Securing Your System

Objectives:

- ✓ 110.1 Perform security administration tasks
- ✓ 110.2 Set up host security
- ✓ 110.3 Securing data with encryption

ADMINISTERING NETWORK SECURITY

Imagine a home with a door to the outside from every room in the house. That would be a lot of locks to check before leaving the house or going to bed. In the world of network security, having services on your system that are not used is similar to having unnecessary doors to the outside. A term used in cybersecurity that applies to this scenario is attack surface. An attack surface is all the various points where a malicious person may try to gain access to something for nefarious reasons.

Continuing the analogy, envision that the multidoor home has a few doors with old locks that no longer properly work and need to be replaced or repaired. This situation is similar to older security software on your system that needs to be updated, reconfigured, or even ousted so that more modern and secure applications can take its place.

In this section, we’ll take a look at auditing your system for unused services in order to minimize its attack surface. We’ll also look at some older network security technology and how it should be handled.

Disabling Unused Services

- One method for minimizing your system’s attack surface is by disabling unused services (daemons) on your system.
- Having less software running lessens the various targets a malicious actor (another name for an attacker) can leverage.
- While it is tempting to think you know every service running on your Linux system, to be sure, it’s always a good idea to run a thorough audit.

Discovering Open Ports with nmap

- The Network Mapper (nmap) utility is often used for penetration testing (the practice of testing a system, its network, and its apps to find computer security weaknesses that a malicious attacker could exploit).
- However, it is also very useful for network service auditing.

Discovering Open Ports with nmap

- **nmap** - Network exploration tool and security / port scanner

```
nmap [Scan Type...] [Options] {target specification}
```

- ✓ -sT (TCP connect scan)

```
$ nmap -sT 127.0.0.1
$ nmap -sT 8.8.8.8
```

- ✓ -sU (UDP connect scan)

```
# nmap -sU 127.0.0.1
# nmap -sT 8.8.8.8
```

Identifying Open Ports with netstat

❖ **netstat** - Print network connections, routing tables, interface statistics, masquerade connections, and multicast memberships

```
netstat [OPTION...]
```

✓ `-l, --listening`	display listening server sockets
✓ `-t, --tcp`	limit the output to just TCP connections

```
$ netstat --tcp --listening
```

✓ `-u, --udp`	limit the output to just UDP connections

```
$ netstat --udp --listening
```

✓ `-s, --statistics`	display networking statistics (like SNMP)

Surveying Network Sockets via ss and systemd.socket

- ss - another utility to investigate sockets

```
ss [options] [ FILTER ]
```

✓-l, --listening	Display only listening sockets.
✓-t, --tcp	Display TCP sockets.
✓-u, --udp	Display UDP sockets.

```
$ ss -ltu
```

Systemd Managed Socket Unit Files

- To find the potential network socket systemd configuration (unit) files, employ a different `systemctl` command.

```
$ systemctl list-unit-files --type=socket --no-pager
```

  - ✓ A disabled value only means systemd does not manage that particular network socket;
    - it does not mean the service is disabled.

- Take a look at the contents of each enabled network socket's unit file.

```
$ systemctl cat rpcbind.socket
```

Auditing Open Files with lsof and fuser

- **lsof** - list open files

```
lsof [OPTIONS...]
```

- ✓ Since Linux treats network connections and sockets as files, they will appear in the lsof output list.
- ✓ **Displaying UDP open files**

```
# lsof -iUDP
```

- ✓ **Displaying TCP listening open files**

```
# lsof -iTCP -sTCP:LISTEN
```

- ✓ **Displaying open files for a particular protocol**

```
# lsof -i tcp:22
```

Auditing Open Files with lsof and fuser

- **fuser** - identify processes using files or sockets

```
fuser [OPTIONS...]
```

- ✓ Display a program or user's process ID (PID) that is employing the protocol and port.
- ✓ -n SPACE, --namespace SPACE
  - Select a different name space. The name spaces file (file names, the default), udp (local UDP ports), and tcp (local TCP ports) are supported.
- ✓ -v, --verbose
  - Verbose mode.

```
# fuser -vn tcp 22
# fuser -vn udp 822
```

Disabling the Services

- First, stop the service, if it is running, and check that it is stopped.

```
$ systemctl stop SERVICE-NAME          $ service SERVICE-NAME stop
$ systemctl status SERVICE-NAME        $ service SERVICE-NAME status
```

- When the service is stopped, be sure to disable it, employing super user privileges, so that when the system reboots, it won't start.

```
$ systemctl disable SERVICE-NAME         $ chkconfig SERVICE-NAME off
$ systemctl is-enabled SERVICE-NAME      $ chkconfig --list SERVICE-NAME
```

- To disable a network service on a Debian-based distro, use super user privileges and the following command:

```
$ update-rc.d -f SERVICE-NAME remove
```

Using Super Server Restrictions

- Typically, when a network service (daemon) starts, such as chronyd, it opens a port.
  - ✓ You can think of a port number as an identification number assigned to a network service.
- Incoming network packets that contain the network service's port number trigger that service into action.
  - ✓ While the network service is waiting for packets containing its port number, it is in a wait state, which is called listening.
- Instead of listening directly on a port, some network services can employ a super server (also called a super daemon) to act as a guard for them.
  - ✓ Instead of the network service, the super server directly listens for packets containing the designated port number.
- When such a packet comes into the system, after initial processing the super server starts the network service and hands the packet off to it.
- When concurrent requests come in, the super server can start additional network service processes as required.
  - ✓ Using this method systems boot faster, because network services are not started until needed.
  - ✓ Also, you can put additional controls into place.
    - For example, you can set limits on network service use, and you can fine-tune connection logging

Using Super Server Restrictions

- **Configuring `xinetd`**
  - ✓ `inetd` was the original super server on Linux systems.
  - ✓ `xinetd`, (extended super daemon) has additional security features.
  - ✓ The primary configuration file: `/etc/xinetd.conf`.
    - This file typically contains only global default options.
    - any line with a preceding pound sign (#) is a comment line and thus inactive.
- **Configuring `xinetd` Services**
  - ✓ The default options in the `/etc/xinetd.conf` file can be overridden by settings in a network service's configuration file stored in the `/etc/xinetd.d/` directory.
    - Notice in the file that there are mandatory settings for each service, such as id.

