# Lab9: Secure Networking: SSH and HTTPS - Solutions

**COM-208: Computer Networks**

## Objectives

- Learning about the security behind ssh, https and making use of them.

## Prerequisites

- Lecture 10
- Wireshark

## HTTPS

HTTPS is a marked improvement over HTTP when it comes to authentication and confidentiality. In this Lab, we will examine the following questions about HTTPS:

1. How do HTTP and HTTPS compare with respect to confidentiality? Which of these two protocols can we trust to transmit sensitive data (e.g. passwords) over the network?
2. How do web browsers (Firefox, in particular) handle the issue of Certificate Authorities (CA) with respect to HTTPS?

## Confidentiality

You are going to test the level of confidentiality that is provided to you by the following websites:

- A Test Login Page

- Yahoo!

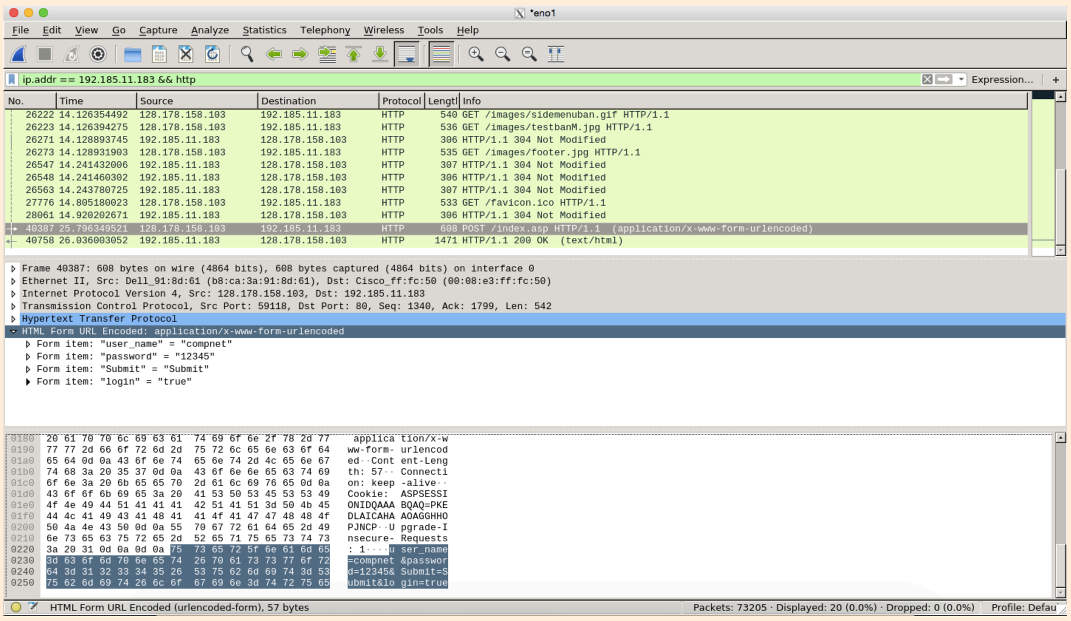
You are going to test that for each website, as follows:

- Start by checking whether the web page address begins with “https://” or “http://”. Note that HTTPS (also called HTTP over TLS, HTTP over SSL, and HTTP Secure) consists of communication over HTTP, but within a connection encrypted by Transport Layer Security (TLS) or its predecessor, Secure Sockets Layer (SSL).
- Start a Wireshark capture. Use a filter for capturing only the traffic to/from the web server: e.g. use an initial filter based on the IP address: “ip.addr == X.X.X.X”. You may find the actual IP addresses using “nslookup” or any other tool (even Wireshark).
- Refresh the login page for the website.
- Submit any username and password for each website. **Do NOT provide a valid password. One of the websites is insecure!**
- Examine the Wireshark trace to identify which protocol is used to connect to the website.
- Examine the trace to see if you can discover the username and password that you provided in the login form. For this, it is better to use a additional filter for the traffic concerning the protocol that is used. You may need to add a second filter to the initial one, by using the AND operator “&&” and examine the content of the messages that are exchanged. Consider adding “http” for the test login page, and “ssl” to for the yahoo login page.

## Questions

a. Which protocols does your browser use to connect to each website? What are the confidentiality guarantees of each of the two websites?

> We access to the test login page website using simple HTTP which provides no confidentiality. Thus, we can retrieve the password we entered in the form (in this case login name: “compnet” and password: “12345”). See figure below.



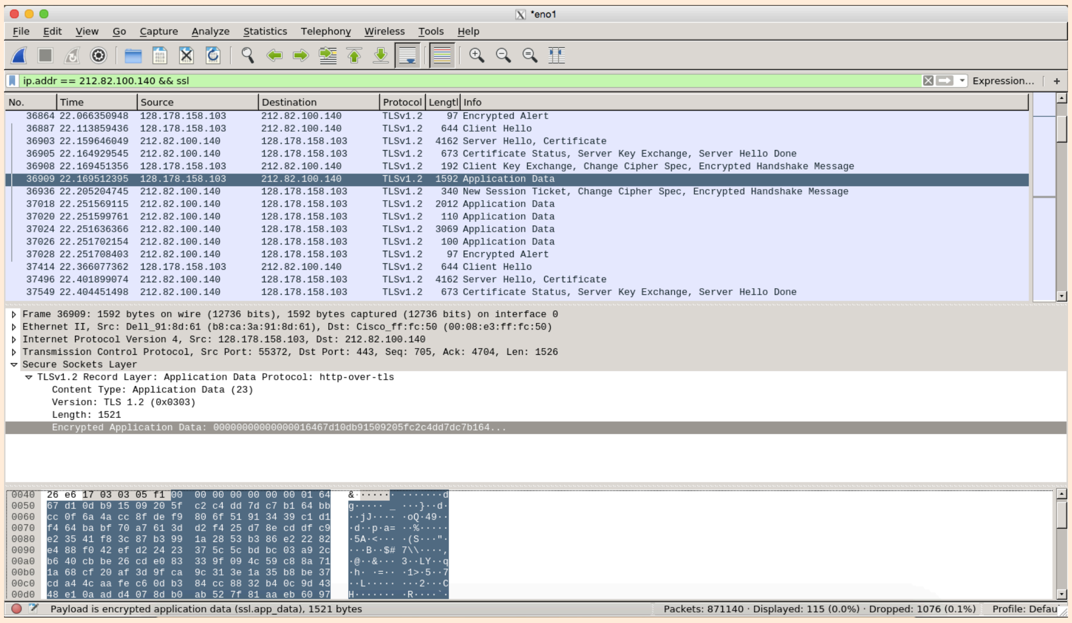
```
ip.addr == 192.185.11.183 && http

No.   Time         Source           Destination      Protocol Lengtl Info
26222 14.126354492 128.178.158.103  192.185.11.183   HTTP      540 GET /images/sidemenuban.gif HTTP/1.1
26223 14.126394275 128.178.158.103  192.185.11.183   HTTP      536 GET /images/testbanM.jpg HTTP/1.1
26271 14.128893745 192.185.11.183   128.178.158.103  HTTP      306 HTTP/1.1 304 Not Modified
26273 14.128931903 128.178.158.103  192.185.11.183   HTTP      535 GET /images/footer.jpg HTTP/1.1
26547 14.241432006 192.185.11.183   128.178.158.103  HTTP      307 HTTP/1.1 304 Not Modified
26548 14.241460302 192.185.11.183   128.178.158.103  HTTP      306 HTTP/1.1 304 Not Modified
26563 14.243780725 192.185.11.183   128.178.158.103  HTTP      307 HTTP/1.1 304 Not Modified
27776 14.805180023 128.178.158.103  192.185.11.183   HTTP      533 GET /favicon.ico HTTP/1.1
28061 14.920202671 192.185.11.183   128.178.158.103  HTTP      306 HTTP/1.1 304 Not Modified
40387 25.796349521 128.178.158.103  192.185.11.183   HTTP      608 POST /index.asp HTTP/1.1  (application/x-www-form-urlencoded)
40758 26.036003052 192.185.11.183   128.178.158.103  HTTP     1471 HTTP/1.1 200 OK  (text/html)

Frame 40387: 608 bytes on wire (4864 bits), 608 bytes captured (4864 bits) on interface 0
Ethernet II, Src: Dell_91:8d:61 (b8:ca:3a:91:8d:61), Dst: Cisco_ff:fc:50 (00:08:e3:ff:fc:50)
Internet Protocol Version 4, Src: 128.178.158.103, Dst: 192.185.11.183
Transmission Control Protocol, Src Port: 59118, Dst Port: 80, Seq: 1340, Ack: 1799, Len: 542
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
    Form item: "user_name" = "compnet"
    Form item: "password" = "12345"
    Form item: "Submit" = "Submit"
    Form item: "login" = "true"

0180  20 61 70 70 6c 69 63 61  74 69 6f 6e 2f 78 2d 77    applica tion/x-w
0190  77 77 2d 66 6f 72 6d 2d  75 72 6c 65 6e 63 6f 64   ww-form- urlencod
01a0  65 64 0d 0a 43 6f 6e 74  65 6e 74 2d 4c 65 6e 67   ed··Cont ent-Leng
01b0  74 68 3a 20 35 37 0d 0a  43 6f 6e 6e 65 63 74 69   th: 57·· Connecti
01c0  6f 6e 3a 20 6b 65 65 70  2d 61 6c 69 76 65 0d 0a   on: keep -alive··
01d0  43 6f 6f 6b 69 65 3a 20  41 53 50 53 45 53 53 49   Cookie:  ASPSESSI
01e0  4f 4e 49 44 51 41 41 41  42 51 41 51 3d 50 4b 45   ONIDQAAA BQAQ=PKE
01f0  44 4c 41 49 43 41 48 41  41 4f 41 47 47 48 48 4f   DLAICAHA AOAGGHHO
0200  50 4a 4e 43 50 0d 0a 55  70 67 72 61 64 65 2d 49   PJNCP··U pgrade-I
0210  6e 73 65 63 75 72 65 2d  52 65 71 75 65 73 74 73   nsecure- Requests
0220  3a 20 31 0d 0a 0d 0a 75  73 65 72 5f 6e 61 6d 65   : 1····u ser_name
0230  3d 63 6f 6d 70 6e 65 74  26 70 61 73 73 77 6f 72   =compnet &passwor
0240  64 3d 31 32 33 34 35 26  53 75 62 6d 69 74 3d 53   d=12345& Submit=S
0250  75 62 6d 69 74 26 6c 6f  67 69 6e 3d 74 72 75 65   ubmit&lo gin=true

HTML Form URL Encoded (urlencoded-form), 57 bytes      Packets: 73205 · Displayed: 20 (0.0%) · Dropped: 0 (0.0%)      Profile: Defau
```

