LPIC-1 102-400 – Lesson 20

110.3 Securing data with encryption

The *SSH* (Secure Shell) service

- Traditionally the **TELNET** service was used for remote shell access to other network nodes. This service is insecure by design, because all information is send in cleartext. No encryption takes place. This a serious liability in system security
- The **SSH** service has replaced TELNET in modern systems because it provides Public-Key Cryptography and thus, secure communications. There are two versions: version 1 and version 2. The first version does not provide sufficient security by modern standards, so the use of version 2 is recommended.
- The **ssh** command is the client that connects to the **sshd** for shell access
- There also the **scp** command for secure, network file transfer

Connect to other network nodes with `ssh`

- The **ssh** command is the client side of an **SSH** system and used for connecting to other systems over network
- `$ ssh user1@example.com # = ssh -l user1 example.com`

```
The authenticity of host 'example.com (10.0.1.50)' can't
  be established.
RSA key fingerprint is
  47:e2:fd:2d:62:b8:b4:37:66:b2:c2:d1:59:a5:ab:98.
Are you sure you want to continue connecting (yes/no)?
```

- By answering "**yes**" in the question above, the remote host's public key will be saved permanently to the **~/.ssh/known_hosts** file and you will not be asked again about this host. If you answer "**no**" then the connection is dropped and the **~/.ssh/known_hosts** file is not updated

The configuration files for ssh and sshd

- **/etc/ssh/ssh-config** # configuration file for the **ssh** client
  Port 22 # the default ssh port
  Protocol 2 # connect to SSH hosts using version 2 only!

- **/etc/ssh/sshd-config** # the **sshd** daemon configuration file
  PermitRootLogin yes # set to **no** for disabling remote root
  # login
  PubkeyAuthentication yes # activate public key
  # authentication
  PasswordAuthentication no # disable password
  # authentication
  Protocol 2 # use SSH version 2 for connections
  X11Forwarding yes # support executing graphical applications via SSH. Disable it if you don’t need it

- **/etc/ssh_known_hosts** or **/etc/ssh/ssh_known_hosts** # global file for known hosts to SSH

Secure file transfer with `scp`

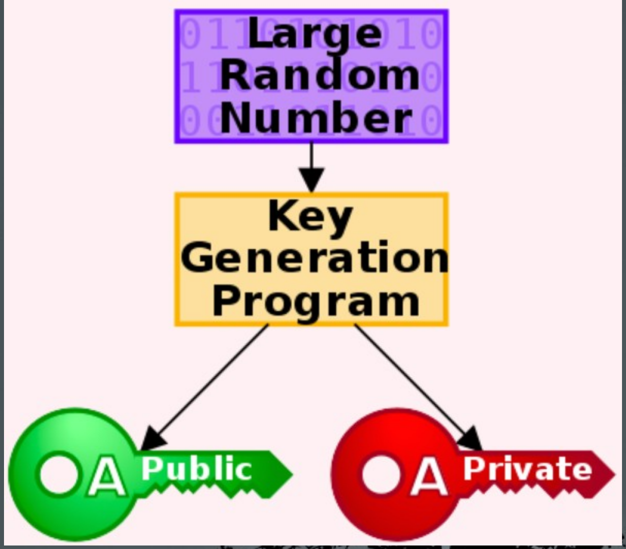
- The **scp** command is used for the secure transfer of files from the local host to the remote and vice-versa

- **$ scp mydoc.odt user1@example.com:** # copy file **mydoc.odt** to the home directory of **user1**, in the **example.com** server. The ":" is very important because without that, **scp** behaves like a local **cp**
- **$ scp mydoc.odt user1@example.com:Documents** # copy local file **mydoc.odt** to directory **Documents** under the remote home directory of **user1**
- **$ scp user1@example.com:Documents/mydoc.odt .** # copy remote **mydoc.odt** from **Documents** on the remote home directory of **user1** on **example.com**, to the local working directory