Using Super Server Restrictions

Name	Description
cps	Specifies the maximum rate of incoming connections to the network service. The first number sets the maximum rate before the service pauses, and the second number establishes the number of seconds to pause.
instances	Sets the maximum number of service processes that can be active at the same time. Set to UNLIMITED to allow no limits.
logtype	Determines where log messages are sent. If set to SYSLOG, log messages are sent to the syslog protocol application, and the syslog facility and severity must be set as well. If set to FILE, log messages are appended to the filename listed.
log_on_failure	Establishes what additional information, besides the service ID, is logged when a server process cannot be started. The data that can be included is HOST, USERID, and ATTEMPT.
log_on_success	Sets what information is logged when a server process is started and when it exits. The data that can be included is PID, HOST, USERID, EXIT, DURATION, and TRAFFIC.
max_load	Determines the one-minute load average that when reached, the network service stops accepting connections, until the load level drops.
no_access	Establishes remote hosts banned from this network service.
only_from	Sets the remote hosts or subnets which may use this network service.
per_source	Specifies the maximum number of network server processes that can be started per IP address.

Using Super Server Restrictions

```
service ssh
{
 disable     = no
 socket_type = stream
 protocol    = tcp
 port        = 22
 wait        = no
 user        = root
 server      = /usr/sbin/sshd
 server_args = -I
 only_from   = 192.168.1.100
}
```

Restricting via TCP Wrappers

- TCP Wrappers are an older method for controlling access to network services.
- If a service can employ TCP Wrappers, it will have the `libwrap` library compiled with it.
  - ✓ You can check for support by using the `ldd` command.

```
$ which sshd
/usr/sbin/sshd
$ ldd /usr/sbin/sshd | grep libwrap
libwrap.so.0 […]
```

  - ✓ TCP Wrappers employ two files to determine who can access a particular service.
  - ✓ These files are `/etc/hosts.allow` and `/etc/hosts.deny`.
    - The `hosts.allow` file typically allows access to the designated service in the form of a whitelist,
    - whereas the `hosts.deny` file commonly blocks access to all addresses included in a blacklist.
    - These files have simple record syntax:

```
SERVICE: IPADDRESS…
```

Restricting via TCP Wrappers

- The search order of these files is critical.
  - ✓ The `hosts.allow` file is checked for the remote IP address or hostname.
    - If found, access is allowed, and no further checks are made.
  - ✓ The `hosts.deny` file is checked for the remote address.
    - If found, access is denied.
    - If not found, access is allowed.
- It is best to employ the `ALL` wildcard in the `/etc/hosts.deny` file:

```
ALL: ALL
```

- This disables all access to all services for any IP address not listed in the `/etc/hosts.allow` file.
- The record's `IPADDRESS` can be either IPv4 or IPv6.

```
sshd: 172.243.24.15, 172.243.24.16, 172.243.24.17
```

- Typing in every single IP address that is allowed to access the OpenSSH service is not necessary.
  - ✓ You can specify entire subnets.

```
sshd: 172.243.24.
```

ADMINISTERING LOCAL SECURITY

Part of securing your system is dealing with local security. This includes understanding password protections and making sure they are properly enforced. A rather important security measure, which sometimes goes unmanaged, is correctly enforcing secure root account access. These topics, along with setting limits and locating potentially dangerous files on your system, are covered in this section.

Securing Passwords

- Looking at password storage
  - ✓ /etc/passwd file on early Linux systems

```
$ ls -l /etc/passwd
-rw-r--r--. 1 root root 2631 Jun 12 18:04 /etc/passwd
```

  - ✓ rainbow tables
  - ✓ /etc/shadow file

```
$ ls -l /etc/shadow
----------. 1 root root 2143 Jun 12 18:04 /etc/shadow
```

  - ✓ Use pwconv utility to move passwords in the /etc/passwd file to the /etc/shadow file
  - ✓ SUID permission set on the passwd program's file

```
$ which passwd
/usr/bin/passwd
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 59640 Jan 25 2018 /usr/bin/passwd
```

Securing Passwords

- Dealing with password problems
  - ✓ Use either the grep or `getent` command to check the `/etc/passwd` and `/etc/shadow` file records.

```
$ sudo getent passwd mohsen
mohsen:x:1002:1002::/home/JKirk:/bin/bash
$ sudo getent shadow mohsen
mohsen:!:17806:0:99999:7:::
```

  - ✓ Usernames are case sensitive on linux
  - ✓ Determine if the account is locked via `passwd -S`
    - To unlock the account use `usermod -U` or `passwd -u`
  - ✓ Checking if an account is expired with the `chage` command

```
$ sudo chage -l Jarcher
```

    - set a new expiration date for the account via `chage -E` command

Manage Accounts

- What to do in order to properly manage accounts:
  - ✓Do not permit logins to the root user account.
  - ✓Allow only one user per user account.
  - ✓Set expiration dates on temporary user accounts.
  - ✓Remove unused user accounts.