*eno1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression…
+

Figure 1

On the other hand, when we try to access the Yahoo! website, the password cannot be found. This is because HTTPS encrypts transmitted data in order to provide confidentiality. This is done here through the TLSv1.2 protocol. You can filter by SSL to find the packets exchanged with the Yahoo server:

*Client->Server:* ***Client Hello***
*Server->Client:* ***Server Hello, Certificate, Server Hello Done***
*Client->Server:* ***Client Key Exchange, Change Cipher Text, Encrypted Handshake Message***
*Server->Client:* ***Change Cipher Text, Encrypted Handshake Message***
*Server<->Client:* ***Application Data***
*Server<->Client:* ***Encrypted Alert***



```
ip.addr == 212.82.100.140 && ssl

No.   Time         Source           Destination      Protocol Length Info
36864 22.066350948 128.178.158.103  212.82.100.140   TLSv1.2    97 Encrypted Alert
36887 22.113859436 128.178.158.103  212.82.100.140   TLSv1.2   644 Client Hello
36903 22.159646049 212.82.100.140   128.178.158.103  TLSv1.2  4162 Server Hello, Certificate
36905 22.164929545 212.82.100.140   128.178.158.103  TLSv1.2   673 Certificate Status, Server Key Exchange, Server Hello Done
36908 22.169451356 128.178.158.103  212.82.100.140   TLSv1.2   192 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
36909 22.169512395 128.178.158.103  212.82.100.140   TLSv1.2  1592 Application Data
36936 22.205204745 212.82.100.140   128.178.158.103  TLSv1.2   340 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
37018 22.251569115 212.82.100.140   128.178.158.103  TLSv1.2  2012 Application Data
37020 22.251599761 212.82.100.140   128.178.158.103  TLSv1.2   110 Application Data
37024 22.251636366 212.82.100.140   128.178.158.103  TLSv1.2  3069 Application Data
37026 22.251702154 212.82.100.140   128.178.158.103  TLSv1.2   100 Application Data
37028 22.251708403 212.82.100.140   128.178.158.103  TLSv1.2    97 Encrypted Alert
37414 22.366077362 128.178.158.103  212.82.100.140   TLSv1.2   644 Client Hello
37496 22.401899074 212.82.100.140   128.178.158.103  TLSv1.2  4162 Server Hello, Certificate
37549 22.404451498 212.82.100.140   128.178.158.103  TLSv1.2   673 Certificate Status, Server Key Exchange, Server Hello Done

▷ Frame 36909: 1592 bytes on wire (12736 bits), 1592 bytes captured (12736 bits) on interface 0
▷ Ethernet II, Src: Dell_91:8d:61 (b8:ca:3a:91:8d:61), Dst: Cisco_ff:fc:50 (00:08:e3:ff:fc:50)
▷ Internet Protocol Version 4, Src: 128.178.158.103, Dst: 212.82.100.140
▷ Transmission Control Protocol, Src Port: 55372, Dst Port: 443, Seq: 705, Ack: 4704, Len: 1526
▽ Secure Sockets Layer
  ▽ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 1521
      Encrypted Application Data: 0000000000000016467d10db91509205fc2c4dd7dc7b164...

0040  26 e6 17 03 03 05 f1 00  00 00 00 00 00 00 01 64   &·······  ·······d
0050  67 d1 0d b9 15 09 20 5f  c2 c4 dd 7d c7 b1 64 bb   g····· _  ···}··d·
0060  cc 0f 6a 4a cc 8f de f9  80 6f 51 91 34 39 c1 d1   ··jJ····  ·oQ·49··
0070  f4 64 ba bf 70 a7 61 3d  d2 f4 25 d7 8e cd df c9   ·d··p·a=  ··%·····
0080  e2 35 41 f8 3c 87 b3 99  1a 28 53 b3 86 e2 22 82   ·5A·<···  ·(S···"·
0090  e4 88 f0 42 ef d2 24 23  37 5c 5c bd bc 03 a9 2c   ···B··$#  7\\····,
00a0  b6 40 cb be 26 cd e0 83  33 9f 09 4c 59 c8 8a 71   ·@··&···  3··LY··q
00b0  1a 68 cf 20 af 3d 9f ca  9c 31 3e 1a 35 b8 be 37   ·h· ·=··  ·1>·5··7
00c0  cd a4 4c aa fe c6 0d b3  84 cc 88 32 b4 0c 9d 43   ··L·····  ···2···C
00d0  48 e1 0a ad d4 07 8d b0  ab 52 7f 81 aa eb 60 97   H·······  ·R····`·

Payload is encrypted application data (ssl.app_data), 1521 bytes
Packets: 871140 · Displayed: 115 (0.0%) · Dropped: 1076 (0.1%)    Profile: Defau
```

*eno1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

Figure 2

b. **IMPORTANT:** What security advice would you give to a friend/relative who is struggling with computers about this issue: What information from the browser could they use to identify an insecure website?

