwards it to her. Alice extracts the message using her private key $K_{A(PRI)}$.

This scheme offers confidentiality of transmission from Alice to Bob and from Bob to Alice. This scheme does not however address the issues of integrity or non-repudiation.

6.4 Digital Signature

Apart from confidentiality of data another application of public-key cryptography is digital signature. Digital signature schemes can be used for sender authentication and non-repudiation. In such a scheme a user who wants to send a message computes a digital signature of this message and then sends this digital signature together with the message to the intended receiver. Digital signature schemes have the property that signatures can only be computed with the knowledge of a private key. To verify that a message has been signed by a user and has not been modified the receiver only needs to know the corresponding public key.

6.5 The hybrid system

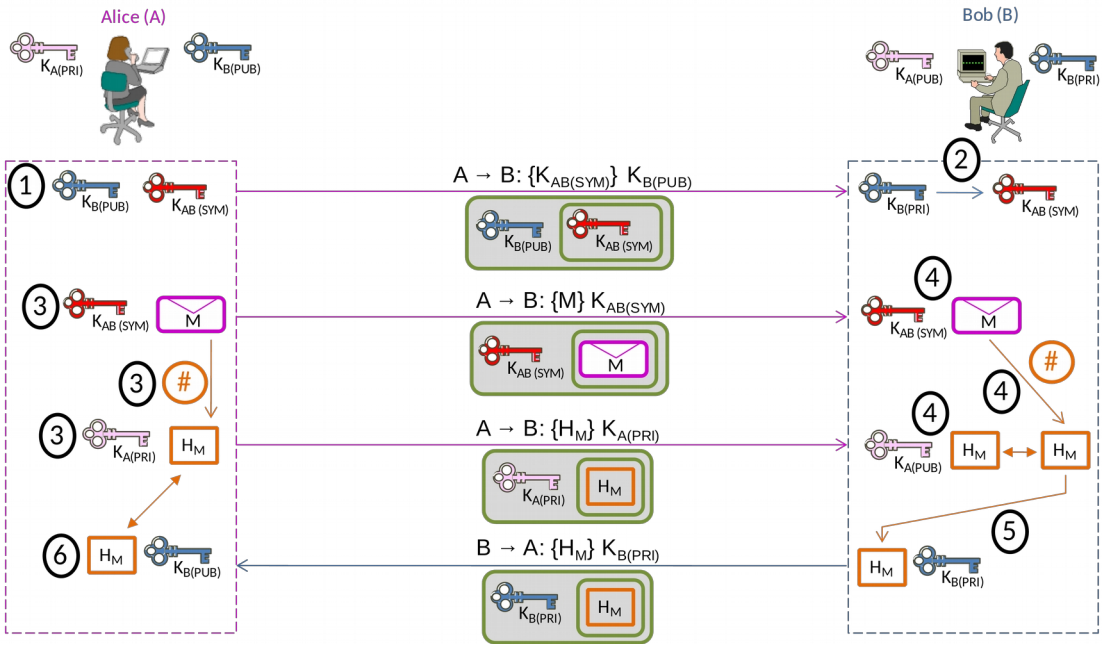

Illustration 8: The hybrid system

This is an enhancement of the system on the previous page where Digital Signatures have been added to provide authentication, integrity and non-repudiation as well as confidentiality.

Alice and Bob wish to communicate with each other so they each have a public key and a private key. Alice has a copy of Bob's public and Bob has a copy of Alice's public key.

Alice wishes to send a message to Bob so she generates a symmetric key $K_{AB(SYM)}$ which she encrypts with Bob's public key $K_{B(PUB)}$ and forwards it to him. Bob extracts $K_{AB(SYM)}$ from the message using his private key $K_{B(PRI)}$.

Alice encrypts the message she wants to send using the shared symmetric key $K_{AB(SYM)}$ and forwards it to Bob, she also generates a message digest from the message and using her own private key $K_{A(PRI)}$ to encrypt the hash forwards the encrypted hash to Bob.