Public Key Cryptography

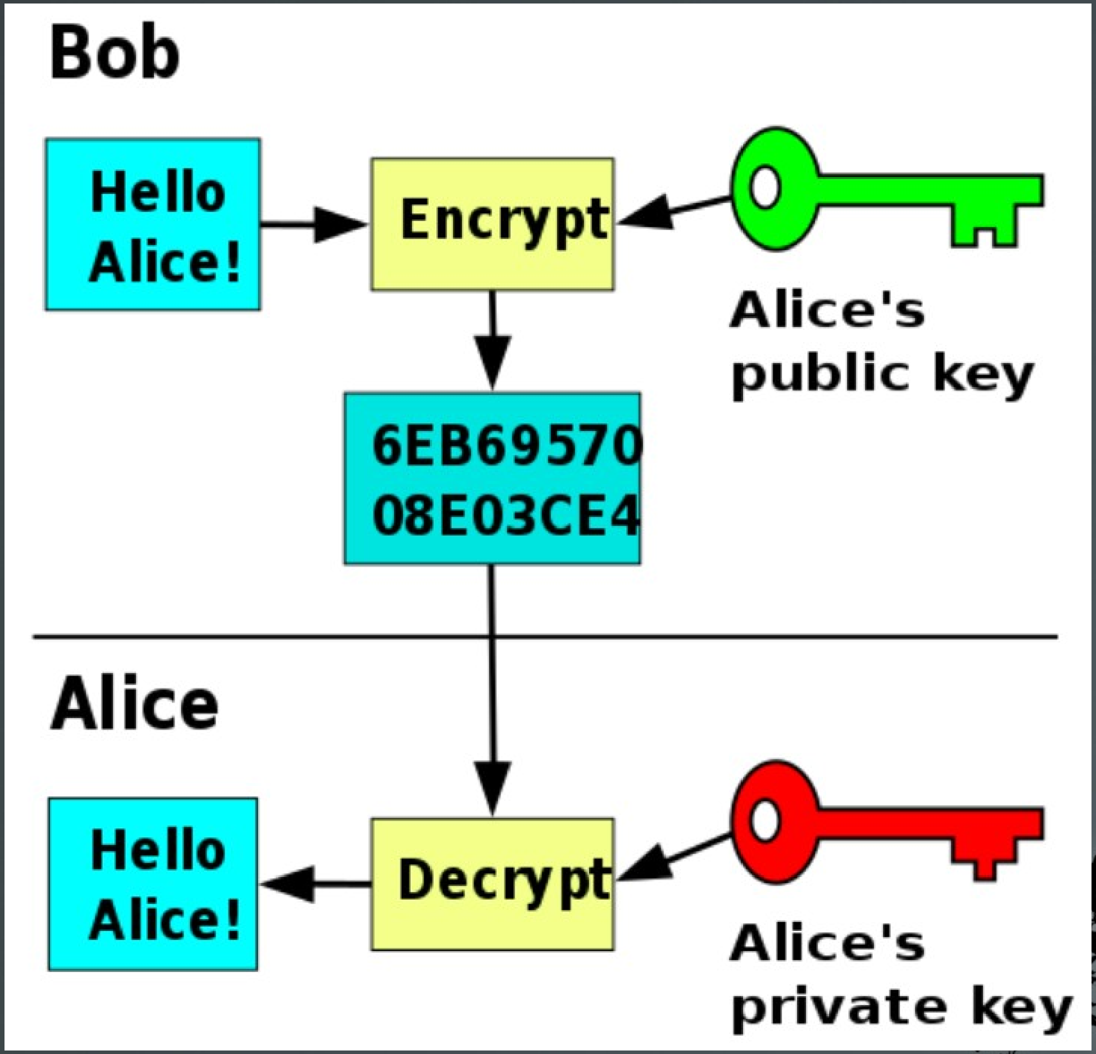
- Public-Key Cryptography is an asymmetric cryptography technique used by SSH, SSL, PGP, GPG etc
- Some of the encryption algorithms are RSA, DSA and ECDSA
- A key generator, produces two keys where we can use the one key to encrypt cleartext data and the other to decrypt encrypted data. One key assumes the role of private key, and the other key is the public. The public key can be shared to anybody while the private key must be protected.