We cannot trust that a password submitted is secure, just because the webpage does not render the password as it is being typed. One can tell which protocol is being used by checking whether the web page address begins with "https://" or "http:". Most modern browsers (e.g., Firefox) display a padlock icon before the address, if the protocol used to connect to the website is HTTPS. If a user clicks on the padlock, they can also see which CA signed for it.

## Certificate Authorities

To identify itself, a web server provides you with a certificate. This certificate is signed by a Certificate Authority (CA). When a web browser receives a certificate from a website, it checks whether the CA that has issued the certificate belongs to one of the authorities that the browser recognizes.

In Firefox, recognized authorities can be found in:

*Menu (top-right corner of Firefox)/Preferences/Privacy & Security/View Certificates/Authorities*

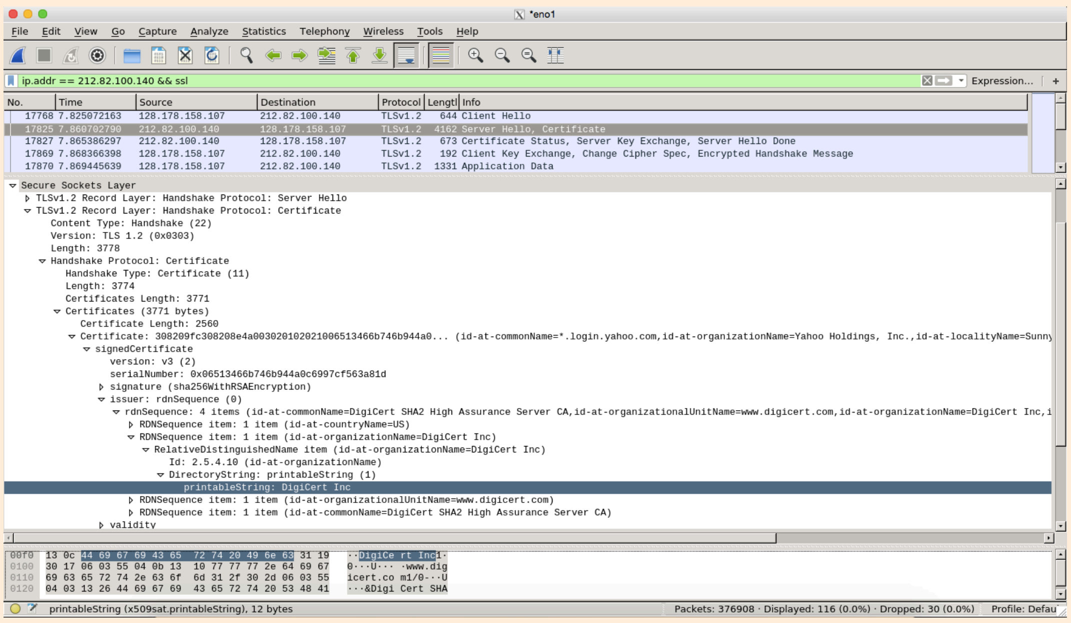
The response from the web server which carries the certificate is carried on a “Certificate”-type record.

## Questions

a. Use Wireshark to discover which Certificate Authority signed the certificate of Yahoo!.

b. Is this Certificate Authority recognized by your web browser?

In Wireshark, find the message “**Certificate**” after the message “Server Hello”. In that message, you can find the certificate issuer, as show in the screenshot below:

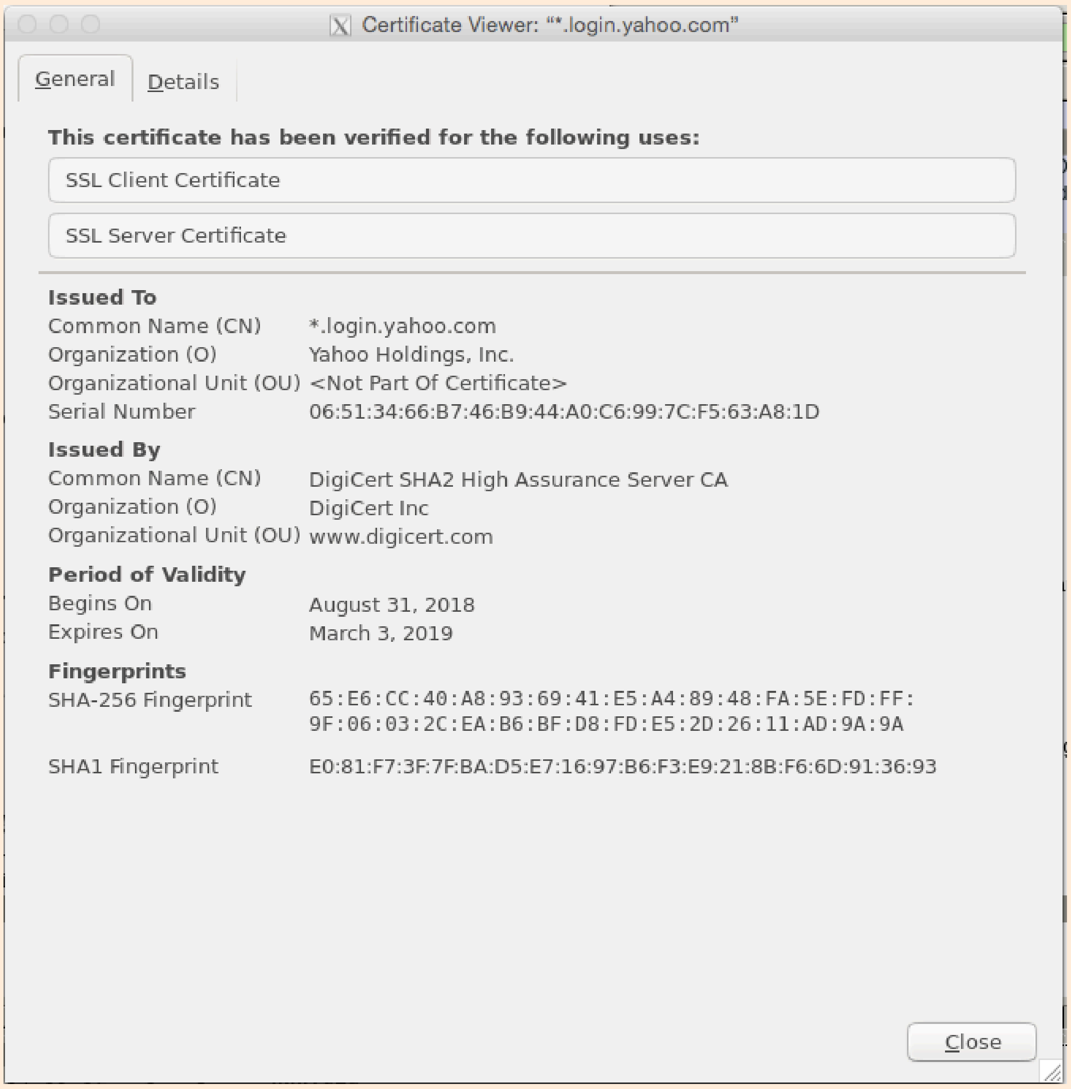


```
ip.addr == 212.82.100.140 && ssl

No.   Time         Source           Destination      Protocol Length Info
17768 7.825072163  128.178.158.107  212.82.100.140   TLSv1.2   644 Client Hello
17825 7.860702790  212.82.100.140   128.178.158.107  TLSv1.2  4162 Server Hello, Certificate
17827 7.865386297  212.82.100.140   128.178.158.107  TLSv1.2   673 Certificate Status, Server Key Exchange, Server Hello Done
17869 7.868366398  128.178.158.107  212.82.100.140   TLSv1.2   192 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
17870 7.869445639  128.178.158.107  212.82.100.140   TLSv1.2  1331 Application Data

Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello
  TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 3778
    Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 3774
      Certificates Length: 3771
      Certificates (3771 bytes)
        Certificate Length: 2560
        Certificate: 308209fc308208e4a003020102021006513466b746b944a0... (id-at-commonName=*.login.yahoo.com,id-at-organizationName=Yahoo Holdings, Inc.,id-at-localityName=Sunny
          signedCertificate
            version: v3 (2)
            serialNumber: 0x06513466b746b944a0c6997cf563a81d
            signature (sha256WithRSAEncryption)
            issuer: rdnSequence (0)
              rdnSequence: 4 items (id-at-commonName=DigiCert SHA2 High Assurance Server CA,id-at-organizationalUnitName=www.digicert.com,id-at-organizationName=DigiCert Inc,i
                RDNSequence item: 1 item (id-at-countryName=US)
                RDNSequence item: 1 item (id-at-organizationName=DigiCert Inc)
                  RelativeDistinguishedName item (id-at-organizationName=DigiCert Inc)
                    Id: 2.5.4.10 (id-at-organizationName)
                    DirectoryString: printableString (1)
                      printableString: DigiCert Inc
                RDNSequence item: 1 item (id-at-organizationalUnitName=www.digicert.com)
                RDNSequence item: 1 item (id-at-commonName=DigiCert SHA2 High Assurance Server CA)
            validity

00f0  13 0c 44 69 67 69 43 65  72 74 20 49 6e 63 31 19   ··DigiCe rt Inc1·
0100  30 17 06 03 55 04 0b 13  10 77 77 77 2e 64 69 67   0···U··· ·www.dig
0110  69 63 65 72 74 2e 63 6f  6d 31 2f 30 2d 06 03 55   icert.co m1/0-··U
0120  04 03 13 26 44 69 67 69  43 65 72 74 20 53 48 41   ···&Digi Cert SHA

printableString (x509sat.printableString), 12 bytes    Packets: 376908 · Displayed: 116 (0.0%) · Dropped: 30 (0.0%)    Profile: Defau
```