Limiting root Access

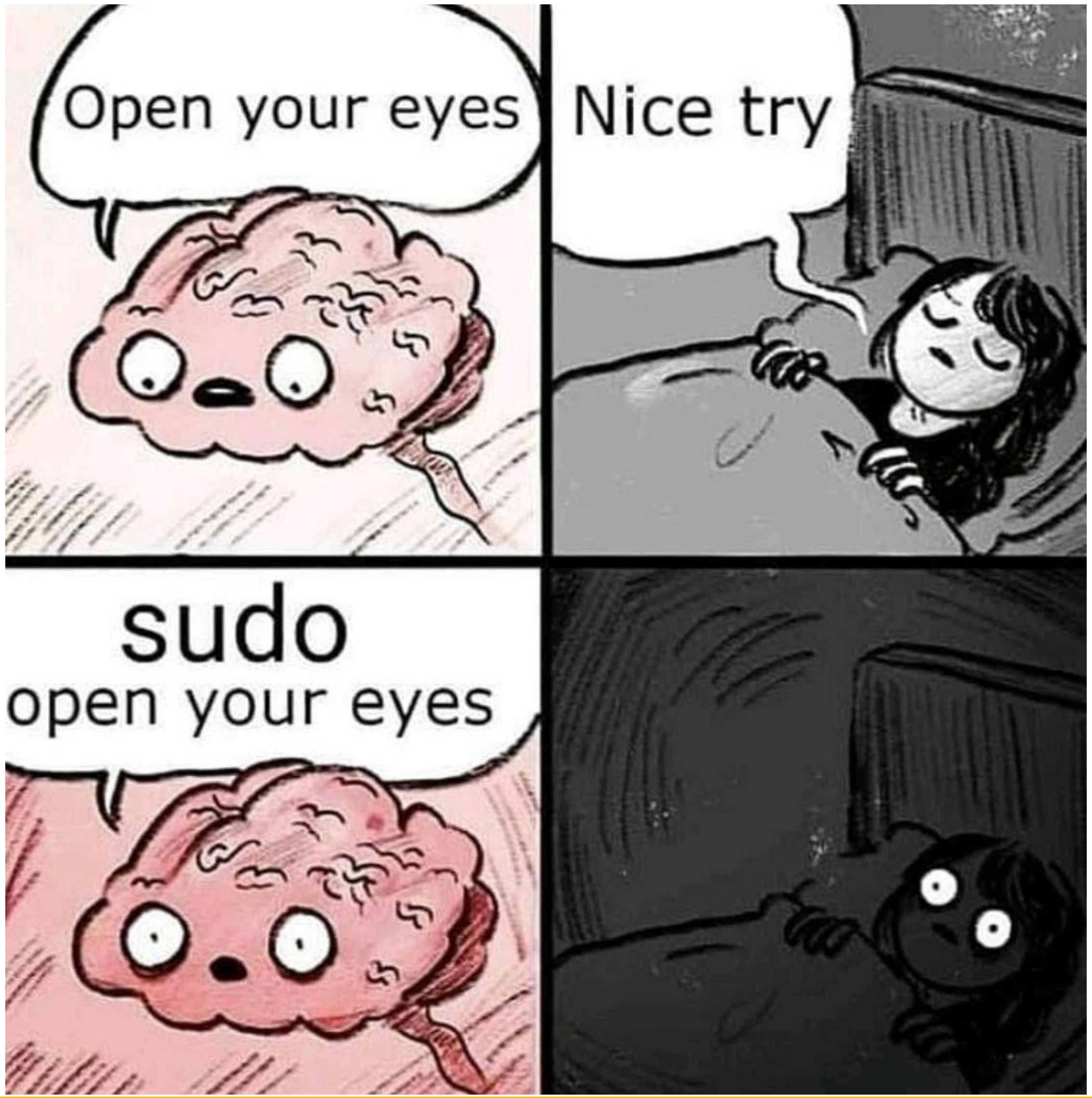
- **Switching the user with su**
  - ✓ log into the root user account: `$ su -`
  - ✓ enter another user's account: `$ su - mohsen`
  - ✓ Issuing a single command as another user: `$ su -c "passwd mohsen"`
- **Doing the Job as a Super User with sudo**
  - ✓ Viewing the `/etc/sudoers` configuration file: `$ sudo cat /etc/sudoers` or `$ sudo visudo`
    - Access for a particular user in the `/etc/sudoers` is designated using this format:

```
USERNAME HOSTNAME-OF-SYSTEM=(USER:GROUP) COMMANDS
```

    - Group records in the sudoers file are preceded with a percent sign (%).

```
$ sudo getent shadow anisa
$ journalctl -r -n 10 | grep sudo
```

Using sudo ☺



Repudiation

- Logging in as the root user can set up what is called a repudiation environment.
- A repudiation environment means that a person can deny actions.
- Therefore, if a system administrator uses the root account to perform some illegal or trouble making activity, the admin can legally deny being responsible for that activity.
- Systems where every user has an account and password and no one can log into the root user's account sets up a nonrepudiation environment.
- This means actions are logged and responsibility for them cannot be easily denied.
- A nonrepudiation environment can be created using sudo.

Auditing User Access

- `$ who`
  - ✓ shows all the current system users, the terminal they are using, the date and time they entered the system, and in cases of remote users, their remote IP address
- `$ who am i` and `$ who mom likes`
  - ✓ display information concerning only the current process's account
- `$ w`
  - ✓ The first displayed line shows the current time, how long the system has been up, how many users are currently accessing the system, the CPU load averages for the last 1, 5, and 15 minutes.
  - ✓ The next several lines concern current system user information. USER, TTY, LOGIN, IDLE, JCPU, PCPU, WHAT
  - ✓ The w utility pulls user information from the `/var/run/utmp` file.
  - ✓ It also gathers additional data for display from the `/proc/` directory files.
- `$ last`
  - ✓ The last command pulls information from the `/var/log/wtmp` file and displays a list of accounts showing the last time they logged in or out of the system or if they are still logged on.

