

# CS 35L

LAB 3,

TA: Sucharitha Prabhakar

EMAIL ID: [prabhakarsucharitha@gmail.com](mailto:prabhakarsucharitha@gmail.com)

# Outline

**Security basics**


**Intrusion detection**

**Security policies**



# Communication Over the Internet

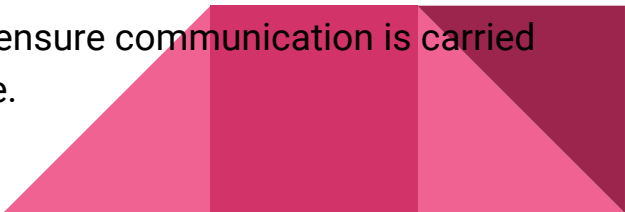
What type of guarantees do we want?

- Confidentiality
    - Message secrecy
  - Data integrity
    - Message consistency
  - Authentication
    - Identity confirmation
  - Authorization
    - Specifying access rights to resources
  - Availability
    - Information must be available to authorized individuals when they need it.
- 

# Telnet

A network protocol that allows a user on one computer to log onto another computer that is part of the same network.

Provides a bidirectional interactive text-oriented communication facility using a virtual terminal connection

- Telnet, by default, does not encrypt any data sent over the connection (including passwords), and so it is often feasible to eavesdrop on the communications and use the password later for malicious purposes; anybody who has access to a device located on the network between the two hosts where Telnet is being used can intercept the packets passing by and obtain login, password and whatever else is typed with a packet analyzer
  - Most implementations of Telnet have no authentication that would ensure communication is carried out between the two desired hosts and not intercepted in the middle.
- 

# What is SSH?

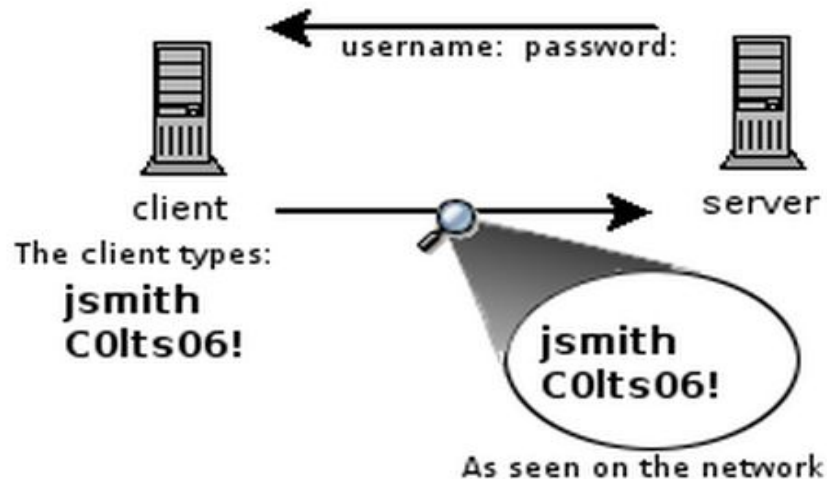
## Secure Shell

- Used to remotely access
- Successor of telnet
- Encrypted and better authenticated session