*eno1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

Figure 3

The certificate issuer is DigiCert Inc. This authority is recognized by Firefox. To see that, click on the padlock item on Firefox placed on the left-hand side of the address bar. Then, click on the arrow at the top-left of the opened window. Then, click on “More Information” at the bottom of the newly opened window. Lastly, click on "View Certificate" in the newly opened window. The final window shows the details of the authority that is used by yahoo and recognized by Firefox. The final window is as below.


Certificate Viewer: "*.login.yahoo.com"

General Details

**This certificate has been verified for the following uses:**

SSL Client Certificate

SSL Server Certificate

**Issued To**

| | |
|---|---|
| Common Name (CN) | *.login.yahoo.com |
| Organization (O) | Yahoo Holdings, Inc. |
| Organizational Unit (OU) | <Not Part Of Certificate> |
| Serial Number | 06:51:34:66:B7:46:B9:44:A0:C6:99:7C:F5:63:A8:1D |

**Issued By**

| | |
|---|---|
| Common Name (CN) | DigiCert SHA2 High Assurance Server CA |
| Organization (O) | DigiCert Inc |
| Organizational Unit (OU) | www.digicert.com |

**Period of Validity**

| | |
|---|---|
| Begins On | August 31, 2018 |
| Expires On | March 3, 2019 |

**Fingerprints**

| | |
|---|---|
| SHA-256 Fingerprint | 65:E6:CC:40:A8:93:69:41:E5:A4:89:48:FA:5E:FD:FF:<br>9F:06:03:2C:EA:B6:BF:D8:FD:E5:2D:26:11:AD:9A:9A |
| SHA1 Fingerprint | E0:81:F7:3F:7F:BA:D5:E7:16:97:B6:F3:E9:21:8B:F6:6D:91:36:93 |

Close

Figure 4

## Managing Certificate authorities

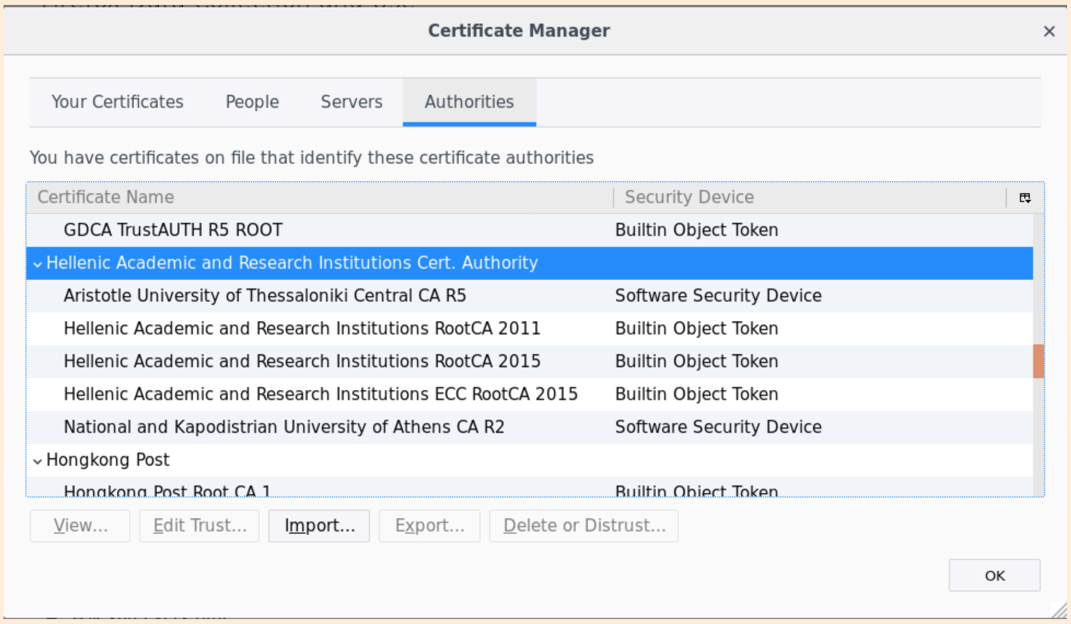
As we said, in Firefox, recognized authorities can be found in

*Menu (top-right corner of Firefox)/Preferences/Privacy & Security/View Certificates/Authorities*

For demonstration purposes, we are going to examine what would happen if we removed one of those servers from the list.

- Visit https://www.auth.gr.
- Go to the list of the recognized server from Firefox. Remove every entry of the parent CA organization ***"Hellenic Academic and Research Institutions Cert. Authority"***. Then, restart Firefox to make changes take effect.
- Visit https://www.auth.gr again. What is the warning message you get?
- Now, visit https://mail.yahoo.com. Why do you not get a similar warning message for the Yahoo! website?

Firefox displays a warning because the site is certified by an unknown authority: Aristotle University of Thessaloniki Central CA R5.