Setting Login, Process, and Memory Limits

❖ ulimit - get and set user limits

```
ulimit [OPTIONS...]
```

✓ `-a`	List the limits for the current user account
✓ `-c`	Set the maximum core file size
✓ `-d`	Set the maximum data segment size for processes
✓ `-e`	Set the maximum allowed scheduling priority
✓ `-f`	Set the maximum file size allowed to be written
✓ `-i`	Set the maximum number of pending signals
✓ `-k`	Set the maximum number of kqueues that can be allocated
✓ `-l`	Set the maximum size of memory that can be locked

Setting Login, Process, and Memory Limits

✓ `-m`	Set the maximum resident set size
✓ `-n`	Set the maximum number of open file descriptors
✓ `-p`	Set the maximum pipe size in 512k blocks
✓ `-r`	Set the maximum real-time scheduling priority value
✓ `-s`	Set the maximum stack size
✓ `-t`	Set the maximum amount of CPU time the user account is allowed
✓ `-u`	Set the maximum number of processes the user can run simultaneously
✓ `-v`	Set the maximum amount of virtual memory available to the user
✓ `-x`	Set the maximum number of file locks

Locating SUID/SGID Files

```
# find / -perm /6000 -type f > SUID-SGID_Audit.txt
```

- After you have this report, review it to ensure that all the files listed should have those permissions.
- Then you can lock down the audit file's permissions and use the file as a baseline report for later audits.
- This would be a great shell script that you run periodically as a cron job, alerting you to any changes.

```
$ sudo find / -perm /6000 -type f > SUID-SGID_Audit_June-13.txt
$ diff SUID-SGID_Audit.txt SUID-SGID_Audit_June-13.txt
164a165
> /usr/bin/threat-actor
$ ls -l /usr/bin/threat-actor
-rwsrwsrwx 1 root root 0 Jun 13 17:25 /usr/bin/threat-actor
```

EXPLORING CRYPTOGRAPHY CONCEPTS

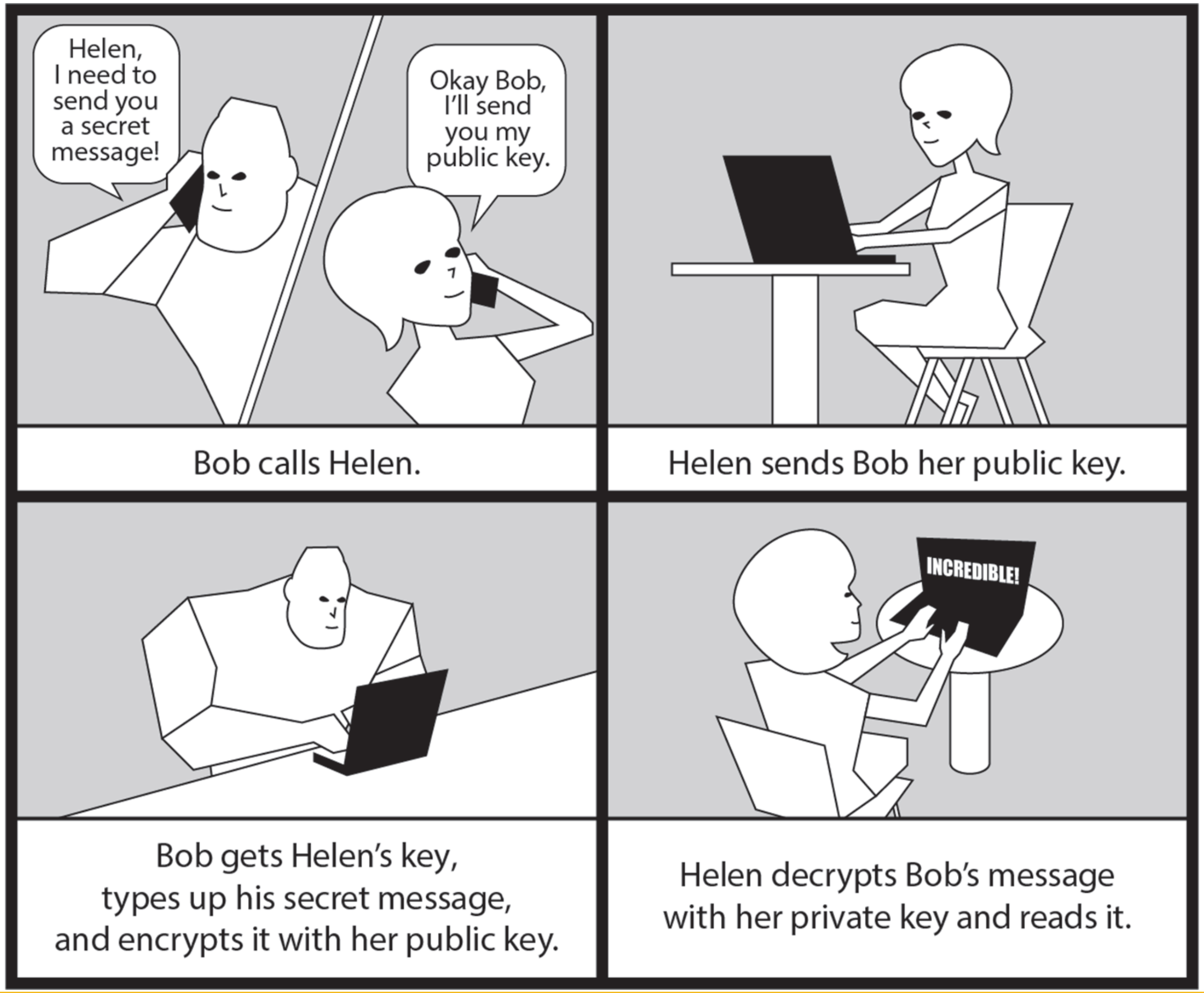
The primary purpose of cryptography is to encode data in order to hide it or keep it private. In cryptography, plaintext (text that can be read by humans or machines) is turned into ciphertext (text that cannot be read by humans or machines) via cryptographic algorithms. Turning plaintext into ciphertext is called encryption. Converting text from ciphertext back into plaintext is called decryption.

Cryptographic algorithms use special data called keys for encrypting and decrypting; they are also called cipher keys. When encrypted data is shared with others, some of these keys must also be shared.