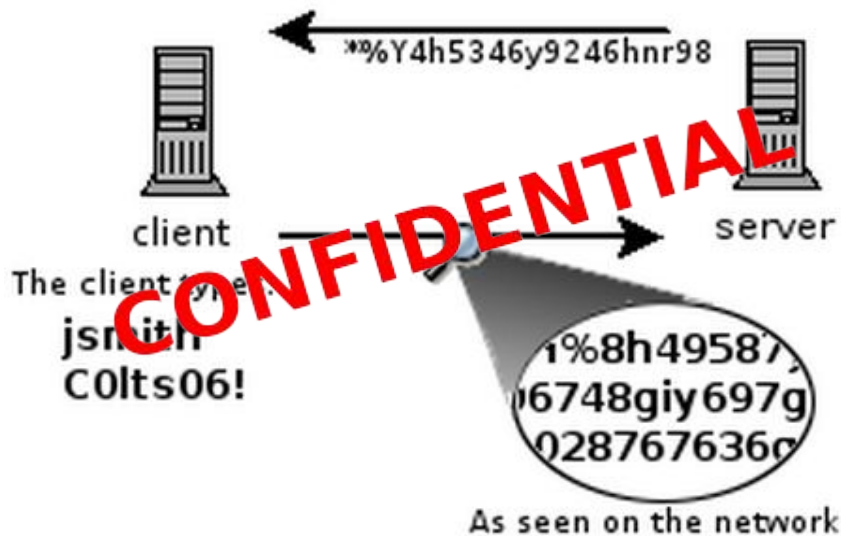


# What is SSH?

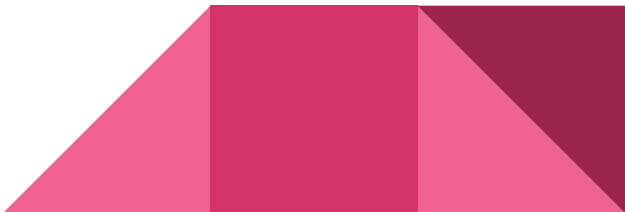
An unencrypted login session  
such as through telnet



An encrypted login session  
such as through SSH



# Encryption Types

- Symmetric Key Encryption
    - a.k.a shared/secret key
    - Key used to encrypt is the same as key used to decrypt
  - Asymmetric Key Encryption: Public/Private
    - 2 different (but related) keys: public and private
    - Only creator knows the relation. Private key cannot be derived from public key
  - Data encrypted with public key can only be decrypted by private key and vice versa
  - Public key can be seen by anyone
  - Never publish private key!!!
- 

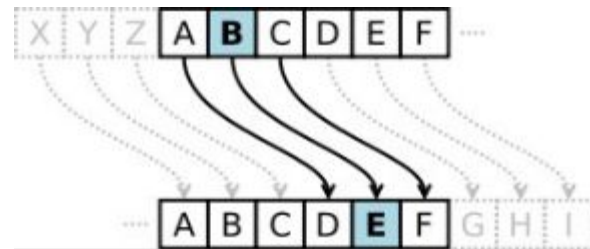
# Symmetric Key Encryption

Same secret key used for encryption and decryption

Example : Data Encryption Standard (DES)

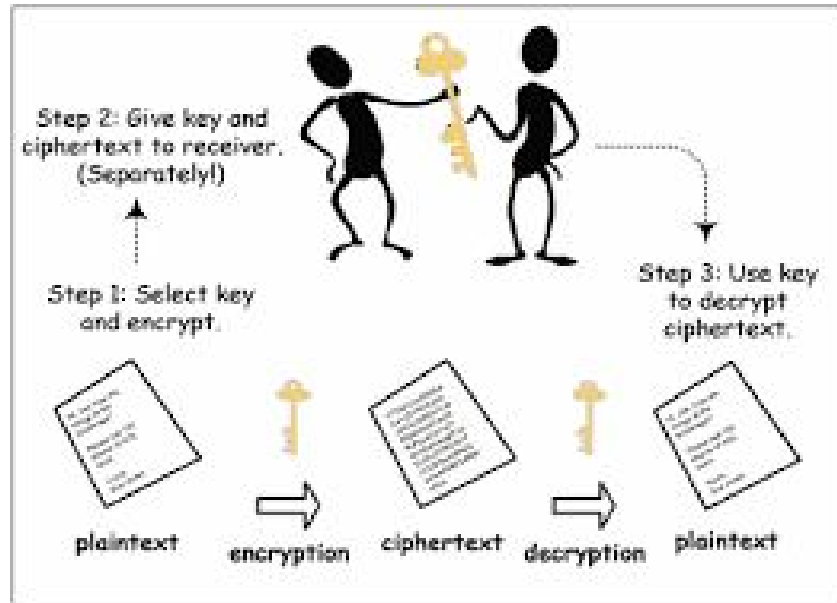
Caesar's cipher

- Map the alphabet to a shifted version
  - Plaintext – SECRET.
  - Ciphertext – VHFUHW
  - Key is 3 (number of shifts of the alphabet)