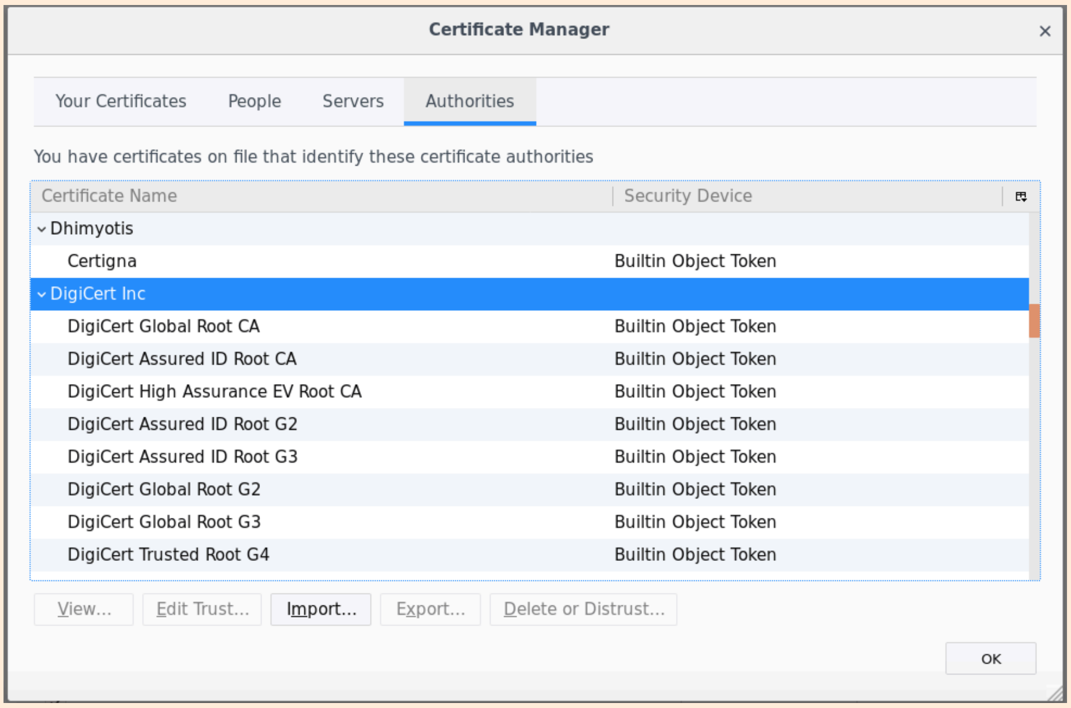


You have certificates on file that identify these certificate authorities

| Certificate Name | Security Device |
|---|---|
| GDCA TrustAUTH R5 ROOT | Builtin Object Token |
| Hellenic Academic and Research Institutions Cert. Authority | |
| Aristotle University of Thessaloniki Central CA R5 | Software Security Device |
| Hellenic Academic and Research Institutions RootCA 2011 | Builtin Object Token |
| Hellenic Academic and Research Institutions RootCA 2015 | Builtin Object Token |
| Hellenic Academic and Research Institutions ECC RootCA 2015 | Builtin Object Token |
| National and Kapodistrian University of Athens CA R2 | Software Security Device |
| Hongkong Post | |
| Hongkong Post Root CA 1 | Builtin Object Token |

Figure 5

Yahoo is certified by DigiCert Inc, known to Firefox, hence there is no warning shown.


Certificate Manager

Your Certificates | People | Servers | Authorities

You have certificates on file that identify these certificate authorities

| Certificate Name | Security Device |
|---|---|
| ⌄ Dhimyotis | |
| Certigna | Builtin Object Token |
| ⌄ DigiCert Inc | |
| DigiCert Global Root CA | Builtin Object Token |
| DigiCert Assured ID Root CA | Builtin Object Token |
| DigiCert High Assurance EV Root CA | Builtin Object Token |
| DigiCert Assured ID Root G2 | Builtin Object Token |
| DigiCert Assured ID Root G3 | Builtin Object Token |
| DigiCert Global Root G2 | Builtin Object Token |
| DigiCert Global Root G3 | Builtin Object Token |
| DigiCert Trusted Root G4 | Builtin Object Token |

Figure 6

## SSH

The `ssh` tool is a powerful utility which grants secure access to a target remote machine. One of the most common use-cases for `ssh` is to run commands on the target machine as if we had direct physical access to that machine.

In this Lab, we are going to learn how to achieve the following:

1. Connect to a target machine.
2. Execute a command on a target machine.
3. Run a graphical application on a target machine.
4. Copy files to/from a target machine.

We will also delve a bit deeper on two important security-related aspects of `ssh`:

1. Setting up public key authentication (i.e., "How can I avoid having to type my password each time I want to connect to a target machine?")
2. Fingerprints (i.e., "How do I know that the machine I am connecting to is really the machine I asked ssh to connect to?")

## Setup

(You should be able to run these exercises on just about any GNU/Linux system which runs an SSH server)

For the purposes of Lab 9, we will be using one of the machines at INF3 as the target machine. The machines at INF3 can be accessible at icin3pcX.epfl.ch, where X is a value in {01, 02, 03, ..., 64}. You will use your GASPAR accounts to access to those machines.

**If you receive an error on connecting to one of the INF3 machines, please try to connect to some other machine by simply choosing another X value in the hostname above. The error might be due to overloading the machine with more ssh connections than the machine can handle.**

## SSH use cases

## Connecting to a target machine

The basic syntax which makes you connect to machine "hostname" using your current username, using the ssh command is:

```
$ ssh hostname
```

In our case, the "hostname" is **icin3pcX.epfl.ch**, where X is a value in {01, 02, 03, ..., 64}.

If you want to log in as a different user, you should provide the following command:

```
$ ssh username\@hostname
```

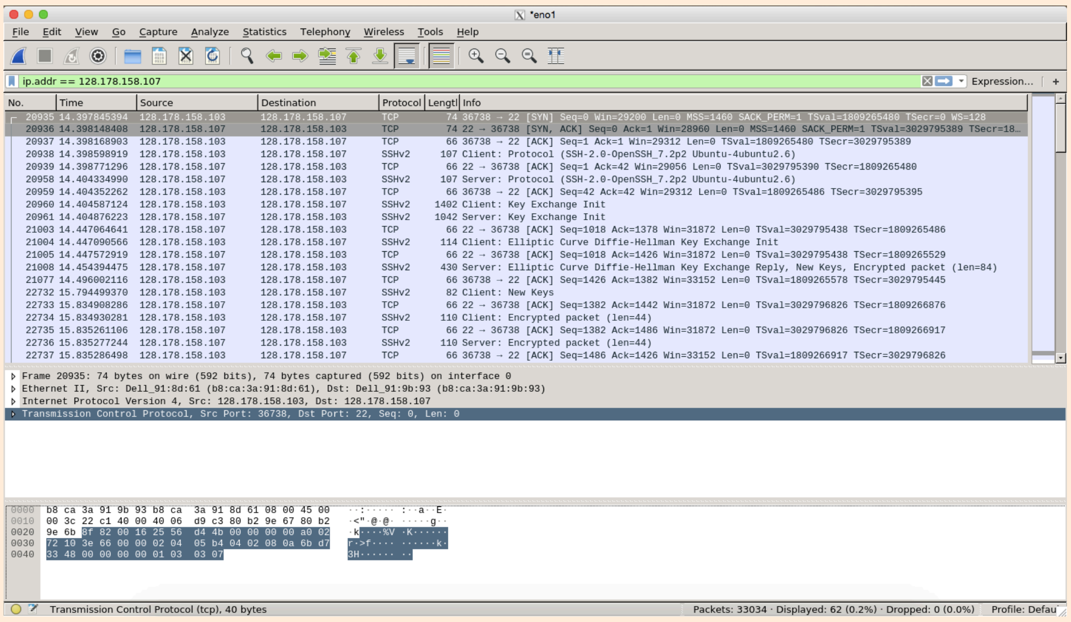
(NOTE: The first time you log in to the server, you will be asked whether you accept a "fingerprint". For the moment, you should type "yes" and continue with the Lab. We will examine fingerprints later in this Lab.)

## Exercise

- Connect to one of the INF3 machines using your GASPAR account.

- Use Wireshark to capture one ssh session. Which transport protocol is used? Which port does the ssh server use?

Transport protocol: TCP; port used by server: 22. There are many packets exchanged even for very few useful data transmitted. We get a lot of ACKs, to achieve good reaction time.