Bob uses the symmetric key $K_{AB(SYM)}$ to decrypt the message, this ensures the confidentiality of the message. He also generates a message digest of it, he then takes the encrypted message digest received and decrypts it using Alice's public key $K_{A(PUB)}$. He now compares the message digest received from Alice with the version he created and they should be the same. If so he is assured of the message integrity.

Finally he can acknowledge the receipt by taking the message digest and encrypting it with his private key $K_{B(PRI)}$ and forwarding it to Alice. Alice decrypts it using Bob's public key $K_{B(PUB)}$ and the output should be identical to the message digest Alice herself created. This verifies the receiver's integrity as well as assuring Alice that Bob received the message.

7. Key Management

One of the obvious issues with the asymmetric key cryptography is how to make the public keys available. For this we need a Public Key Infrastructure (PKI). This is a set of hardware, software, people, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates.

7.1 Certificate Authorities (CA)

CAs are web sites that publish the key bound to a given user. This is achieved using the CA's own key, so that trust in the user key relies on one's trust in the validity of the CA's key. The mechanism that binds keys to users is called the Registration Authority (RA), which might or might not be separate from the CA. The key-user binding are established, depending on the level of assurance the binding has, by software or under human supervision.

The term trusted third party (TTP) may also be used for certificate authority (CA). Moreover, PKI is itself often used as a synonym for a CA implementation.

The ITU-T standard for Certificate Authority is included within the X.509 system.

7.2 Web of Trust

An alternative approach to the problem of public authentication of public key information is the web of trust scheme, which uses self-signed certificates and third party attestations of those certificates. PGP and GnuPG are examples of implementations of the web of trust model. They allow the use of e-mail digital signatures for self-publication of public key information; it is relatively easy to implement one's own Web of Trust.

7.3 Implementations

7.3.1 Privacy Enhanced Mail (PEM)

PEM was an early IETF proposal for securing email using public key cryptography. It has never seen wide deployment as it depended on prior deployment of a hierarchical public key infrastructure (PKI) with a single root. Deployment of such a PKI proved impossible due to cost and legal liability of the root CAs became understood. It was also seen as not a good idea to impose central authority to e-mail.

- RFC 1421 PEM: Part I: Message Encryption and Authentication Procedures
- RFC 1422 PEM: Part II: Certificate-Based Key Management
- RFC 1423 PEM: Part III: Algorithms, Modes, and Identifiers
- RFC 1424 PEM: Part IV: Key Certification and Related Services.

7.3.2 Pretty Good Privacy (PGP)

PGP was created by Philip Zimmermann in 1991, it is a program that provides cryptographic privacy and authentication. PGP is used for signing, encrypting and decrypting e-mails to increase the security of e-mail communications.

PGP follows the OpenPGP standard (RFC 4880) for encrypting and decrypting data.

PGP uses a serial combination of hashing, data compression, symmetric-key cryptography, and public-key cryptography. Each public key is bound to a user name and/or an e-mail address. The first version of this system was a web of trust, however current versions of PGP encryption include both web of trust and certificate authority options through an automated key management server.

7.3.3 Secure/Multi-purpose Internet Mail Extensions (S/MIME)

MIME is the standard that extends the format of e-mail to support:

- Text in character sets other than ASCII
- Non-text attachments
- Message bodies with multiple parts
- Header information in non-ASCII character sets.

S/MIME is a standard for adding cryptographic signature and encryption services to MIME data.

S/MIME is defined in RFC 2633. S/MIME was originally developed by RSA Data Security. However it is now managed by the IETF.

8. GNU Privacy Guard

GnuPG is a complete and free implementation of the OpenPGP standard as defined by RFC4880 (also known as PGP). GnuPG allows to encrypt and sign data and communication, features a versatile key management system as well as access modules for all kinds of public key directories. GnuPG, also known as GPG, is a command line tool with features for easy integration with other applications. A wealth of frontend applications and libraries are available. Version 2 of GnuPG also provides support for S/MIME and Secure Shell (ssh).

GnuPG is Free Software (meaning that it respects your freedom). It can be freely used, modified and distributed under the terms of the GNU General Public License .

Project Gpg4win provides a Windows version of GnuPG stable. It is nicely integrated into an installer and features several front-ends as well as English and German manuals.