Discovering Key Concepts

- **Cipher keys come in two flavors—private and public/private.**
- Private Keys
  - ✓ Symmetric keys, also called private or secret keys, encrypt data using a cryptographic algorithm and a single key.
  - ✓ Plaintext is both encrypted and decrypted using the same key, and it is typically protected by a password called a passphrase.
  - ✓ Symmetric key cryptography is very fast.
  - ✓ If you need others to decrypt the data, you have to share the private key, which is its primary disadvantage.
- Public/Private Key Pairs
  - ✓ Asymmetric keys, also called public/private key pairs, encrypt data using a cryptographic algorithm and two keys.
  - ✓ Typically the public key is used to encrypt the data and the private key decrypts the data.
  - ✓ The private key can be protected with a passphrase and is kept secret.
  - ✓ The public key of the pair is meant to be shared.

Discovering Key Concepts


Helen, I need to send you a secret message!
Okay Bob, I'll send you my public key.
Bob calls Helen.
Helen sends Bob her public key.
INCREDIBLE!
Bob gets Helen's key, types up his secret message, and encrypts it with her public key.
Helen decrypts Bob's message with her private key and reads it.

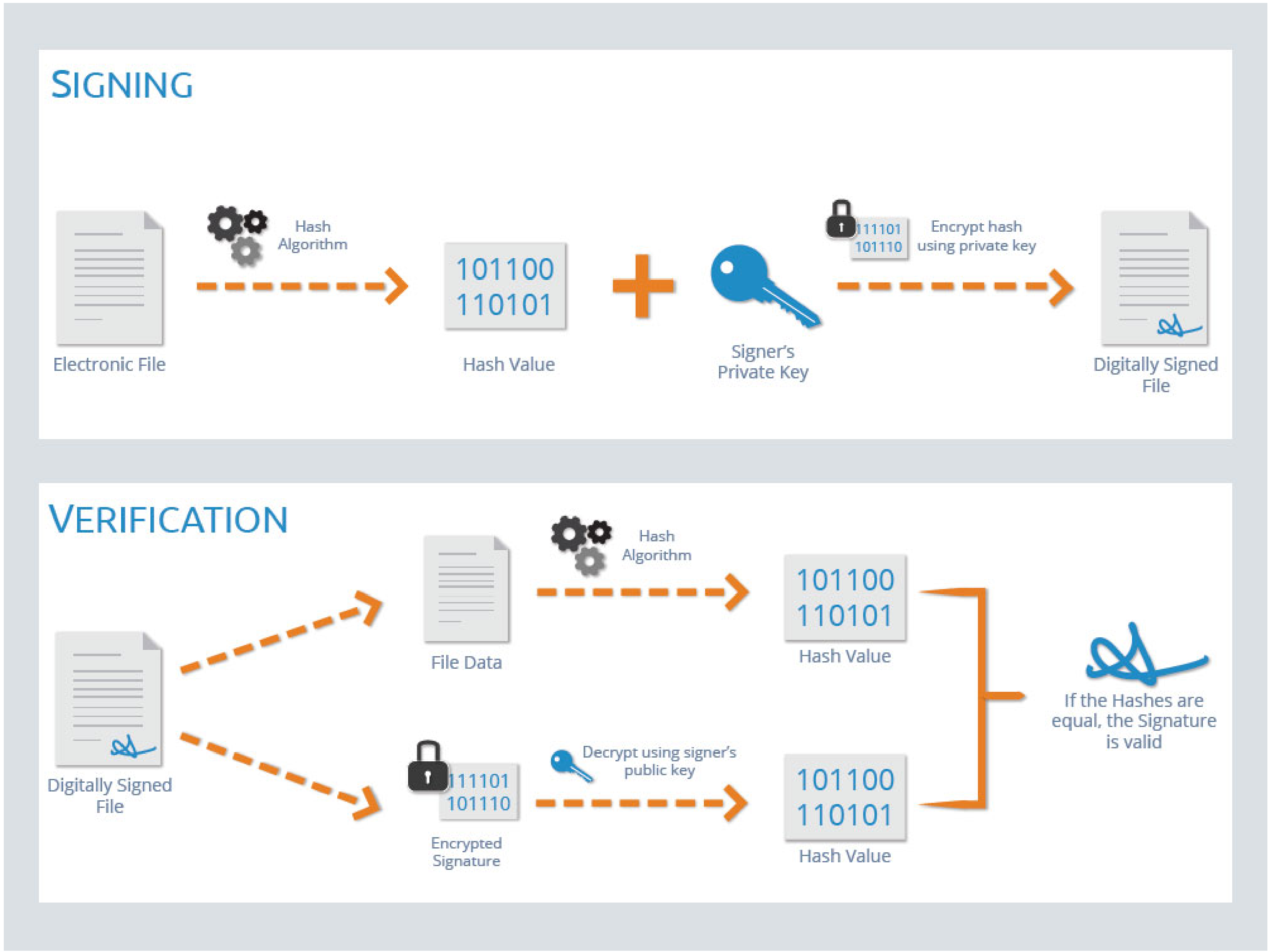
Securing Data

- An important concept in cryptography, is hashing.
- Hashing uses a one-way mathematical algorithm that turns plaintext into a fixed-length ciphertext.
  - ✓ Because it is one way, you cannot "de-hash" a hashed ciphertext.
  - ✓ The ciphertext created by hashing is called a message digest, hash, hash value, fingerprint, or signature.
- The beauty of a cryptographic message digest is that it can be used in data comparison.
  - ✓ For example, if hashing produces the same message digest for plaintext FileA and for plaintext FileB, then both files contain the same data.
  - ✓ This type of hash is often used in cyber forensics.
- A keyed message digest is created using the plaintext file along with a private key.

Signing Transmissions

- **Another practical implementation of hashing is in digital signatures.**
- **A digital signature is a cryptographic token that provides authentication and data verification.**
- **It is simply a message digest of the original plaintext data, which is then encrypted with a user's private key and sent along with the ciphertext.**
  - ✓ The ciphertext receiver decrypts the digital signature with the sender's public key so that the original message digest is available.
  - ✓ The receiver also decrypts the ciphertext and then hashes its plaintext data.
- **When the new message digest is created, the data receiver can compare the new message digest to the sent message digest.**
  - ✓ If they match, the digital signature is authenticated, which means the encrypted data did come from the sender.
  - ✓ Also, it indicates the data was not modified in transmission.