Image
User:KohanX
Wikipedia



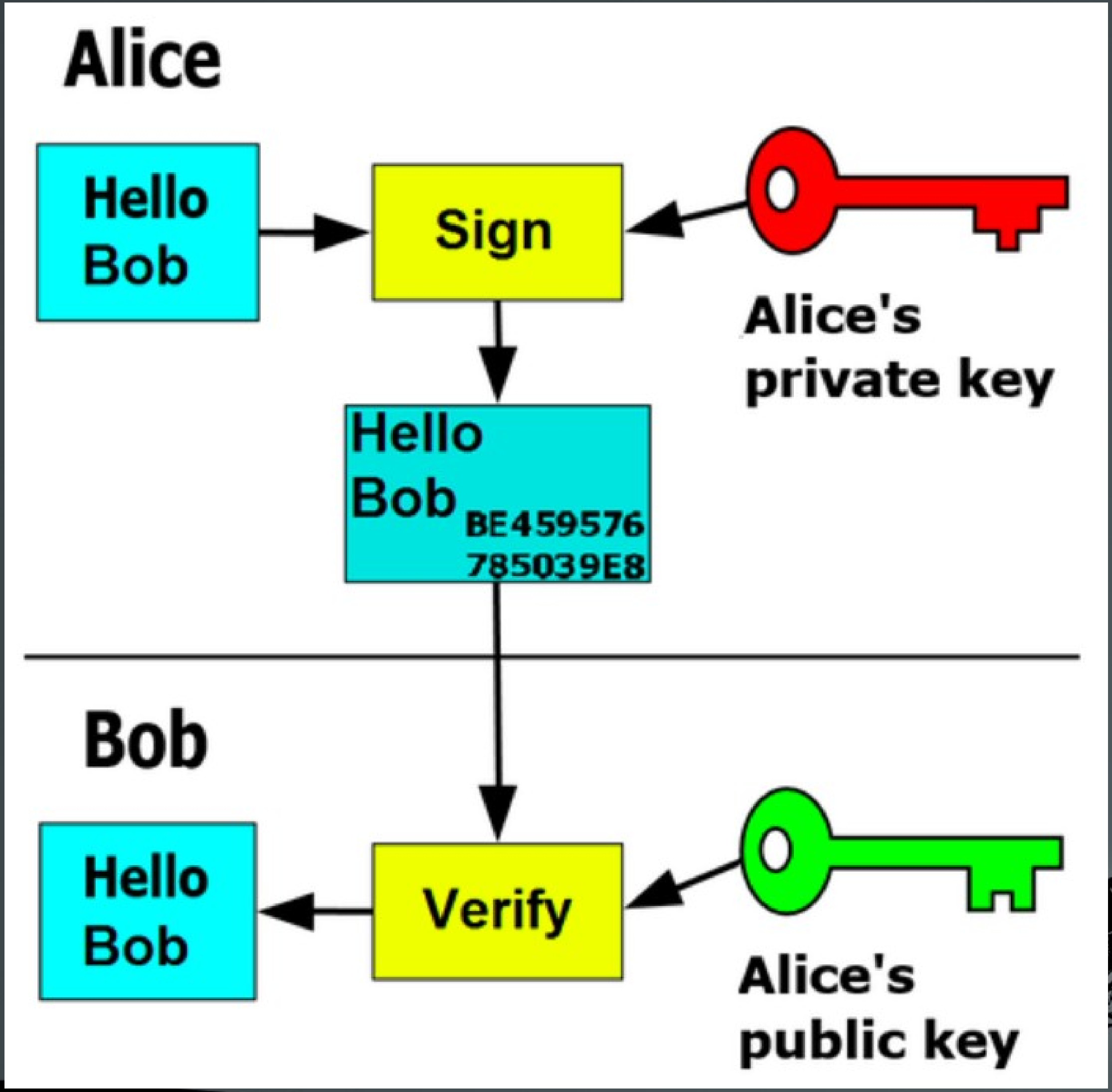
Public Key Cryptography

Provides confidentiality but not non-repudiability (signing)



Public Key Cryptography

Provides non-repudiability (signing) but not confidentiality



Public and Private SSH keys

- During the initialization of the **sshd** daemon the host ssh private and public keys will be created. The default algorithm use is RSA

```
# ls -la /etc/ssh/*key*
-rw-------   1 root root    227 Jun 29 15:58 ssh_host_ecdsa_key
-rw-r--r--   1 root root    177 Jun 29 15:58 ssh_host_ecdsa_key.pub
-rw-------   1 root root    411 Jun 29 15:58 ssh_host_ed25519_key
-rw-r--r--   1 root root     97 Jun 29 15:58 ssh_host_ed25519_key.pub
-rw-------   1 root root   1679 Jun 29 15:58 ssh_host_rsa_key
-rw-r--r--   1 root root    397 Jun 29 15:58 ssh_host_rsa_key.pub
```

- When we connect to an SSH service we get its public RSA key so the SSH client can use that to encrypt the connection
- When you connect to a host for the first time, you are prompted to accept the public ssh key, and if you accept it this will be saved in the ~/**.ssh/known_hosts** file for reuse
- If the SSH public key of a host changes, the system will issue a strict warning and refuse to connect until you delete the old key from **~/.ssh/known_hosts**

SSH Public Key Authentication

- SSH provides various means of authentication. Besides the traditional username and password authentication, there is also the Public Key Authentication which is emerging as the best practice for secure SSH connections

-
  ```
  user1@local:~$ whoami # local user is user1
    user1
  ```

-
  ```
  user1@local:~$ ssh-keygen -t rsa -b 2048 # generate user ssh
                                            # key pair
  ```