8.1 Generate a private key

Most people make their keys valid until infinity, which is the default option. If this is done don't forget to revoke the key when it is no longer in use.

Make sure that the name on the key is not a pseudonym, and that it matches the name in the users passport, or other government issued photo-identification! Additional e-mail addresses can be added to the key later.

A passphrase will be asked for twice. Usually, a short sentence or phrase that isn't easy to guess can be used. Next a request will be made to tap on the keyboard or do any of the things normally done on the computer in order for randomisation to take place. This is done so that the encryption algorithm has more human-entered elements, which, combined with the passphrase entered above, will result in the user's private key.

Key-ID of the created key is: **6E64AF4C**

```
$ gpg --gen-key

gpg (GnuPG) 1.4.16; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory `/home/alovelace/.gnupg' created
gpg: new configuration file `/home/alovelace/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/alovelace/.gnupg/gpg.conf' are not yet active during
this run
gpg: keyring `/home/alovelace/.gnupg/secring.gpg' created
gpg: keyring `/home/alovelace/.gnupg/pubring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 1w
Key expires at Wed 16 Mar 2016 14:04:20 EAT
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Ada Lovelace
Email address: alovelace@mak.ac.ug
Comment: March Key
You selected this USER-ID:
    "Ada Lovelace (March Key) <alovelace@mak.ac.ug>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.

Enter passphrase: babbage
Re-enter passphrase: babbage

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

gpg: /home/alovelace/.gnupg/trustdb.gpg: trustdb created
gpg: key 6E64AF4C marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2016-03-16
pub   2048R/6E64AF4C 2016-03-09 [expires: 2016-03-16]
      Key fingerprint = E150 F5AC F0E5 A492 6891  0903 F315 80E3 6E64 AF4C
uid                  Ada Lovelace (March Key) <alovelace@mak.ac.ug>
sub   2048R/DC6CB630 2016-03-09 [expires: 2016-03-16]
```

8.2 Generate a public key

```
$ gpg --armor --output pubkey.txt --export 'Ada Lovelace'

$ cat pubkey.txt

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQENBFbgA5sBCAC12JYl3KtCjH7jOtfZWBw7gISy5Zl8Y4WMQnEsD7F/na1xQqWB
2kN8ka2MzBCyUFlWQ6sJu/F8jfkzS3YGy4eCa7OZeAdQgZ4EQU+eC1rIo8cLhPA+
jyL5EacQ+jG4kBDsLD+kD8AA55whmAGoapK2lZNyX8tuoW7Ex94BkATB3EgwJ/0Q
53aGrsH93BhIEesc32duvpS0uRe9xY+iEnU9ZquCE6hdCqJBXHo2HdCqs2nN8os8
AlwAfLvN7uRc4Yv7qi5tpWM5+9L30lZlm2/Ydkl2WPxotkCp6mqp+RvZmL7w6hh/
dmflZ/Ts+SzrPMdt9QtE/hJA/J/j8uOedc8jABEBAAG0K0FkYSBMb3ZlbGFjZSAo
TWFyY2ggS2V5KSA8YWxvdmVsYWNlQGMycy5pZT6JAT4EEwECACgFAlbgA5sCGwMF
CQAJOoAGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJEPMVgONuZK9MqIEH/1B3
BHTjJGsXbWsobJnKMJuJeHNaL+9ibmHkgPK3r1K77D8n0xyO4/k0rpS/BNbp2e9V
hEbIpk3/tgIJOGgEhm7ckSmTZSfOyTBi5YO0c/GzhAptNKDbk+qSRVgoV+qtIaeE
PBvUWthZzpa6qRO8b24hdi7QwR354Lf2ZOnU/+WY5/DBGxl4+NGJ3BIN5wB7yL8L
PeHTAyseftgN0wR6C9AwEXx7mW0kBLLFVEmwAB6sJzXvgOOsRpQ5Wr5bF5C1CWf5
MG+VQ0abjnteP3I6YibQcDEExqDXfo1nebVzKu1Nke7bCef6jvrEJ6W2BlxAGf0L
e8c2+SB5QsJFEAGmBSG5AQ0EVuADmwEIAMGIj2BqX2OP2kl7GYXaiaV+xSxndNR0
Rtv7yvehKbJ9YhmplxyHLOIXBqoFWC9YUPcoy40HmhfPrhXrvIA5Uiema6dm/BFB
OtnrpHM0JWHgBwmk3GOQWH2WKHwlRjIhc36l93wesuJYENskIy/WtLEfiuvkS4ZA
fiEhw91VVp0LFTrBahprOgYzcOhra78RhvxI8/vTS03a1GYryuOQoArCjj+TFNMh
GIVgxIY2XWxOlKO7hh75VRRrbM6dhZJkemLKPiKzqbpPfpQaCN11kytQLtJ+r0KH
Lrv3GjhGaChRehDSkVoSltPzsYpSslj/bG5jK0BqmTRzNnOUXQjWXvsAEQEAAYkB
JQQYAQIADwUCVuADmwIbDAUJAAk6gAAKCRDzFYDjbmSvTBUtB/0cxtlyK9jB82rl
QVCNViJIsfnKYC+wZ4h84HhoCpzyTBweRm1nnSNu06paps+rS/GXQ0yOOfT4b/NA
Lv5iJwKANRqkShH4LsxbGYd8Ps/jMEc8lRnSTNwlHnKGzjSco9wGnF/A2omqc2gd
LAFlHZPUJDnzhG2H5jHvgwJs6Oncgs5FyjtA/FnUUqMzy+ITQCbYQEnDQn8CmNyB
Vnxz04u+Td2ajRznD3V29UvXgTaG7gU5842CNsLiezrfqqPgnNnRISxpAboy3xCp
UbqBG/i8z6hNwGDBZRGuwKROAYC5dNDE+SBugYub16SDkhe2dR5tGbURFPSeOdLp
f17ONf3/
=JYlA
-----END PGP PUBLIC KEY BLOCK-----
```