Signing Transmissions



LOOKING AT SSH

When you connect over a network to a remote server, if it is not via an encrypted method, network sniffers can view the data being sent and received. Secure Shell (SSH) has resolved this problem by providing an encrypted means for communication. It is the de facto standard software used by those wishing to send data securely to/from remote

systems.

SSH employs public/private key pairs (asymmetric) for its encryption. When an SSH connection is being established between two systems, each sends its public key to the other.

Exploring Basic SSH Concepts

- CentOS: `openssh, openssh-clients, openssh-server`
- Ubuntu: `openssh-server, openssh-client`
- ssh - OpenSSH SSH client (remote login program)

```
ssh [OPTIONS] USERNAME@HOSTNAME
```

```
$ ssh anisa@192.168.0.105
$ ssh anisa@192.168.0.104 "hostname"
```

✓ The OpenSSH application keeps track of any previously connected hosts in the `~/.ssh/known_hosts` file.

  - This data contains the remote servers' public keys.

- scp - secure copy (remote file copy program)

```
ssh [OPTIONS] SOURCE_FILE_LOCATION USERNAME@HOSTNAME:DESTINATION_FILE'S_LOCATION
```

- scp lets you copy files, but it employs SSH to copy the files to a remote system over an encrypted tunnel.

```
$ scp Project42.txt anisa@192.168.0.104:~
anisa@192.168.0.104's password:
Project42.txt 100% 0 0.0KB/s 00:00
```

Exploring Basic SSH Concepts

- Over time users tend to ignore the WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED message.
  - ✓ Thus, it's a good idea to help a user avoid experiencing this warning when first connecting to a system.
- To do that, you'll need to configure a `known_hosts` file for all the users on the system.
  - ✓ However, you don't have to do it for each individual user.
- Instead, the `ssh_known_hosts` file is used by all users on the system and can contain the remote server's public keys for all the systems they will connect to.
- You will have to manually create the file, and it typically resides in the `/etc/ssh/` directory.
- With the `/etc/ssh/ssh_know_hosts` file in place, users will receive the warning message only when something has changed and should be investigated.

Configuring SSH

- Primary OpenSSH configuration files
  - ✓ ~/.ssh/config
    - Contains OpenSSH client configurations. May be overridden by ssh command options.
    - For an individual user’s connections to a remote system.
  - ✓ /etc/ssh/ssh_config
    - Contains OpenSSH client configurations. May be overridden by ssh command options or settings in the ~/.ssh/config file.
    - For every user’s connection to a remote system
  - ✓ /etc/ssh/sshd_config
    - Contains the OpenSSH daemon (sshd) configurations.
    - For incoming SSH connection requests.

Configuring SSH

- There are several OpenSSH configuration directives.
  - ✓ You can peruse them all via the man pages for the `ssh_config` and `sshd_config` files.
- However, there are a few vital directives for the `sshd_config` file:
  - ✓ `AllowTcpForwarding`: Permits SSH port forwarding.
  - ✓ `ForwardX11`: Permits X11 forwarding.
  - ✓ `PermitRootLogin`: Permits the root user to log in through an SSH connection. (Default is yes.)
    - **Typically, should be set to no.**
  - ✓ `Port`: Sets the port number the OpenSSH daemon (sshd) listens on for incoming connection requests. (Default is 22.)
- Using ssh to connect to a nondefault port on a remote system:

1.
```
$ ssh -p 1138 192.168.0.104
```
2. Or create or modify the `~/.ssh/config` file for individual users, or for all client users, modify the `/etc/ssh/ssh_config` file. Set Port to **1138** within the configuration file.

Generating SSH Keys

- Typically, OpenSSH will search for its system's public/private key pairs.
  - ✓ If they are not found, OpenSSH automatically generates them.
  - ✓ These key pairs, also called host keys, are stored in the `/etc/ssh/` directory within files.

```
$ ls -1 /etc/ssh/*key*
/etc/ssh/ssh_host_ecdsa_key
/etc/ssh/ssh_host_ecdsa_key.pub
/etc/ssh/ssh_host_ed25519_key
/etc/ssh/ssh_host_ed25519_key.pub
/etc/ssh/ssh_host_rsa_key
/etc/ssh/ssh_host_rsa_key.pub
```

  - ✓ Private key files should have a 0640 or 0600 (octal) permission setting and be root owned.
  - ✓ The public key files end in the .pub filename extension, whereas the private keys have no filename extension.

Generating SSH Keys

- The filenames follow this standard:

```
ssh_host_KeyType_key
```

- The key filename's KeyType corresponds to the digital signature algorithm used in the key's creation.
  - ✓ **rsa** (Rivest–Shamir–Adleman) is the oldest type, widely used, and highly supported.
  - ✓ **dsa** (Digital Signature Algorithm) is a Federal Information Processing Standard for digital signatures. **Deprecated**.
  - ✓ **ecdsa** (Elliptical Curve Digital Signature Algorithm) is an Elliptic Curve implementation of DSA.
  - ✓ **ed25519** is a variation of the Edwards-curve Digital Signature Algorithm (EdDSA) that offers better security than dsa and ecdsa.

