Communication Over the Internet

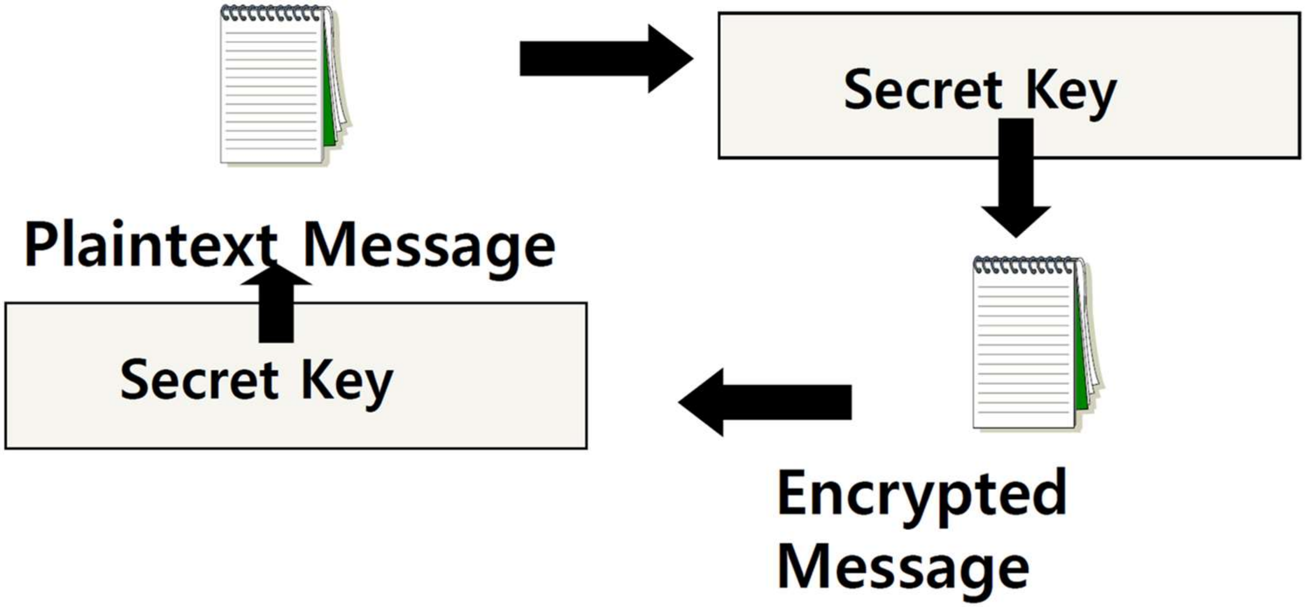
- What type of guarantees do we want?
  - **Confidentiality**
    - Message secrecy
  - **(Data) Integrity**
    - Message consistency
  - **Authentication**
    - Identity confirmation
  - **Also authorization**
    - Specifying access rights to resources

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!!!