The public key can be freely distributed, posted on a web site, or otherwise distributed. It can also be registered with public keyservers so that others can retrieve the key without having to contact the owner directly.

```
$ gpg --send-keys 'Ada Lovelace' --keyserver hkp://subkeys.pgp.net
```

8.3 Encrypting a file for personal use

Taking a file **MyFile.txt** as a target to experiment with.

```
$ ls -la | grep MyFile
-rw-r--r--  1 dobriain dobriain     74 Mar  9 14:23 MyFile.txt

$ cat MyFile.txt
MyFile
======

This is a file used as part of the exercise on encryption.
```

Now encrypt the file. The argument to the **--recipient** is the name used when generating the private key. Note the output is a new file **MyFile.txt.gpg**.

```
$ gpg --encrypt --recipient 'Ada Lovelace' MyFile.txt

$ ls -la | grep MyFile
-rw-r--r--  1 dobriain dobriain     74 Mar  9 14:23 MyFile.txt
-rw-r--r--  1 dobriain dobriain    406 Mar  9 14:23 MyFile.txt.gpg
```

Use the file command to interrogate the file type and the cat command to view the contents.

```
$ file MyFile.txt
MyFile.txt: ASCII text

$ file MyFile.txt.gpg
MyFile.txt.gpg: data

$ cat MyFile.txt
MyFile
======

This is a file used as part of the exercise on encryption.

$ cat MyFile.txt.gpg
?#
?!?l?0#?P?#?a??E??#?,??6?VU+??vU`<??#<"?.?i}|?????]
                                                                vm??#Tq<?&b???j??|
#?h??????G?H?-d??g??  r#???$?O?I??UB?
        ?2%???#?????#)0???O?f?J?O?^Wp#???0+<??
                                                              ??[??#6??X?#K?M#?Hb??#
    ??~##)?4??0   !e?8'?&?@?Db?Xn?'?9?`v?#??sNc???Z?F$^?????P?tUL#?^?o#_?
%?#? ???^??~??h?f?#?##?_???
????Rc2
       ????????@#?\#?2???&?=#,#D
?6?#N???#?????c??#Z#6I
                      ???,N?r?=(0?????#?#?
```