# Symmetric Key Encryption

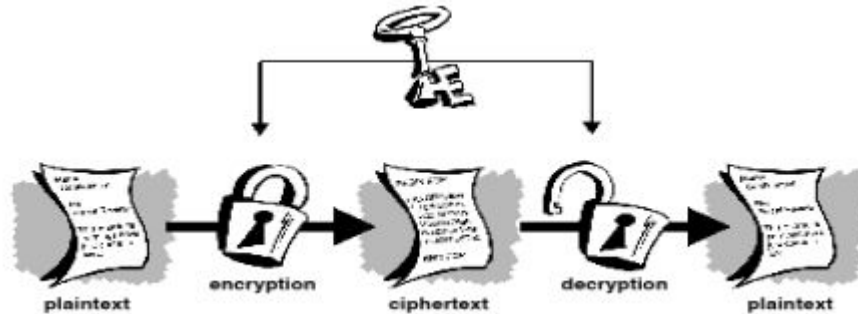


# Problem with Symmetric Key Encryption

Key distribution is a problem

The secret key has to be delivered in a safe way to the recipient

Chance of key being compromised



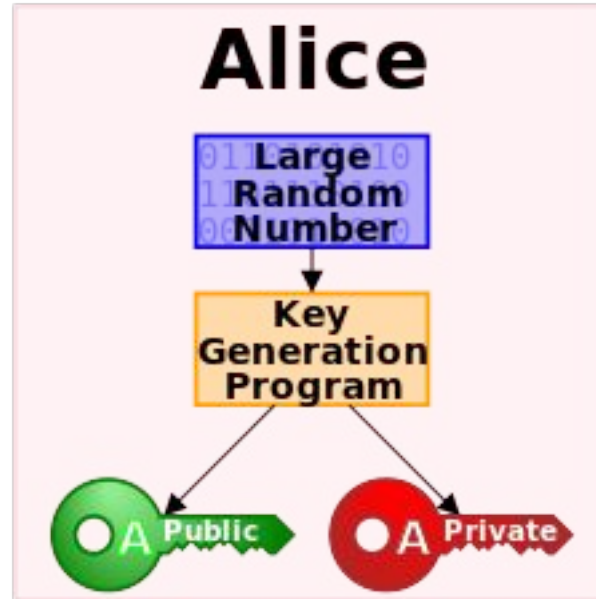
# Public-key Encryption (Asymmetric)

Uses a pair of keys for encryption

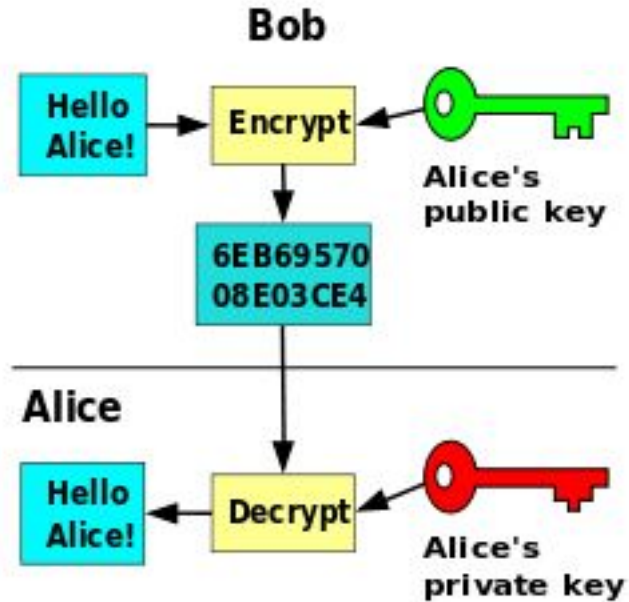
- Public key – Published and known to everyone
- Private key – Secret key known only to the owner
  - Encryption
    - Use public key to encrypt messages
    - Anyone can encrypt message, but they cannot decrypt the ciphertext
  - Decryption
    - Use private key to decrypt messages
  - Example: RSA