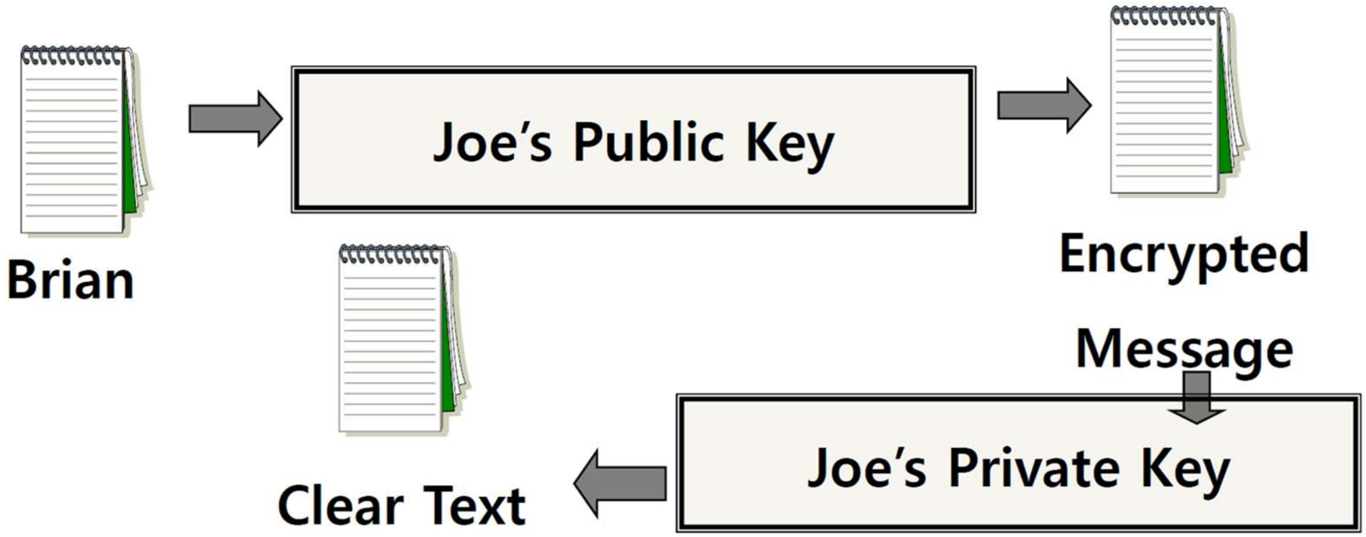
Secret Key (symmetric) Cryptography

- A single key is used to both encrypt and decrypt a message



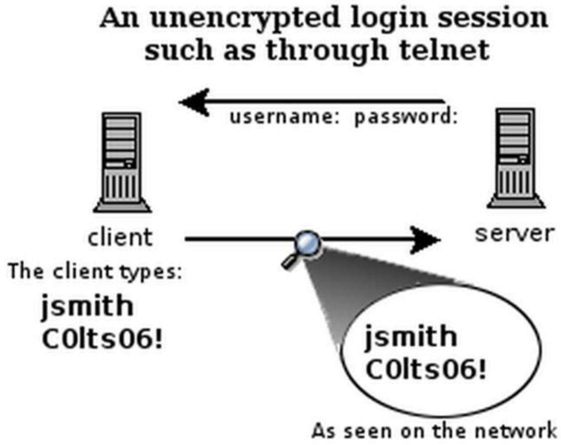
Public Key (asymmetric) Cryptography

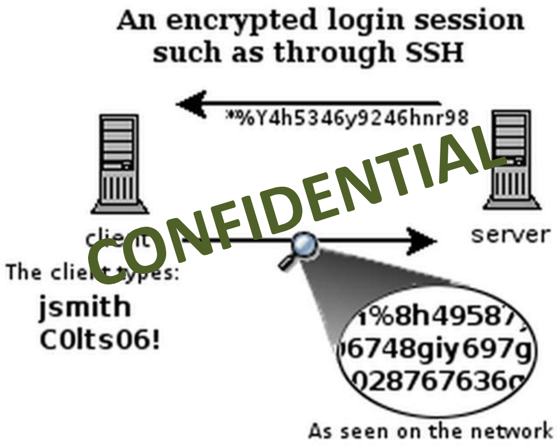
- Two keys are used: a public and a private key. If a message is encrypted with one key, it has to be decrypted with the other.



What is SSH?