8.4 Decrypting the file for personal use

Decrypt the file to a new file called **MyFile2.txt**.

```
$ gpg --output MyFile2.txt --decrypt MyFile.txt.gpg
You need a passphrase to unlock the secret key for
user: "Ada Lovelace (March Key) <alovelace@mak.ac.ug>"
2048-bit RSA key, ID DC6CB630, created 2016-03-09 (main key ID 6E64AF4C)

gpg: encrypted with 2048-bit RSA key, ID DC6CB630, created 2016-03-09
      "Ada Lovelace (March Key) <alovelace@mak.ac.ug>"

Enter passphrase: babbage
```

Check the new file and use the **diff** command to confirm it is identical to the original file **MyFile.txt**.

```
$ cat MyFile2.txt
MyFile
======

This is a file used as part of the exercise on encryption.

$ diff MyFile.txt MyFile2.txt
```

8.5 Passing encrypted files to another person

Open the **TEL3214-Client-Server-Example.imn** network. On the server create a private key and from it a public key for Ada Lovelace.

```
root@svr1:/tmp/pycore.54569/svr1.conf# gpg --gen-key

gpg (GnuPG) 1.4.18; Copyright (C) 2014 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory `/root/.gnupg' created
gpg: new configuration file `/root/.gnupg/gpg.conf' created
gpg: WARNING: options in `/root/.gnupg/gpg.conf' are not yet active during
this run
gpg: keyring `/root/.gnupg/secring.gpg' created
gpg: keyring `/root/.gnupg/pubring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default) 1
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection?
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 1
Key expires at Thu 10 Mar 2016 13:02:39 GMT
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Ada Lovelace
Email address: alovelace@cedat.mak.ac.ug
Comment: Ada Lovelace March key
You selected this USER-ID:
    "Ada Lovelace (Ada Lovelace March key) <alovelace@cedat.mak.ac.ug>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.

Enter passphrase: babbage
Re-enter passphrase: babbage
```

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 3AAB4367 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2016-03-10
pub   2048R/3AAB4367 2016-03-09 [expires: 2016-03-10]
      Key fingerprint = 7F0F EE48 BA1D 5DF9 F7F0  A3D9 D78C 0277 3AAB 4367
uid                                  Ada  Lovelace  (Ada  Lovelace  March  key)
<alovelace@cedat.mak.ac.ug>
sub   2048R/E5DE8209 2016-03-09 [expires: 2016-03-10]
```

Now create a public key from this.

```
root@svr1:/tmp/pycore.54569/svr1.conf# gpg --armor --output pubkey.txt
--export 'Ada Lovelace'