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user1/.ssh/id_rsa):
Created directory '/home/user1/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user1/.ssh/id_rsa.
Your public key has been saved in /home/user1/.ssh/id_rsa.pub.
The key fingerprint is:
83:40:c4:05:39:d8:58:c0:ed:d4:a0:40:6d:87:6c:a4 user1@local
```

SSH Public Key Authentication

-
  ```
  user1@local:~$ ls -l .ssh/ # private and
                             # public keys of user1
  ```
```
-rw------- 1 user1 user1 1679 Apr  4 22:35 id_rsa
-rw-r--r-- 1 user1 user1  400 Apr  4 22:35 id_rsa.pub
```

-
  ```
  user1@local:~$ cat .ssh/id_rsa.pub | ssh \
    user2@remote.dom "xargs -I {}  echo {} >> \
     .ssh/authorized_keys"  # append the
                            # user1@local.dom pubkey
                            # to the
                            # .ssh/authorized_keys
                            # file of user
                            # user2@remote.dom
  ```
```
user2@remote.dom's password: # enter user2 password
```

-
  ```
  ssh user2@remote.dom # ssh without password!
  ```
```
Last login: Wed Apr  4 22:58:35 2012 from local.dom
user2@remote:~$
```

-
  ```
  user2@remote:~$ whoami # verify
    user2
  ```

SSH Public Key Authentication

-
  ```
  user2@remote:~$ grep user1 .ssh/authorized_keys #
                                      # check authorized keys
    ssh-rsa
    AAAAB3NzaC1yc2EAAAADAQABAAABAQDBAfRMHpzJ6NfnOCcB
    OjE5Xxip03eHbIIDGnrpmyc8fWjGqwN3mZZ3HzJ2fNJZJA6r
    dMEtlcGt1MgcPcLUqLx93jZr/
    ZL3Me3d9e9JretivjcicFV4gU/
    2m3pQHy1aKvyioGqytmtUKwEZzJzC+nZNK/
    Fd+glMUu6q8Py3QFspPBb1NEw7cKgYKn5kOcV3Th4KAvYwzo
    +VlHuHIS0MlffGDxId4m7C+DqMX1utdUJ7reYAGNWFlSAh7a
    jqVNDtOGAQC743BOBuyTdPrnLtFc1A45mR2l2P9PU4iqYhiY
    pKLOxqK6oeQIcomq/KcCz1+HPoYWPDq2EB5CWAmaKLNQcb
    user1@local
  ```
- After the **Public Key Authentication** method is enabled it is recommended to disable the password authentication, to mitigate SSH dictionary/brute-force attacks. Because no password is required, this method is also useful for executing scripts from one server to another (e.g. backup scripts), for cluster systems or merely for convenience (password fatigue)

The `ssh-keygen` command

- The **ssh-keygen** command is used for generating an ssh public/private key pair for the **ssh** client or the **sshd** daemon
-
  ```
  $ ssh-keygen -l -f ~/.ssh/id_rsa # show key info
    2048
    a5:c6:87:06:ea:d6:09:f6:4d:a9:25:31:e4:a0:fb:df
    .ssh/id_rsa.pub (RSA)
  ```

<u>**Options**</u>:

- `-b # number of bits. Default is 2048. For new systems 4096 is recommended`
- `-p # change private key encryption passphrase`
- `-f file # set output file`
- `-C 'some comments' # set key comments`
- `-P 'oldlongpassphrase' # old passphrase`
- `-N 'newlongpassphrase' # new passphrase`

Enhance security with `ssh-agent`

- The problem that occurs if we use an unencrypted key, is that if our local system is hijacked then the attacker can login on our remote hosts effortlessly!
- If we use a passphrase during the key generation this cannot happen, but we have to enter the passphrase on every ssh connection!
- The **ssh-agent** provides a convenient way to use encrypted keys by providing the passphrase once, at the first login, and re-using that for the whole session and resulting child shells

Enhance security with `ssh-agent`

- **$ ssh-keygen -t rsa -b 2048** # generate key pair

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user1/.ssh/id_rsa):
Created directory '/home/user1/.ssh'.
Enter passphrase (empty for no passphrase): MySecret
Enter same passphrase again: MySecret
Your identification has been saved in /home/user1/.ssh/id_rsa.
Your public key has been saved in /home/user1/.ssh/id_rsa.pub.
The key fingerprint is:
83:40:c4:05:39:d8:58:c0:ed:d4:a0:40:6d:87:6c:a4 user1@mypc
```

- **$ ssh-agent /bin/bash** # activate **ssh-agent** in a new shell. To activate it on a running shell use should use the command **'eval ssh-agent'**
- **$ ssh-add ~/.ssh/id_rsa** # add key in **ssh-agent**. If you simply run '**ssh-add**' all keys will be added. You only need to do this once.