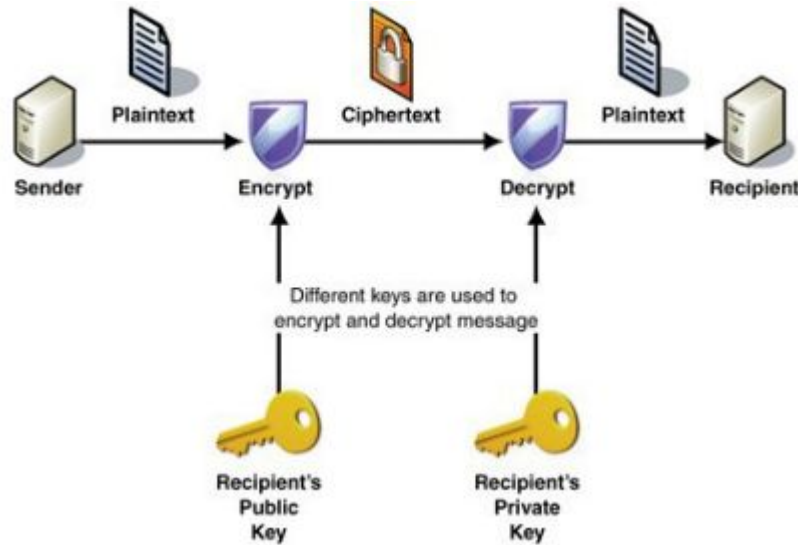
# Key generation



# Encryption



# Public-key Encryption (Asymmetric)



# SSH Protocol

Client ssh's to remote server

```
$ ssh username@somehost
```

If first time talking to server -> host validation

The authenticity of host 'somehost (192.168.1.1)' can't be established.

RSA key fingerprint is 90:9c:46:ab:03:1d:30:2c:5c:87:c5:c7:d9:13:5d:75.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'somehost' (RSA) to the list of known hosts.



# SSH Protocol

ssh doesn't know about this host yet

shows hostname, IP address and fingerprint of the server's public key, so you can be sure you're talking to the correct computer

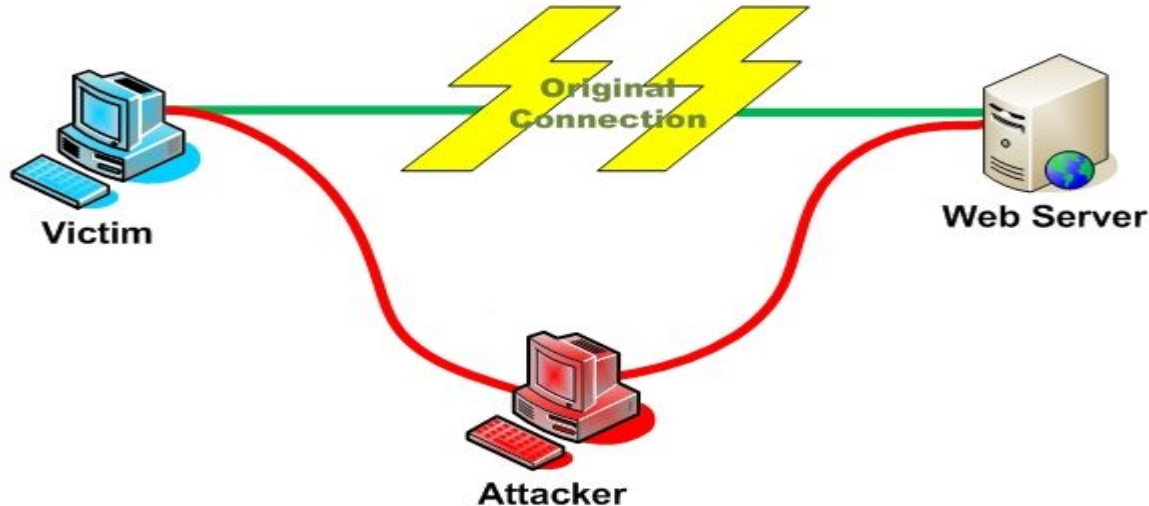
After accepting, public key is saved in `~/.ssh/known_hosts`