root@svr1:/tmp/pycore.54569/svr1.conf# cat pubkey.txt
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQENBFbgHzkBCADASnR2j4651YHPINsvpY9DIjX7MomAxzjpHJmQHdxp8m5jLyDT
ild4zttzJgtk6QVE97+ozAo0Artk/uh3nXLGFKJwe7h491T7hePjMT3T5bfrAx9p
Wy8rP8XhOsw4kwv2x6MXMg+9fAxZbL5sZXS/4NUUsSl5XVaPsDwHBz2vHyxQ9RKA
uQARFR88xeMXpY3Z6So4YGhsKkDsMa/K6u/SBf26QlVtaRTBZx/urzrlzeBbYtgz
6LA+4RevDs7PcbxhRUq2w/TV4CA7oL/36K/LmdwjrrE2QBd93qQaGYeklGnBQOhT
sctAmGrMKZiT3uuj+4hrC6ludbU7YqIJcQFrABEBAAG0QUFkYSBMb3ZlbGFjZSAo
QWRhIExvdmVsYWNlIE1hcmNoIGtleSkgPGFsb3ZlbGFjZUBjZWRhdC5tYWsuYWMu
dWc+iQE+BBMBAgAoBQJW4B85AhsDBQkAAVGABgsJCAcDAgYVCAIJCgsEFgIDAQIe
AQIXgAAKCRDXjAJ3OqtDZxzPB/9zGZD3TJJ/RyFcZeyMVfpVskVSBc01FBt562GI
VFrwTYvf+k6WPX/B2KXsebNx2bJBvCXosQDotIDM6yTNlP8AUUM2pVdhbhuDpZMC
XYIz3PmdCHQ+0si1vpNC8AN22stCtZe7qMWMzk7MGEblE2Ie/WQRPmo9xwQ+tPEq
d5UKK9GqSOG1PdyZLxzZ5ERhMDYpszjt7oTx6cy4Uag7OrusAwDhMjufqxFWLuJu
u/1CfN+QVFvrlhuip80TB2cjLGCqMpjL+sKhj5d3I6CD9TKtQ8MX9FL1QudgVhVg
W+Qm6aNj5N748dgt3LseiUurVjNs0VZ6dtMFpcR5GxQ022MduQENBFbgHzkBCACw
xPCjRq6br7yCqCcXopJ1tKwe5vxvXKyBIADvUVX97evQaxj5eTczgvNrn9fIlb7X
9NTH+yKxize+bFky8IKbwCUmgur2uBKEhXVcyDOJapkgEIGS6KYK4Su8ucTyGBXS
+l88LNxYCcxBDiQlWPCWxt2czws3AZNCTES3LQXgdr9jBYSYRh9Kif7VxH5Aqgyx
vrGYdnq5j9CxQxioHRrgrpA6A6rIOo/DVZ8IONnTHMWT0Y37k4cMT6Gv0ieNu6aW
d6XXljFv+EDcAxGk3DzH8JAJxnVIjXXjUqAm1yreRt0ylgcZWemQwzY1FXsH8UI2
RzHUH5EgvUaGZMFOSXa9ABEBAAGJASUEGAECAA8FAlbgHzkCGwwFCQABUYAACgkQ
14wCdzqrQ2czZggAoWR6Yuiry8k3S/GnkMDh3/jX5aLjf1QEWpvi5SkXGeH3GXAl
5AFOYzmgqp5zqcKIp9gqt1VQi9uKefRGpKRlth8A9WSRxzEOyYB1BrSkuXLtnGmF
PV8CegDUlZDqINkVNb8RObXmEcEq4JZRvHyJneTbsSTAXSkE7eUWEh7z8Sm+M6qi
dxEy8yvHmFZSkz/vYcYeGTvaM8og5JWv1Iw9bdSF7kVbs9GljWI4VnAQ1q/xjFjz
ulKfO/Tp6eShduYdebgQ6n8T8rwYSGrKl//emIZi6VfdL1U/CM/7Ia0Yc36yMQXG
4FjxE5dcfrS+y7K7ZaLG1is8oP+aF5tcPcWEYQ==
=BAof
-----END PGP PUBLIC KEY BLOCK-----
```

8.5.1 Public key

On the ***lap1*** get the public key from the ***svr1***.

```
root@lap1:/tmp/pycore.54569/lap1.conf# sftp root@10.0.2.10
root@10.0.2.10's password: root
Connected to 10.0.2.10.

sftp> cd /tmp/pycore.54569/svr1.conf

sftp> pwd
Remote working directory: /tmp/pycore.54569/svr1.conf

sftp> ls
defaultroute.sh  etc.apache2           etc.ssh         pubkey.txt run.lock
startsshd.sh          var.ftp         var.lock.apache2 var.log
var.log.apache2       var.run         var.run.apache2 var.run.sshd
var.run.vsftpd.empty  var.www         vsftpd.conf

sftp> get pubkey.txt
Fetching /tmp/pycore.54569/svr1.conf/pubkey.txt to pubkey.txt
/tmp/pycore.54569/svr1.conf/pubkey.txt 100% 1763     1.7KB/s   00:00

sftp> quit
```

The lap1 now has the public key from svr1.

```
root@lap1:/tmp/pycore.54569/lap1.conf# ls
defaultroute.sh  pubkey.txt  var.log  var.run
root@lap1:/tmp/pycore.54569/lap1.conf#
```

8.5.2 Generate a secret file and encrypt

```
root@lap1:/tmp/pycore.54569/lap1.conf# echo "This is my little secret" > secretFile.txt

root@lap1:/tmp/pycore.54569/lap1.conf# cat secretFile.txt
This is my little secret
```

Import Ada Lovelace's public key just copied over from ***svr1***.