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





High-Level SSH Protocol

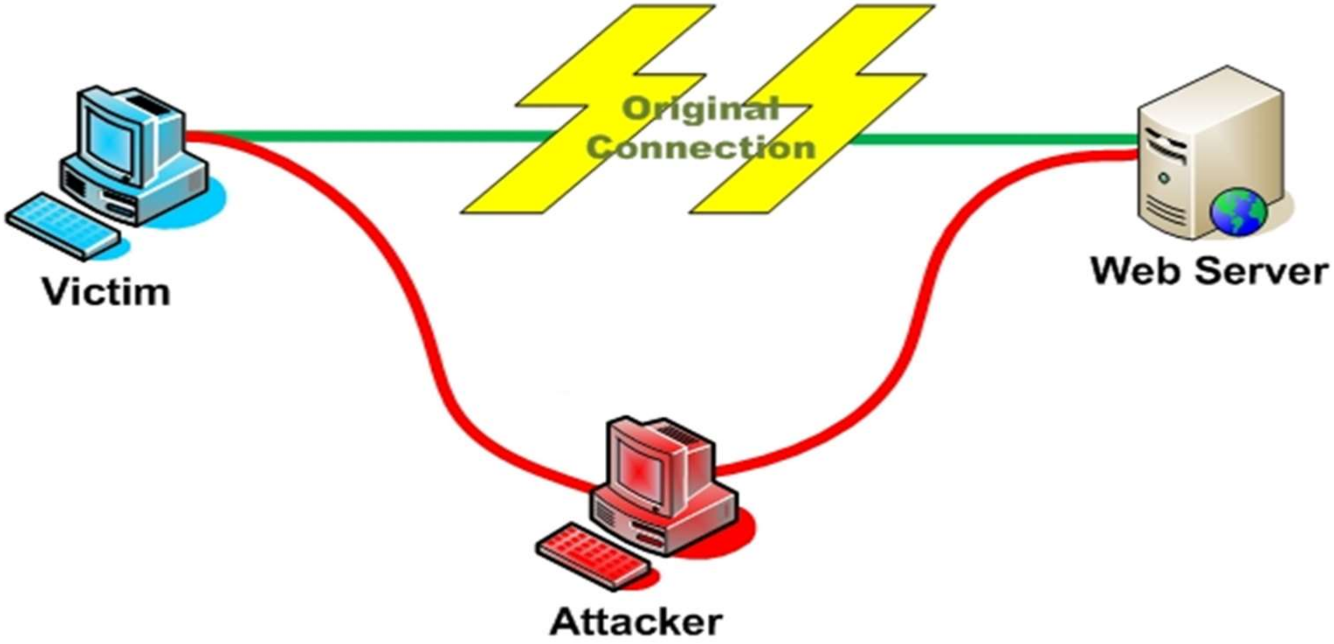
- Client ssh’s to remote server
  - `$ ssh username@somehost`
  - If first time talking to server -> host validation

```
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 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



Host Validation (cont'd)

- 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

AUTHENTICATION

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

User 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

Server Steps

- **Generate public and private keys**
  - $ `ssh-keygen` (by default saved to ~/.ssh/is_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>
    ```

  -
    ```
    $ sudo mkdir .ssh
    ```
- **Change ownership and permission on .ssh directory**

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

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

Client Steps – Make logins convenient

- **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`
- **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 Addresses

- $ `ifconfig`
  - configure or display the current network interface configuration information (IP address, etc.)
- $ `hostname -I`
  - gives the IP address of your machine directly
- $ `ping <ip_addr>`(**p**acket **in**ternet **g**roper)
  - 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

Steps for Generating a Digital Signature

Ensures data integrity (document was not changed during transmission)

SENDER:

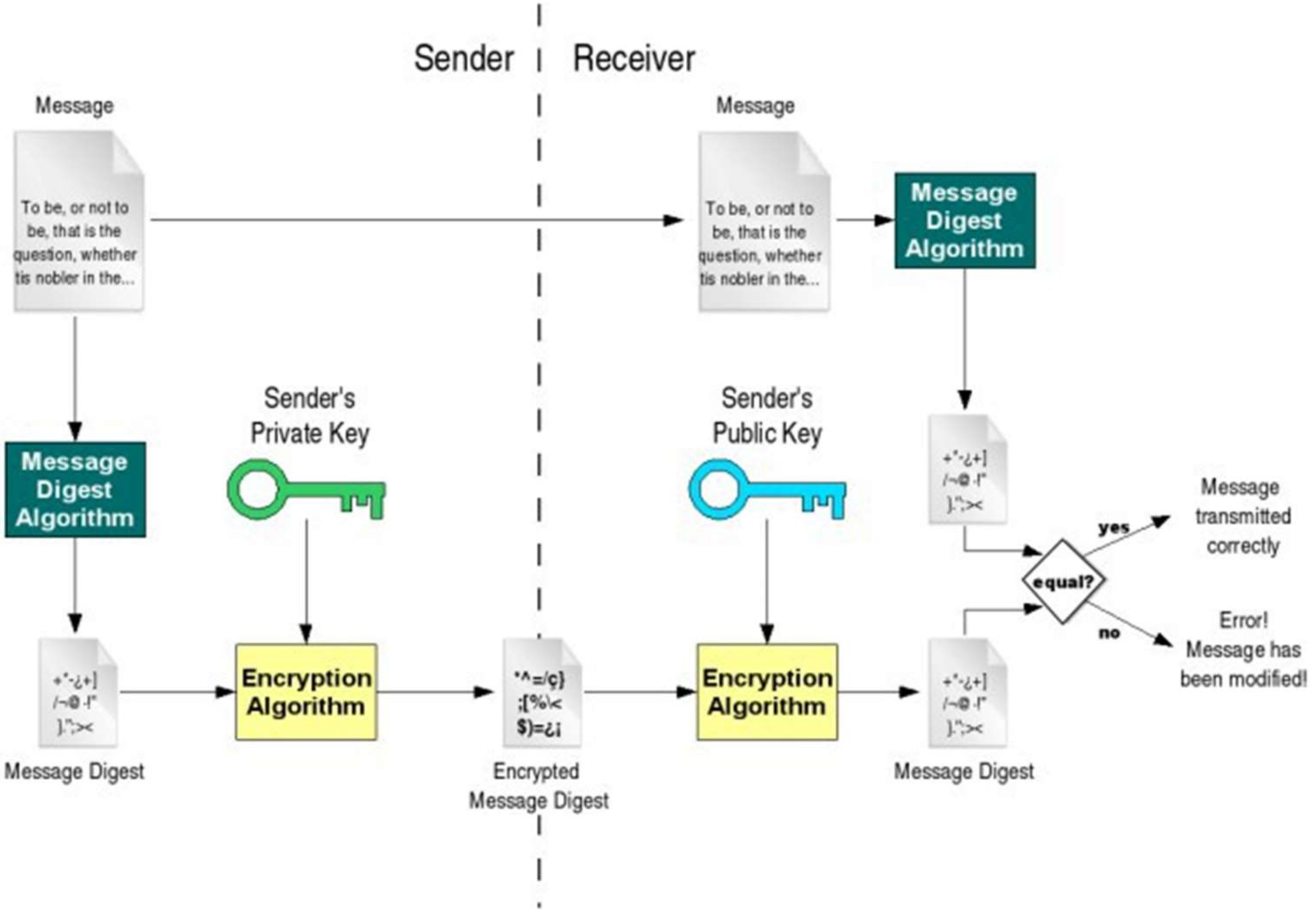
1) Generate a *Message Digest*
   - The message digest is generated using a set of hashing algorithms
   - A message digest is a 'summary' of the message we are going to transmit
   - Even the slightest change in the message produces a different digest
2) Create a Digital Signature
   - The message digest 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*
   - Decrypt the digital signature using the sender’s public key to obtain the message digest generated by the sender
2) Generate the Message Digest
   - Use the same message digest algorithm used by the sender to generate a message digest of the received message
3) Compare digests (the one sent by the sender as a digital signature, and the one generated by the receiver)
   - If they are not *exactly the same* => the message has been tampered with by a third party
   - 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