# Host Validation

- Next time client connects to server
- Check host's public key against saved public key
- If they don't match, warning



# Host Validation

Client asks server to prove that it is the owner of the public key using asymmetric encryption

Encrypt a message with public key

If server is true owner, it can decrypt the message with private key

If everything works, host is successfully validated



# Session Encryption

Client and server agree on a symmetric encryption key (session key)

All messages sent between client and server

- encrypted at the sender with session key
- decrypted at the receiver with session key

Anybody who doesn't know the session key (hopefully, no one but client and server) doesn't know any of the contents of those messages



# Client Authentication

- Password-based authentication
  - Prompt for password on remote server
  - If username specified exists and remote password for it is correct then the system lets you in
- Key-based authentication
  - Generate a key pair on the client
  - Copy the public key to the server (`~/.ssh/authorized_keys`)
  - Server authenticates client if it can demonstrate that it has the private key
  - The private key can be protected with a passphrase
  - Every time you ssh to a host, you will be asked for the passphrase (inconvenient!)



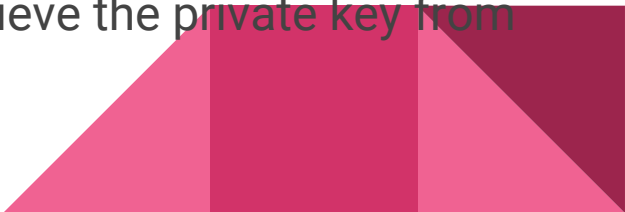
# ssh-agent (passphrase-less ssh)

A program used with OpenSSH that provides a secure way of storing the private key

ssh-add prompts user for the passphrase once and adds it to the list maintained by ssh-agent

Once passphrase is added to ssh-agent, the user will not be prompted for it again when using SSH

OpenSSH will talk to the local ssh-agent daemon and retrieve the private key from it automatically



# X Window System

Windowing system that forms the basis for most GUIs on UNIX

X is an architecture-independent system for remote graphical user interfaces and input device capabilities

X is a network-based system. It is based upon a network protocol such that a program can run on one computer but be displayed on another (X Session Forwarding)



# Lab

Securely log in to each others' computers

- Use ssh (OpenSSH)

Use key-based authentication

- Generate key pairs

Make logins convenient

- Type your passphrase once and be able to use ssh to connect to any other host without typing any passwords or passphrases

Use port forwarding to run a command on a remote host that displays on your host

# Setup

## Ubuntu

Make sure you have openssh-server and openssh-client installed

\$ dpkg --get-selections | grep openssh should output:

- openssh-server install
- openssh-client install

If not:

\$ sudo apt-get install openssh-server

\$ sudo apt-get install openssh-client





# Server Steps

Generate public and private keys

\$ ssh-keygen (by default saved to ~/.ssh/id\_rsa and id\_rsa.pub) – don't change the default location