SSH Port channels (Tunneling)

- **$ ssh -X user@10.0.1.50** # allows executing graphical commands on 10.0.1.50 and having the graphical window on the local machine (X11Forwarding must be set to yes).

- **$ ssh -N -f -L 2525:smtp.example.com:25 bob@gate.example.com** # forward the local **2525** port to the remote **25** port of the server **smtp.example.com** using an intermediate proxy **gate.example.com**

- **$ telnet localhost 2525** # this will lead to the **25** of the server **smtp.example.com**

- **$ ssh -L 3306:localhost:3306 bob@mysql.example.com**

  # forward the local **3306** port to the remote **3306** port of **mysql.example.com**

- Port forwarding can be a security issue in some environments and so it can be disabled with **AllowTcpForwarding no**

`ssh` options

- **-l** # set user name
- **-X** # execute graphical program from the remote machine to the local X server
- **-L** # connect a local port to a remote
- **-R** # connect a remote port to a local
- **-N** # do not execute a remote command (e.g. bash)
- **-f** # send ssh process to the background
- **-v** # verbose output (useful for debugging)

The `gpg` encryption and signing utility

- The **GPG** (GNU Privacy Guard) utility is used as an encryption and signing tool for files and emails
- It uses mainly Public Key Cryptography and it was designed as an alternative to the proprietary PGP (Pretty Good Privacy)
- It can be used as a standalone utility or be integrated with other applications like email clients

The `gpg` encryption and signing tool

- **$ gpg –gen-key** # generate a **GPG** key pair

```
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? 4
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 4096
Requested keysize is 4096 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) 5y
Key expires at Tue 04 Apr 2023 12:52:36 AM EEST
Is this correct? (y/N) y
```

The `gpg` encryption and signing tool

```
Real name: Bob Crypt
Email address: bob.crypt@example.com
Comment: Bob the One
You selected this USER-ID:
    "Bob Crypt (Bob the One) <bob.crypt@example.com>"

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

gpg: gpg-agent is not available in this session

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.
```

The `gpg` encryption and signing tool

```
...+++++
....+++++
gpg: /home/bob/.gnupg/trustdb.gpg: trustdb created
gpg: key 1C877AA9 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 2017-04-03
pub   4096R/1C877AA9 2012-04-04 [expires: 2023-04-03]
      Key fingerprint = 537D E04B 6852 4F7E 5880  AFAC E49A 1815 1C87 7AA9
uid                  Bob Crypt (Bob the one) <bob.crypt@example.com>

Note that this key cannot be used for encryption.  You may want to use
the command "--edit-key" to generate a subkey for this purpose.
```

The `gpg` encryption and signing tool

- **$ ls -la .gnupg/**

```
total 1736
drwx------  5 user1 user1   4096 Aug 26 10:09 .
drwxr-xr-x 31 user1 user1   4096 Aug 22 23:02 ..
drwx------  2 user1 user1   4096 Jun 16  2016 crls.d
-rw-rw-r--  1 user1 user1      0 Jun 16  2016 .gpg-v21-migrated
drwx------  2 user1 user1   4096 Jun 16  2016 openpgp-revocs.d
drwx------  2 user1 user1   4096 Jun 16  2016 private-keys-v1.d
-rw-------  1 user1 user1 845144 Aug 23 01:07 pubring.gpg
-rw-------  1 user1 user1 844721 Aug 22 23:13 pubring.gpg~
-rw-------  1 user1 user1    600 Aug 25 18:25 random_seed
-rw-------  1 user1 user1      0 Apr 22  2016 secring.gpg
-rw-r--r--  1 user1 user1  49152 Aug  1 04:15 tofu.db
-rw-------  1 user1 user1   7040 Aug  1 04:15 trustdb.gpg
```

Other `gpg` functions

- **$ gpg --import user_test_example.asc** # import a public key of another user from an **.asc** file
- **$ gpg --edit-key "User.test"** # sign an imported key with our key.
- **$ gpg –list-keys** # list of personal and imported keys
- **$ gpg --export my_gpg_key_backup** # extract your key for backup
- **$ gpg -e -u "Bob Crypt" -r "User Test" mydoc.odt** # encrypt file so only **User Test** can open it
- **$ gpg -d mydoc.odt** # decrypt the **mydoc.odt** file from **User Test**

License