```
root@lap1:/tmp/pycore.54569/lap1.conf# gpg --import pubkey.txt
gpg: key 3AAB4367: "Ada Lovelace (Ada Lovelace March key)
<alovelace@cedat.mak.ac.ug>" not changed
gpg: Total number processed: 1
gpg:              unchanged: 1
```

Search for Ada Lovelace's key in the keyring.

```
root@lap1:/tmp/pycore.54569/lap1.conf# gpg --list-keys
/root/.gnupg/pubring.gpg
------------------------
pub   2048R/3AAB4367 2016-03-09 [expires: 2016-03-10]
uid                  Ada Lovelace (Ada Lovelace March key) <alovelace@cedat.mak.ac.ug>
sub   2048R/E5DE8209 2016-03-09 [expires: 2016-03-10]
```

Encrypt the secret file **pubkey.txt** with Ada Lovelace's public key.

```
root@lap1:/tmp/pycore.54569/lap1.conf# gpg --encrypt --recipient 'Ada
Lovelace' secretFile.txt
```

There is now a new encrypted file in the local directory. Confirm it is encrypted.

```
root@lap1:/tmp/pycore.54569/lap1.conf# ls secretFile*
secretFile.txt   secretFile.txt.gpg

root@lap1:/tmp/pycore.54569/lap1.conf# file secretFile.txt.gpg
secretFile.txt.gpg: PGP RSA encrypted session key - keyid: 9EBC8885 982DEE5
RSA (Encrypt or Sign) 2048b .
```

Send the new encrypted file to ***svr1***.

```
root@lap1:/tmp/pycore.54569/lap1.conf# sftp root@10.0.2.10
root@10.0.2.10's password: root
Connected to 10.0.2.10.
sftp> cd /tmp/pycore.54569/svr1.conf

sftp> put secretFile.txt.gpg
Uploading secretFile.txt.gpg to
/tmp/pycore.54569/svr1.conf/secretFile.txt.gpg
secretFile.txt.gpg 100%  366     0.4KB/s   00:00

sftp> quit
```

8.6 Decrypt secret file on svr1

Confirm secret file is on ***svr1***.

```
root@svr1:/tmp/pycore.54569/svr1.conf# ls secret*
secretFile.txt.gpg
```

Confirm secret file is on ***svr1***.

```
root@svr1:/tmp/pycore.54569/svr1.conf# gpg --output lap1_secret.txt --decrypt
secretFile.txt.gpg

You need a passphrase to unlock the secret key for
user: "Ada Lovelace (Ada Lovelace March key) <alovelace@cedat.mak.ac.ug>"
2048-bit RSA key, ID E5DE8209, created 2016-03-09 (main key ID 3AAB4367)

Enter passphrase: babbage

gpg: encrypted with 2048-bit RSA key, ID E5DE8209, created 2016-03-09
      "Ada Lovelace (Ada Lovelace March key) <alovelace@cedat.mak.ac.ug>"
```

Check the decrypted file.

```
root@svr1:/tmp/pycore.54569/svr1.conf# cat lap1_secret.txt
This is my little secret
```

- To summarise, ***lap1*** received the public key for Ada Lovelace on ***svr1***.
- ***lap1*** used this public key to encrypt a file.
- ***lap1*** sent the file to ***svr1***.
- ***svr1*** decrypted the file using Ada Lovelace's private key.

8.7 Digitally signing a file

GPG also provides a mechanism to digitally sign a file. Ada Lovelace wishes to have a file signed so those with her public key can confirm that she did indeed create the file, i.e. non repudiation.