*eno1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression… +

```
ip.addr == 128.178.158.107

No.   Time         Source           Destination      Protocol Lengtl Info
20935 14.397845394 128.178.158.103  128.178.158.107  TCP      74 36738 → 22 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1809265480 TSecr=0 WS=128
20936 14.398148408 128.178.158.107  128.178.158.103  TCP      74 22 → 36738 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=3029795389 TSecr=18…
20937 14.398168903 128.178.158.103  128.178.158.107  TCP      66 36738 → 22 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1809265480 TSecr=3029795389
20938 14.398598919 128.178.158.103  128.178.158.107  SSHv2   107 Client: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.6)
20939 14.398771296 128.178.158.107  128.178.158.103  TCP      66 22 → 36738 [ACK] Seq=1 Ack=42 Win=29056 Len=0 TSval=3029795390 TSecr=1809265480
20958 14.404334990 128.178.158.107  128.178.158.103  SSHv2   107 Server: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.6)
20959 14.404352262 128.178.158.103  128.178.158.107  TCP      66 36738 → 22 [ACK] Seq=42 Ack=42 Win=29312 Len=0 TSval=1809265486 TSecr=3029795395
20960 14.404587124 128.178.158.103  128.178.158.107  SSHv2  1402 Client: Key Exchange Init
20961 14.404876223 128.178.158.107  128.178.158.103  SSHv2  1042 Server: Key Exchange Init
21003 14.447064641 128.178.158.107  128.178.158.103  TCP      66 22 → 36738 [ACK] Seq=1018 Ack=1378 Win=31872 Len=0 TSval=3029795438 TSecr=1809265486
21004 14.447090566 128.178.158.103  128.178.158.107  SSHv2   114 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
21005 14.447572919 128.178.158.107  128.178.158.103  TCP      66 22 → 36738 [ACK] Seq=1018 Ack=1426 Win=31872 Len=0 TSval=3029795438 TSecr=1809265529
21008 14.454394475 128.178.158.107  128.178.158.103  SSHv2   430 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=84)
21077 14.496002116 128.178.158.103  128.178.158.107  TCP      66 36738 → 22 [ACK] Seq=1426 Ack=1382 Win=33152 Len=0 TSval=1809265578 TSecr=3029795445
22732 15.794499370 128.178.158.103  128.178.158.107  SSHv2    82 Client: New Keys
22733 15.834908286 128.178.158.107  128.178.158.103  TCP      66 22 → 36738 [ACK] Seq=1382 Ack=1442 Win=31872 Len=0 TSval=3029796826 TSecr=1809266876
22734 15.834930281 128.178.158.103  128.178.158.107  SSHv2   110 Client: Encrypted packet (len=44)
22735 15.835261106 128.178.158.107  128.178.158.103  TCP      66 22 → 36738 [ACK] Seq=1382 Ack=1486 Win=31872 Len=0 TSval=3029796826 TSecr=1809266917
22736 15.835277244 128.178.158.107  128.178.158.103  SSHv2   110 Server: Encrypted packet (len=44)
22737 15.835286498 128.178.158.103  128.178.158.107  TCP      66 36738 → 22 [ACK] Seq=1486 Ack=1426 Win=33152 Len=0 TSval=1809266917 TSecr=3029796826

▷ Frame 20935: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▷ Ethernet II, Src: Dell_91:8d:61 (b8:ca:3a:91:8d:61), Dst: Dell_91:9b:93 (b8:ca:3a:91:9b:93)
▷ Internet Protocol Version 4, Src: 128.178.158.103, Dst: 128.178.158.107
▷ Transmission Control Protocol, Src Port: 36738, Dst Port: 22, Seq: 0, Len: 0

0000  b8 ca 3a 91 9b 93 b8 ca  3a 91 8d 61 08 00 45 00   ··:·····  :··a··E·
0010  00 3c 22 c1 40 00 40 06  d9 c3 80 b2 9e 67 80 b2   ·<"·@·@·  ·····g··
0020  9e 6b 8f 82 00 16 25 56  d4 4b 00 00 00 00 a0 02   ·k····%V  ·K······
0030  72 10 3e 66 00 00 02 04  05 b4 04 02 08 0a 6b d7   r·>f····  ······k·
0040  33 48 00 00 00 00 01 03  03 07                     3H······  ··

Transmission Control Protocol (tcp), 40 bytes      Packets: 33034 · Displayed: 62 (0.2%) · Dropped: 0 (0.0%)      Profile: Defau
```

Figure 7

## Executing commands remotely

Once you have connected to a remote machine, this launches a terminal prompt. You can use this prompt to issue commands to that machine, as if you had direct physical access to it.

If you are only interested in running a single command and then quitting the SSH session, you can specify that command directly to the ssh program:

```
$ ssh username\@hostname command
```

## When to use

- The latter way to invoke the ssh command is useful if you want to perform actions on a remote machine as part of an automated script. This capability becomes even more powerful if you have set up public key authentication (covered later in this Lab).

## Exercise

- Practice both ways to run a command on one of the machines at INF3 (e.g., the date command).

## Running graphical applications on a remote machine (X11 forwarding)

You can also use ssh to run graphical applications remotely. In order to do that, you need to instruct ssh to forward the display server session (X11) to your current machine. This is commonly referred to as X11 forwarding.

In order to perform X11 forwarding you must provide the right flag to the ssh command when you instruct it to connect to the target machine. Read the **man** page of ssh and find out the appropriate flag for X11 forwarding.

## When to use

- You don't have sufficient rights to install your favorite application on the machine you are using.
- The machine you are using is too weak to support the application you want to run. Thus, you are tapping in to the resources of the remote machine.

## Exercise

a. Connect to one of the INF3 machines (without X11 forwarding) and try to run a graphical application (e.g., firefox). What is the error message you get?

```
$ ssh icin3pc07.epfl.ch
...
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted
    ↪ by
applicable law.
Last login: Mon Dec 3 14:02:51 2018 from 128.178.158.103
icin3pc07:~$ firefox
Error: no DISPLAY environment variable specified
```

b. Reconnect to one of the INF3 machines with X11 forwarding enabled this time. Try to run the same graphical application, again.

```
$ ssh icin3pc07.epfl.ch
...
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted
    ↪ by
applicable law.
Last login: Mon Dec 3 14:02:51 2018 from 128.178.158.103
icin3pc07:~$ firefox
```

`firefox` is executed successfully (the window is open). This is because the `-Y` option enables X11 forwarding.

## Copying files using scp

The `scp` tool (Secure Copy) to copy files between two different hosts, just like the `cp` ↪ command is used to copy files between different directories. In fact, the two tools are very similar in syntax (e.g. the -r to copy directory subtrees recursively). The basic syntax for the `scp` command is:

```
$ scp file1 file2 ... username@hostname:/path/to/target/folder/
```

where ... means that you can specify multiple files to be copied from your machine (in the same way that **cp** can copy multiple files from the source directory).

## Exercise

- While connected to your machine, use scp to copy a file from your machine to one of the INF3 machines. To do that, you can create a few temporary files by using Linux's touch command. Moreover, do not forget that your files on an INF3 machine is kept only until your session is closed. Hence, copying files from your machine to one of the INF machines by scp will copy the files, but also delete right after the command is executed as the scp session completes right after the copying operation is done.

To see that the files are actually copied, you can open another terminal and ssh to your target machine. Then, you can do the copy operation from your host machine to the target machine by using the previous terminal you have opened. Lastly, you can go back to the terminal you newly opened, and check if the files are actually copied to the target machine (by ls command).

On the host machine (which is icin3pc03 in this example):

```
icin3pc03:~$ touch temp1
icin3pc03:~$ touch temp2
icin3pc03:~$ ls
Desktop Downloads temp1 temp2
icin3pc03:~$ scp temp1 temp2 icin3pc07.epfl.ch:~
username@icin3pc07.epfl.ch's password:
temp1 100 0 0.0KB/s 00:00temp2 100 0 0.0KB/s 00:00
```

On the target machine (which is icin3pc07 in this example):

```
icin3pc07:~$ cd ~/
icin3pc07:~$ ls
Desktop temp1 temp2
```

## SSH security

## Setting up public key authentication

By this time in the Lab, you might have realized that it can get quite tedious, having to retype your password every time you want to connect to a specific remote machine. Luckily, there is a way to automate the authentication process (i.e., so that you don't have to type your password every time) without giving up on security. You can achieve that using public key authentication.

## The Bigger Picture (and the theory behind it)

Let's say that your machine is Alice, and that the target machine (in this case, one of the INF3 machines) is Bob. Your goal is to set Alice and Bob up in a way such that Alice can securely authenticate herself to Bob every time she tries to connect (i.e., perform an **ssh connection**). We can use public key authentication to assist us with this task.