Create an account for the client on the server

\$ sudo useradd -d /home/<homedir\_name> -m <username>

\$ sudo passwd <username>

Create .ssh directory for new user

\$ cd /home/<homedir\_name>



# Server steps

```
$ sudo mkdir .ssh
```

Change ownership and permission on .ssh directory

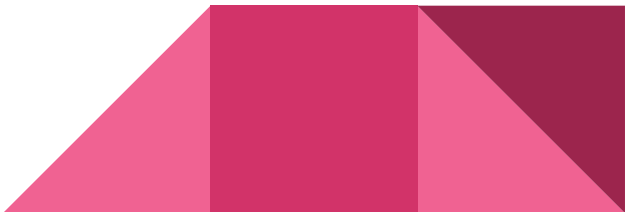
```
$ sudo chown -R username .ssh
```

```
$ sudo chmod 700 .ssh
```

Optional: disable password-based authentication

```
$ emacs /etc/ssh/sshd_config
```

change PasswordAuthentication option to no



# Client steps

Generate public and private keys

```
$ ssh-keygen
```

Copy your public key to the server for key-based authentication  
(`~/.ssh/authorized_keys`)

```
$ ssh-copy-id -i UserName@server_ip_addr
```

Add private key to authentication agent (`ssh-agent`)

```
$ ssh-add
```



# Client steps

SSH to server

```
$ ssh UserName@server_ip_addr
```

```
$ ssh -X UserName@server_ip_addr (X11 session forwarding)
```

Run a command on the remote host

```
$ xterm, $ gedit, $ firefox, etc.
```



# How to check IP Address

- `$ ifconfig`
  - configure or display the current network interface configuration information (IP address, etc.)
- `$ ping <ip_addr>(packet internet groper)`
  - Test the reachability of a host on an IP network
  - measure round-trip time for messages sent from a source to a destination computer
  - Example: `$ ping 192.168.0.1`, `$ ping google.com`



# Assignment 10 report format

1. Introduction
2. Background
3. Summary
4. Results
5. Your opinion



# Finals

- 100 points
- 180 minutes
- Open book + open notes
- Theory questions
- Programming questions (No coding on lab machines)
- No electronic gadgets allowed



# Digital Signature

An electronic stamp or seal - almost exactly like a written signature, except more guarantees!

Is appended to a document

- Or sent separately (detached signature)

Ensures data integrity

- Document was not changed during transmission





# Steps for generating a digital signature

SENDER:

## 1. Generate a Message Digest

- a. The message digest is generated using a set of hashing algorithms
- b. A message digest is a 'summary' of the message we are going to transmit
- c. Even the slightest change in the message produces a different digest

## 2. Create a Digital Signature

- a. The message digest is encrypted using the sender's private key. The resulting encrypted message digest is the digital signature

## 3. Attach digital signature to message and send to receiver



# Steps for generating a digital signature

RECEIVER:

1. Recover the Message Digest

- a. Decrypt the digital signature using the sender's public key to obtain the message digest generated by the sender

2. Generate the Message Digest

- a. Use the same message digest algorithm used by the sender to generate a message digest of the received message