Generating SSH Keys

- There may be times you need to manually generate these keys or create new ones.
- ssh-keygen — authentication key generation, management and conversion
  - ✓ **-t** sets the KeyType.
  - ✓ **-f** designates the private key file to store the key.
  - ✓ The public key is stored in a file with the same name, but the .pub file extension is added.
  - ✓ Notice that this command asks for a passphrase, which is associated with the private key.

```
$ sudo ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key
Generating public/private rsa key pair.
[…]
```

Authenticating with SSH Keys

- Entering the password for every command employing SSH can be tiresome.
- However, you can use keys instead of a password to authenticate.
  1. Log into the SSH client system.
  2. Generate an SSH ID key pair.
  3. Securely transfer the public SSH ID key to the SSH server computer.
  4. Log into the SSH server system.
  5. Add the public SSH ID key to the ~/.ssh/authorized_keys file on the server system.

```
$ ssh-keygen -t rsa -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
[…]
$ ls .ssh/
id_rsa id_rsa.pub known_hosts
```

Authenticating with SSH Keys

- After these keys are generated on the client system, the public key must be copied to the server system.
- Using a secure method is best, and the ssh-copy-id utility allows you to do this.
- Not only does it copy over your public key, it also stores it in the server system's ~/.ssh/authorized_keys file for you.

```
$ ssh-copy-id -n anisa@192.168.0.104
[…]
Would have added the following key(s):
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCsP[…]
8WJVE5RWAXN[…]
=-=-=-=-=-=-=-=
$ ssh-copy-id anisa@192.168.0.104
[…]Source of key(s) to be installed: "/home/anisa/.ssh/id_rsa.pub"
```

- The –n option allows you to see what keys would be copied and installed on the remote system without actually doing the work (a dry run).
- Thus, the id_rsa.pub key file is securely copied to the server system, and the key is installed in the ~/.ssh/authorized_keys file.

Authenticating with SSH Keys

- Notice that when using the ssh-copy-id command, the user must enter their password to allow the public ID key to be copied over to the server.
- Now that the public ID key has been copied over to the SSH server system, the ssh command can be used to connect from the client system to the server system with no need to enter a password.

```
$ ssh anisa@192.168.0.104
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-36-generic x86_64)
[…]
anisa@Ubuntu1804:~$ ls .ssh
authorized_keys known_hosts
$ scp Project4x.tar anisa@192.168.0.104:~
Project4x.tar 100% 40KB 6.3MB/s 00:00
$ ssh anisa@192.168.0.104
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-36-generic x86_64)
[…]
anisa@Ubuntu1804:~$ ls
Desktop Downloads Music Project4x.tar Templates Documents examples.desktop Pictures Public
Videos
```

Authenticating with the Authentication Agent

- **Another method to connect to a remote system with SSH is via the authentication agent.**
  - ✓ Using the agent, you only need to enter your password to initiate the connection.
  - ✓ After that, the agent remembers the password during the agent session.
- **A few steps are needed to set up this authentication method:**
  1. Log into the SSH client system.
  2. Generate an SSH ID key pair and set up a passphrase.
  3. Securely transfer the public SSH ID key to the SSH server computer.
  4. Log into the SSH server system.
  5. Add the public SSH ID key to the ~/.ssh/authorized_keys file on the server system.
  6. Start an agent session.
  7. Add the SSH ID key to the agent session.

```
$ ssh-keygen -t ecdsa -f ~/.ssh/id_ecdsa
Generating public/private ecdsa key pair.
[…]
$ ssh-copy-id -i ~/.ssh/id_ecdsa anisa@192.168.0.104
Number of key(s) added: 1
[…]
```

Authenticating with the Authentication Agent

- When you have the key pair properly created with a passphrase on the remote system, securely transmitted, and installed on the server's authorized key file, you can employ the ssh-agent utility to start an SSH agent session.
- After the session is started, add the private ID key to the session via the ssh-add command.

```
$ ssh-agent /bin/bash
[anisa@localhost ~]$ ssh-add ~/.ssh/id_ecdsa
Enter passphrase for /home/anisa/.ssh/id_ecdsa:
Identity added: /home/anisa/.ssh/id_ecdsa (/home/anisa/.ssh/id_ecdsa)
[anisa@localhost ~]$ ssh anisa@192.168.0.104
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-36-generic x86_64)
[…]
```

- After the private SSH ID key is added using the ssh-add command and entering the private passphrase, you can connect to remote systems without entering a password or passphrase again.
- However, if you exit the agent session and start it up again, you must readd the key and reenter the passphrase.
- An SSH agent session allows you to enter the session one time and add the key, and then connect as often as needed to remote systems via encrypted SSH methods without entering a password or passphrase over and over again.
- Not only does this provide security, it also provides convenience, which is a rare combination.

Tunneling

- **Another way to provide security through OpenSSH is via SSH port forwarding, sometimes called SSH tunneling.**
  - ✓ SSH port forwarding allows you to redirect a connection from one particular network port to port 22, where the SSH service is waiting to receive it.
- **This allows data traffic to move back and forth through a secure encrypted tunnel, similar to a virtual private network.**
- **If you need to provide remote X11 GUI interactions, you can employ OpenSSH to use a secure tunnel.**
  - ✓ This is called X11 forwarding.
  - ✓ X11 forwarding lets you interact with various X11-based graphical utilities on a remote system through an encrypted network connection.
- **First check to see if X11 forwarding is permitted in the openSSH configuration file, `/etc/ssh/sshd_config`.**

```
# grep "X11Forwarding yes" /etc/ssh/sshd_config
X11Forwarding yes
```

- **The command to use is `ssh -X user@remote-host`**
  - ✓ the user is the user account that resides on the remote-host system.
  - ✓ The remote-host has the X11-based GUI utilities you wish to employ and can be designated via an IP address or a hostname.

Using SSH Securely

- There are a few things you can do to enhance SSH's security on your systems:
  - ✓ **Use a different port for SSH than the default port 22.**
    - Keep in mind that there are advantages and disadvantages to doing this. It may be a better alternative to beef up your firewall as opposed to changing the default SSH port.
  - ✓ **Disable root logins via SSH.**
    - By default, any system that allows the root account to log in and has OpenSSH enabled permits root logins via SSH.
    - Because root is a standard username, malicious attackers can use it in brute-force attacks.
    - Since root is a super user account, it needs extra protection.
  - ✓ **Ensure protocol 2 is in use.**
    - An earlier version of OpenSSH, protocol 1, is considered insecure.
    - Most likely your system is employing protocol 2, sometimes called OpenSSH 2.
    - Find the Protocol directive in the `/etc/ssh/sshd_config` file.
    - If it is set to 1, then change it to 2, and restart the OpenSSH service or reload its configuration file.