Let's assume that Alice has generated a pair of keys: the private key Ka- and the public key Ka+. Alice keeps Ka- (the private key) to herself. Alice, then, needs to notify Bob about Ka+ (the public key). Since both Alice and Bob are machines under our control, we can find a way to copy the public key from Alice to Bob in some secure way.

Finally, we instruct Bob to allow Alice to authenticate herself using public key authentication, (instead of checking whether Alice can provide the right password). Since Alice is the only machine with access to Ka- (the private key), we know that we can trust this method of authentication.

Note that if Eve manages to steal Ka- (e.g.., by stealing Alice's machine and reading the filesystem) she will also be able to authenticate herself as Alice. As an added measure of precaution, Alice could also encrypt Ka- using a passphrase. Now, when the user wants to make Alice connect to Bob, they will first have to provide the passphrase to decrypt Ka-.

From a usability perspective, this throws us back to the initial problem, where the user has to provide some kind of a password every time Alice tries to connect to Bob. Luckily, we can configure Alice in such a way, so that we have to decrypt the Ka- only once per session. For the reminder of the session, Alice can retain the decrypted key in memory without sacrificing security.

In the rest of this section, we are going to learn how to link security theory to SSH practice.

## Generating a key pair

We use the `ssh-keygen` command to generate a public-key/private-key pair for SSH. We recommend that you invoke the command `ssh-keygen` in the following way:

```
$ ssh-keygen -t rsa -b 4096
```

Once you invoke the command, you will presented with a prompt to provide:

- Where to store your key-pair. You should press ENTER, and the key-pair will be stored in files `/home/username/.ssh/id_rsa` (private key) and `/home/username/.ssh` ↪ `/id_rsa.pub` (public key).
- A passphrase can be used to encrypt your key. If you specify an empty passphrase, anybody that can read the private key file (e.g. the root) will be able to authenticate as you on your target machine. Instead, if you specify a non-empty passphrase, you need to provide the same passphrase every time you connect. We will examine a secure way to overcome this restriction.

## Exercise

- Generate a key pair using `ssh-keygen`. You should specify a non-empty passphrase.

```
icin3pc01:~$ ssh-keygen -t rsa Generating public/private rsa
key pair. Enter file in which to save the key (/home/user-
name/.ssh/id_rsa): Enter passphrase (empty for no passphrase): En-
ter same passphrase again: Your identification has been saved in
/home/username/.ssh/id_rsa. Your public key has been saved
in /home/username/.ssh/id_rsa.pub. The key fingerprint is:
f1:5f:2d:bd:ff:0b:f4:59:6b:f0:6e:f5:35:46:9a:7a username@icin3pc01 The
key's randomart image is: +–[ RSA 2048]—-+ | | | | | . | | o + | |
S . o * +| | . . + + + B| | o. . B = | | . E + . o | | . . o =
| +————————+
```

## Enabling public key authentication at the target

When your machine tries to connect to the target machine, the target machine checks whether it has enabled public key authentication for your machine. An SSH server performs this check by reading file `/home/username/.ssh/authorized_keys` (on the server's filesystem), if it exists, and checking whether it contains your machine's public key.

In order to enable the target machine to authenticate your machine using public key authentication, you should append the contents of your public key file (`/home/username` ↪ `/.ssh/id_rsa.pub` from your machine) to `/home/username/.ssh/authorized_keys` (on the target machine).

## Exercise

- Copy the contents of id_rsa.pub from your machine to authorized_keys on the target machine (e.g. by using scp or using a graphical text editor). If the .ssh folder or the authorized_keys file do not already exist, you will have to create them.
- Test whether public key authentication works. If you have set everything up properly by now, you will be asked to provide a passphrase when you connect to the target machine. Just like in Exercise 2.1.4, the machines in INF3 will delete your files after the latest live session has expired on the target machine. Hence, keep a live session to your target machine in another terminal to prevent the authorized_keys file to be deleted.

(where the host machine is icin3pc03 and the target machine is icin3pc07):

```
icin3pc03:~$ scp .ssh/id_rsa.pub icin3pc07.epfl.ch:~/tmpfile
icin3pc03:~$ ssh icin3pc07.epfl.ch
icin3pc07:~$ mkdir .ssh
icin3pc07:~$ cat tmpfile >> .ssh/authorized_keys
icin3pc07:~$ rm tmpfile
icin3pc07:~$ exit
icin3pc03:~$ ssh icin3pc07.epfl.ch
Enter passphrase for key '/home/sirin/.ssh/id_rsa':
```

As can be seen, at the last ssh, the passphrase is asked rather than the password.

## Remembering the decrypted private keys

Moving from password based authentication to public key authentication does not provide much improvement if you have to type the passphrase every time. Fortunately, the process can be automated with the help of **ssh-agent**, which remembers the passphrase and provides it to **ssh** the next time public key authentication is used. In the INF3 configuration, **ssh-agent** is enabled by default.

However, if you need to run **ssh-agent** manually, you can do so with the following commands:

```
$ ssh-agent $SHELL
$ ssh-add /home/username/.ssh/id_rsa
```

## Exercise

- Kill the **ssh-agent** process with the **pkill ssh-agent** command.
- Restart the **ssh-agent** process with the commands listed in this section, and give your passphrase. Then, try to connect to the target machine. If you have configured everything properly, you will manage to connect to the target machine without having to provide a password. Try to logout and reconnect to the target machine. You may verify that no password is needed.

```
icin3pc03:~$ ssh-agent $SHELL
icin3pc03:~$ ssh-add /home/username/.ssh/id_rsa
```

```
Enter passphrase for /home/username/.ssh/id_rsa:
Identity added: /home/username/.ssh/id_rsa
(/home/username/.ssh/id_rsa)
icin3pc03:~$ ssh icin3pc07.epfl.ch
Last login: Wed Dec 4 19:40:28 2013 from icin3pc03.epfl.ch
icin3pc03:~$ logout
icin3pc03:~$ ssh icin3pc07.epfl.ch
Last login: Wed Dec 4 19:40:28 2013 from icin3pc03.epfl.ch
```

## Fingerprints

So far, we have seen how Alice can authenticate herself to Bob. In order for this to be secure, Alice should also have some way to verify that she is authenticating herself to Bob, and not to Persa (somebody that tries to impersonate Bob).

## Known hosts

SSH handles this feature in the following way. When Alice tries to connect to Bob, Bob sends his public host key to Alice. `ssh` stores host key pairs in directory `/etc/ssh`.

If this is not the first time Alice connects to Bob, Alice will be able to recall Bob's public key from last time; she will only continue with the authentication if what she remembers about Bob matches what Bob told her about him. For the purposes of SSH, Alice keeps track of the public key she knows about Bob in file `/home/username/.ssh/known_hosts`,

## Exercise

- Check whether your machine trusts the target machine. You should open file `/`
  ↪ `home/username/.ssh/known_hosts` on your machine and compare it to the public host key file on the target machine. (`/etc/ssh/ssh_host_ecdsa_key.pub`).