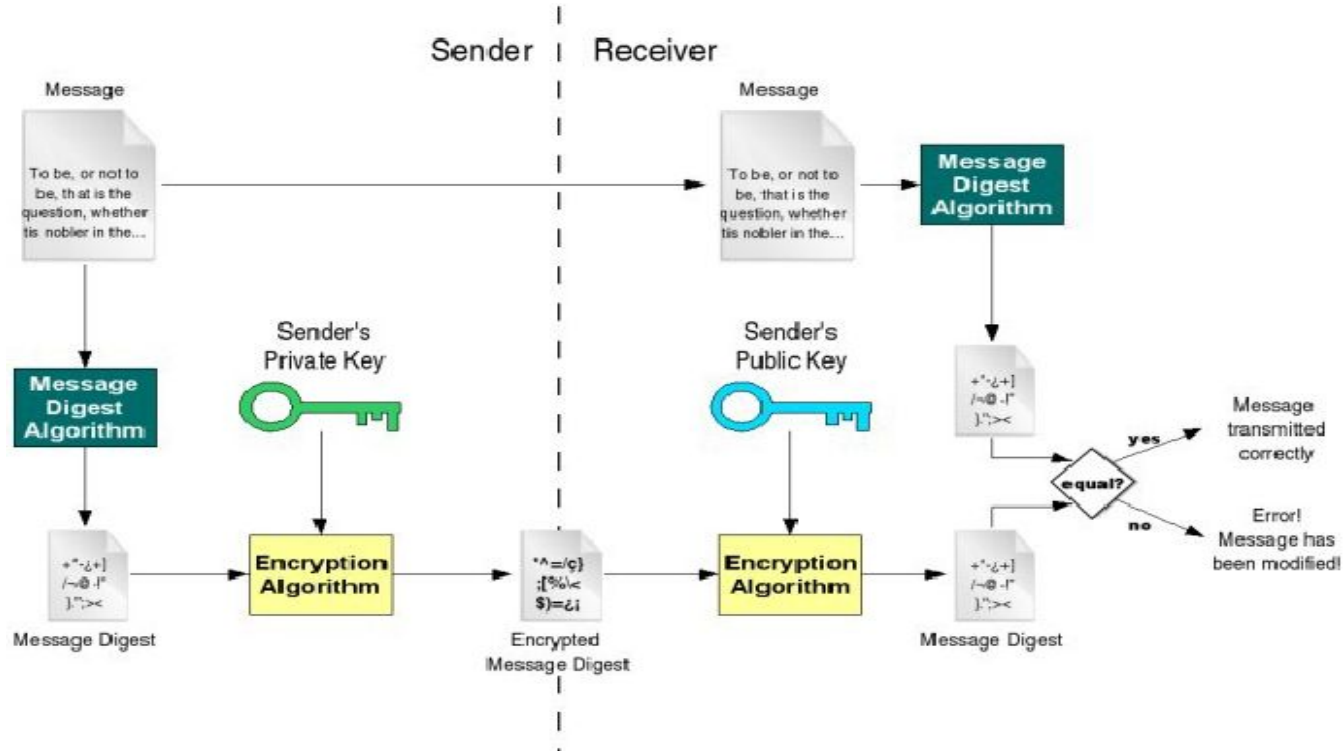
# Steps for generating a digital signature

## RECEIVER:

1. Compare digests (the one sent by the sender as a digital signature, and the one generated by the receiver)
  - a. If they are not exactly the same => the message has been tampered with by a third party
  - b. We can be sure that the digital signature was sent by the sender (and not by a malicious user) because only the sender's public key can decrypt the digital signature and that public key is proven to be the sender's through the certificate. If decrypting using the public key renders a faulty message digest, this means that either the message or the message digest are not exactly what the sender sent.



# Digital Signature



# Detached Signature

Digital signatures can either be attached to the message or detached

A detached signature is stored and transmitted separately from the message it signs

Commonly used to validate software distributed in compressed tar files

You can't sign such a file internally without altering its contents, so the signature is created in a separate file



# Homework

Answer 2 questions in the file hw.txt

1. Generate a key pair with the GNU Privacy Guard's commands
  - a. `$ gpg --gen-key` (choose default options)
2. Export public key, in ASCII format, into hw-pubkey.asc
  - a. `$ gpg --armor --output hw-pubkey.asc --export 'Your Name'`



# Homework

- Make a tarball of the above files + log.txt and zip it with gzip to produce hw.tar.gz
  - `$ tar -cf hw.tar <files>`
  - `$ gzip hw.tar -> creates hw.tar.gz`
- Use the private key you created to make a detached clear signature hw.tar.gz.sig for hw.tar.gz
  - `$ gpg --armor --output hw.tar.gz.sig --detach-sign hw.tar.gz`
- Use given commands to verify signature and file formatting
- These can be found at the end of the assignment spec