USING GPG

In cases where you need to employ file encryption and transfer the file beyond what scp can handle, GNU Privacy Guard (GPG or GnuPG) can help. Besides encrypting files, you can apply digital signatures to them, providing higher security.

Because GPG was based on the Pretty Good Privacy product, which is not open source, you'll often see it referred to as OpenPGP as well.

Generating Keys

- To begin using GPG, you'll need to generate a public/private key pair.
  - ✓ GPG uses the typical asynchronous keys, where the public key is available publicly but the private key is kept private.
    - GPG calls the private key a secret key.

```
$ gpg --gen-key
```

    - Optionally you can use before generating key => `# rngd -r /dev/urandom`

Generating Keys

- After the keys are generated, they are stored in a file, which is called your keyring in the `~/.gnupg/` directory.
- For someone to encrypt a file for you to decrypt, you'll need to make a copy of your public key for them.
- This is called exporting your key, and it will put the copy of the key into a file.

```
gpg --export EMAIL-ADDRESS > FILENAME.pub
```

  - ✓ The `EMAIL-ADDRESS` identifies your public key on your keyring, and it was entered when you generate the key.
  - ✓ The `FILENAME` can be anything.

```
$ gpg --export mohsen.m.amini@anisa.co.ir > mohsen_gpg.pub
```

- After you export (copy) your public key to a file, you can give it to the people who want to encrypt files to send to you.
  - ✓ There is no reason to keep the public key secret, so you can email it or, if you have a public web page, place it there.

Importing Keys

- When you receive the public key file for encrypting files to someone else, you'll need to add it to your keyring before employing it.
- This process is called importing, and it uses the

```
$ gpg --import FILENAME.pub
```

  - ✓ After you have their public key added to your keyring, you can begin to encrypt files for them.
- After you load a new key onto your keyring, it's a good idea to check the various keys residing there.

```
$ gpg --list-keys
```

```
$ gpg --import mohsen_gpg.pub
$ gpg --list-keys
```

Encrypting and Decrypting Data

- First create the file through a normal method, such as using a word processor or text editor.
- Then encrypt the file using their public key:
  - ✓ `gpg --out ENCRYPTED-FILE --recipient EMAIL-ADDRESS --encrypt ORIGINAL-FILE`
  - ✓ Note that the `ORIGINAL-FILE` is the unencrypted file you first created and the `ENCRYPTEDFILE` is the ciphertext file.
  - ✓ The `EMAIL-ADDRESS` is the email address of the person who will be receiving this encrypted file, and it should be the address used to identify that person's public key on your keyring.

```
$ whoami
anisa
$ gpg --out encryptfile --recipient mohsen.m.amini@anisa.co.ir --encrypt secretfile.txt
```

Encrypting and Decrypting Data

❖ The person who receives your encrypted file can then decrypt it using their private (secret) key.

```
$ gpg --out DECRYPTED-FILE --decrypt ENCRYPTED-FILE
```

✓ When you issue the preceding command, it will ask for the passphrase that identifies your private key.

```
$ whoami
Mohsen
$ gpg --out CBmessage.txt --decrypt encryptfile
gpg: encrypted with 3072-bit RSA key, ID E3E4D21F82775FA1, created 2019-06-14
"Mohsen Mohammad Amini < mohsen.m.amini@anisa.co.ir>"
$ cat CBmessage.txt
Mohsen,
This is last session.
Will we meet eachother in future?
```

Signing Messages and Verifying Signatures

- While encrypting helps to protect the privacy of a document, it does not protect the document from being modified in transit.
- If a malicious actor (Evelyn) gets a hold of Helen’s public key, the threat actor could encrypt a message for Helen and claim it came from Bob.
  - ✓ Or worse, if Bob is sending Helen source code, a threat actor could modify it in some way before it reaches Helen.
- You can digitally sign gpg encrypted files.
  - ✓ This process creates a time stamp and certifies the file.
- If the file is modified in any way, the gpg utility will alert the file’s receiver when checked.

Signing Messages and Verifying Signatures

```
$ cat newsecret.txt
Hi Anisa,
This file came from me.
Signed,
Mohsen Mohammad Amini
$ gpg --out tosign --recipient mohsen.m.amini@anisa.co.ir --encrypt
newsecret.txt
$ gpg --output signed --sign tosign
```

- When the recipient receives the encrypted and signed file, they can verify it came from the sender and that no modification occurred in the transfer:

```
$ gpg --verify RECEIVED-SIGNED-FILE
```

Signing Messages and Verifying Signatures

- Checking the signature and decrypting the original file takes two steps.
  - ✓ First you must decrypt and verify the signature.

```
$ gpg --out MessageFromCB.gpg --decrypt signed
```

  - ✓ After the signature is decrypted and verified, the original file is decrypted using the methods described earlier in this section.

```
$ gpg --out MessageFromCB.txt --decrypt MessageFromCB.gpg
$ cat MessageFromCB.txt
Hi Anisa,
This file came from me.
Signed,
Mohsen Mohammad Amini
```

Revoking a Key

❖ **If your private key has been compromised or stolen, you need to revoke your public key.**

1. Generate a revocation certificate.
2. Import the revocation certificate into your keyring.
3. Make available the revocation certificate to those who have your public key.

```
$ whoami
Mohsen
$ gpg --out key-revocation.asc --gen-revoke mohsen.m.amini@anisa.co.ir
[…]
Create a revocation certificate for this key? (y/N) y
[…]
```

✓ After you have the revocation certification, you can import it into your keyring to officially void your public/private key pair:

```
$ gpg --import FILENAME.asc
```