> where the host machine is icin3pc03 and the target machine is icin3pc07:
>
> The `known_hosts` file contains a list of machines (and their public keys) that your machine trusts. A new machine is added to the `known_hosts` file the first time your machine connected to it (when you opted to trust the fingerprint they provided). Let compare the content of `.ssh/known_hosts`
> ↪ and `/etc/ssh/ssh_host_ecdsa_key.pub` files:

On the host machine:

```
icin3pc03:~$ cat .ssh/known_hosts
|1|iAjQZIQwePQRiK9DVNtJSnN4SqM=|dPx87G3xGFoIxC1mwo75kgapPBQ=
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBOrYzYnLvCV4PF5znA4I4aD7u7bEzYYkk42y02
    ↪ +5OlueBSE+D16L4A=
|1|6voBJfcWINRp3o84ZJjIIm2hTPo=|7pbFJrM9nXo5w+HjsSkwSpGGskw=
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBOrYzYnLvCV4PF5znA4I4aD7u7bEzYYkk42y02
    ↪ +5OlueBSE+D16L4A=
```

On the target machine:

```
icin3pc07:~$ cat /etc/ssh/ssh_host_ecdsa_key.pub
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBOrYzYnLvCV4PF5znA4I4aD7u7bEzYYkk42y02
    ↪ +5OlueBSE+D16L4A=
root@sysadmin-OptiPlex-9020
```

As can be seen, the key, which is

```
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBOrYzYnLvCV4PF5znA4I4aD7u7bEzYYkk42y02
    ↪ +5OlueBSE+D16L4A=
```

is the same in both files.

## Learning new hosts

If this is the first time that Alice connects to Bob, Alice will have to decide whether she trusts that the person she is receiving the public key from is indeed Bob. If she decides she wants to trust Bob, she will store his public key and continue with the authentication. SSH uses fingerprints to handle this process

A fingerprint is a summary of Bob's public key. The proper way to verify that the machine you are connecting to is indeed the machine you think it to be is to obtain the fingerprint in some other way (e.g. make a phone call to the administrator of that machine).

## Exercise

- Connect to the host machine. Once you connect to the host machine, execute the following command:

```
$ ssh-keygen -l -f /etc/ssh/ssh_host\_ecdsa\_key.pub
```

Save the output to somewhere, and go back to your host machine (either by exiting from the ssh connection, or opening a new terminal on your local machine).

- Make your machine forget all known hosts it has encountered (i.e., delete file `/home ↪ /username/.ssh/known_hosts` on your local machine). Then, reconnect to the host machine and check whether the fingerprint it advertises to you, matches the one you obtained in the previous step.

(where icin3pc03 is the host and icin3pc07 is the target machine)

On the target machine:

```
icin3pc07:~$ ssh-keygen -l -f
/etc/ssh/ssh_host_ecdsa_key.pub
256 SHA256:0cKucqt2PXxw0nNwmTqJzeqJk6mZ/qu2XoU4LL+AeXg
root@sysadmin-OptiPlex-9020 (ECDSA)
```

On the host machine:

```
icin3pc03:~$ ssh icin3pc07
The authenticity of host 'icin3pc07 (128.178.158.107)' can't be
established.
ECDSA key fingerprint is
SHA256:0cKucqt2PXxw0nNwmTqJzeqJk6mZ/qu2XoU4LL+AeXg.
Are you sure you want to continue connecting (yes/no)?
```

As can be seen, the fingerprints, which is `SHA256:0 ↪ cKucqt2PXxw0nNwmTqJzeqJk6mZ/qu2XoU4LL+AeXg`, are the same. However, verifying the fingerprint does not offer good security. A non-naive attacker, mounting a man in the middle attack, can provide the victim with a fake fingerprint. The proper way to verify a public key is to get the fingerprint of the machine you want is to obtain it by some other channel (e.g., call the administrator of that machine).

## Food for thought

Now that you know a bit more about the inner workings of the `ssh` tool, you should be able to answer the following questions:

## Questions

1. Compare the security of using public key authentication to using passwords in the following scenario:

Suppose that you own two laptops (laptop1 and laptop2), which you use to connect to your server. At some point, laptop1 gets stolen, and the attacker is able to deduce the password and/or the private key of the laptop (depending on which authentication message is used) from information stored on the filesystem.

a. If you were using password-based authentication, what steps would you take to prevent that attacker from accessing your server. Would that change anything in the way that you are using laptop2 to access the server?

b. How would your response change if you were using public-key authentication to access your server? Would you have to change anything on laptop2?

If somebody managed to steal the password you were using to access your server, you should change the password for that account on your server. This would imply that every time you wanted to connect to your server from laptop2, you should provide the new password. Some users would find that annoying.

Instead, if somebody stole laptop1's private key, all you need to do is remove a single line in the configuration of your server. You need to remove the line in the **authorized_keys** file which makes your server trust the public key of laptop1. You wouldn't have to make any changes to laptop2 to keep accessing your server after that change.

2. Use `ssh` in the verbose mode (option `-v`). Describe the message exchange.

*Solution (where the host machine is icin3pc03 and the target machine icin3pc07):*

```
sirin@icin3pc03:~$ ssh -v icin3pc07.epfl.ch
OpenSSH_7.2p2 Ubuntu-4ubuntu2.6, OpenSSL 1.0.2g 1 Mar 2016
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 19: Applying options for *
debug1: Connecting to icin3pc07.epfl.ch [128.178.158.107] port
    ↪ 22.
debug1: Connection established.
debug1: identity file /home/sirin/.ssh/id_rsa type 1
debug1: key_load_public: No such file or directory
debug1: identity file /home/sirin/.ssh/id_rsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/sirin/.ssh/id_dsa type -1
```

```
debug1: key_load_public: No such file or directory
debug1: identity file /home/sirin/.ssh/id_dsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/sirin/.ssh/id_ecdsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/sirin/.ssh/id_ecdsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/sirin/.ssh/id_ed25519 type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/sirin/.ssh/id_ed25519-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_7.2p2
Ubuntu-4ubuntu2.6
```

Checking the presence of private keys for public-key authentication.*

```
debug1: Remote protocol version 2.0, remote software version
OpenSSH_7.2p2 Ubuntu-4ubuntu2.6
debug1: match: OpenSSH_7.2p2 Ubuntu-4ubuntu2.6 pat OpenSSH*
    ↪ compat
0x04000000
debug1: Authenticating to icin3pc07.epfl.ch:22 as 'sirin'
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
```

Negotiating cryptographic algorithms.

```
debug1: kex: algorithm: curve25519-sha256@libssh.org
debug1: kex: host key algorithm: ecdsa-sha2-nistp256
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com
MAC: compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com
MAC: compression: none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
```

Generating a shared secret using the Diffie-Hellman algorithm. The last message includes the public key of the remote party and a signature.

```
debug1: Server host key: ecdsa-sha2-nistp256
SHA256:0cKucqt2PXxw0nNwmTqJzeqJk6mZ/qu2XoU4LL+AeXg
debug1: Host 'icin3pc07.epfl.ch' is known and matches the ECDSA
    ↪ host
key.
debug1: Found key in /home/sirin/.ssh/known_hosts:1
debug1: rekey after 134217728 blocks
```

The public key of the remote party is recognized.

```
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
```

Form now on, the new keys generated based on the shared secret will be used.

```
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=
debug1: SSH2_MSG_SERVICE_ACCEPT received
```

Requesting user authentication.

```
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/sirin/.ssh/id_rsa
debug1: Server accepts key: pkalg rsa-sha2-512 blen 279
debug1: Authentication succeeded (publickey).
Authenticated to icin3pc07.epfl.ch ([128.178.158.107]:22).
```

Public-key authentication successful.

```
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: pledge: network
debug1: client_input_global_request: rtype hostkeys-00@openssh.
    ↪ com
want_reply 0
debug1: Sending environment.
debug1: Sending env LC_PAPER = de_CH.UTF-8
debug1: Sending env LC_ADDRESS = de_CH.UTF-8
debug1: Sending env LC_MONETARY = de_CH.UTF-8
debug1: Sending env LC_NUMERIC = de_CH.UTF-8
debug1: Sending env LC_TELEPHONE = de_CH.UTF-8
debug1: Sending env LC_IDENTIFICATION = de_CH.UTF-8
debug1: Sending env LANG = en_US.UTF-8
debug1: Sending env LC_MEASUREMENT = de_CH.UTF-8
debug1: Sending env LC_TIME = de_CH.UTF-8
debug1: Sending env LC_NAME = de_CH.UTF-8
Last login: Mon Dec 3 17:51:23 2018 from 128.178.158.103
icin3pc07:~$
```