```
root@svr1:/tmp/pycore.54569/svr1.conf# echo 'This file will be signed" >
sign.txt

root@svr1:/tmp/pycore.54569/svr1.conf# gpg --armor --detach-sign sign.txt

You need a passphrase to unlock the secret key for
user: "Ada Lovelace (Ada Lovelace March key) <alovelace@cedat.mak.ac.ug>"
2048-bit RSA key, ID 3AAB4367, created 2016-03-09

Enter passphrase: babbage


root@svr1:/tmp/pycore.54569/svr1.conf# ls -la sign*
-rw-rw-rw- 1 root root  25 Mar  9 19:18 sign.txt
-rw-rw-rw- 1 root root 473 Mar  9 19:20 sign.txt.asc

root@svr1:/tmp/pycore.54569/svr1.conf# file sign.txt.asc
sign.txt.asc: PGP signature Signature (old)


root@svr1:/tmp/pycore.54569/svr1.conf# cat sign.txt.asc
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iQEcBAABAgAGBQJW4HdvAAoJENeMAnc6q0NnQekH/3bck0fGF3FSblpQSeVfrZLJ
sCpGNvLeDv+PpyPCLRtPqwHJlCGMFsP6FCh9d07EYJIYnpHuvnwSPaJCsXqS6OcX
f11vbSo24BLWYDN/T9v7Kt3ui7jEhUYqQNZQXMzlciVrRpYqU5F4vQClTChQXZ2l
9R71QuOGi98AsAZfitAXU3L3SLPxHwieJefqsuWgLqI75uuB2atoy+FvrFSQ7gdv
nW9ylvehHFLtyXwKMQUZ5OSGW/DUl0M6CRVofu4aY9BsIHV5z9yiMiQG3Vi2t5Kl
/4YAzN34jy0YHPDTFrv3qCdgGtuB/Zv9/6CkYYRjP4XyhBtJM74483lDF+hJzjU=
=SS9+
-----END PGP SIGNATURE-----
```

8.7.1 Verifying a signed file

On lap1 get the two files, the ***sign.txt*** and ***sign.txt.asc*** from ***svr1***.

```
root@lap1:/tmp/pycore.54569/lap1.conf# sftp root@10.0.2.10
root@10.0.2.10's password: root
Connected to 10.0.2.10.


sftp> cd /tmp/pycore.54569/svr1.conf


sftp> mget sign.txt*
Fetching /tmp/pycore.54569/svr1.conf/sign.txt to sign.txt
/tmp/pycore.54569/svr1.conf/sign.txt          100%   25     0.0KB/s   00:00
Fetching /tmp/pycore.54569/svr1.conf/sign.txt.asc to sign.txt.asc
/tmp/pycore.54569/svr1.conf/sign.txt.asc      100%  473     0.5KB/s   00:00


sftp> exit
```

Confirm signature.

```
root@lap1:/tmp/pycore.54569/lap1.conf# gpg --verify sign.txt.asc sign.txt
gpg: Signature made Wed 09 Mar 2016 19:20:15 GMT using RSA key ID 3AAB4367
gpg: Good signature from "Ada Lovelace (Ada Lovelace March key)
<alovelace@cedat.mak.ac.ug>"
```

9. Applications Lab

Carry out the following activities while monitoring the traffic. Note any observations.

SFTP

1. On the NTE Emulator Virtual Machine (VM) create a 100 MB file on the Desktop. Call the file ***100MB_NTE_File***.
2. Compress the file with two different compression protocols.
3. Run the ***TEL3214-Client-Server-Example.imn*** network.
4. Using SFTP connect from ***lap1*** to ***svr1*** using the Username: **nte** and Password: **nte**.
5. Change directory to /**home**/**nte**/**Desktop**.
6. Download the file ***100MB_NTE_File*** to the home directory.
7. Confirm the file downloaded OK.
8. Quit SFTP.

SSH

1. Use SSH to connect to the server ***svr1*** from ***lap1***.
2. Change to the directory /**home**/**nte**/**Desktop**.
3. Change the name of the file ***100MB_NTE_File*** to ***100MB_NTE_File.old***.
4. Logout of ***svr1***.

HTTP

1. Connect to the web-server on ***svr1*** from ***lap1***.
2. Monitor traffic between devices.

GPG

1. Connect to ***svr1***.
2. Create a private/public key pair.
3. Encrypt the file from ***100MB_NTE_File.old***.
4. From ***lap1***, SFTP to ***svr1*** and download the encrypted file ***100MB_NTE_File.old*** as well as the public key.
5. Exit from SFTP.
6. Add the public key to the keyring.
7. Decrypt the downloaded file.

This page is intentionally blank