

CS35L – Winter 2019

Slide set:	8
Slide topics:	SSH
Assignment:	8



Reminders



No late submissions
allowed for Assignment
9 and Assignment 10



Deadline: Saturday,
March 16, 11:59 PM



Assignment 10 report &
presentation
submissions



Date: March 17, 2019



Day: Sunday ☹️



Time: 3:00PM – 6:00PM



Location: TBD



Exam format: Open book, open notes

No electronic devices:
calculators, smartphones,
smart watches, etc.



50% of course grade

Final Exams Reminder

Communication Over the Internet - Requirements



Confidentiality

- Message secrecy

Data integrity

- Message consistency

Authentication

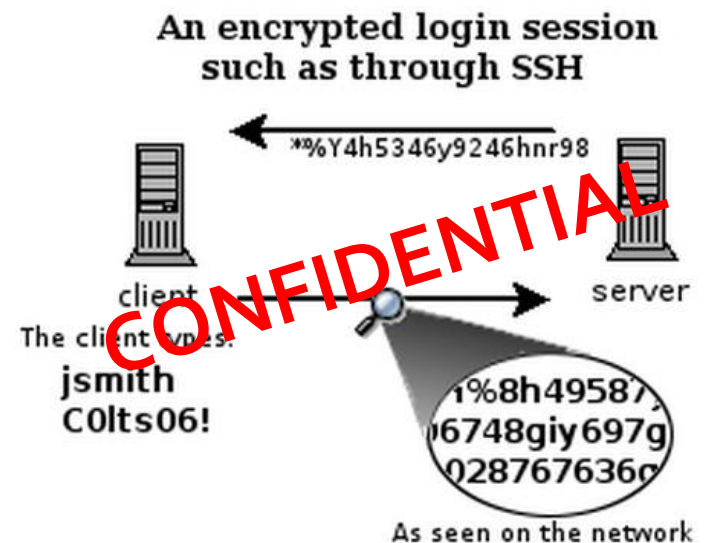
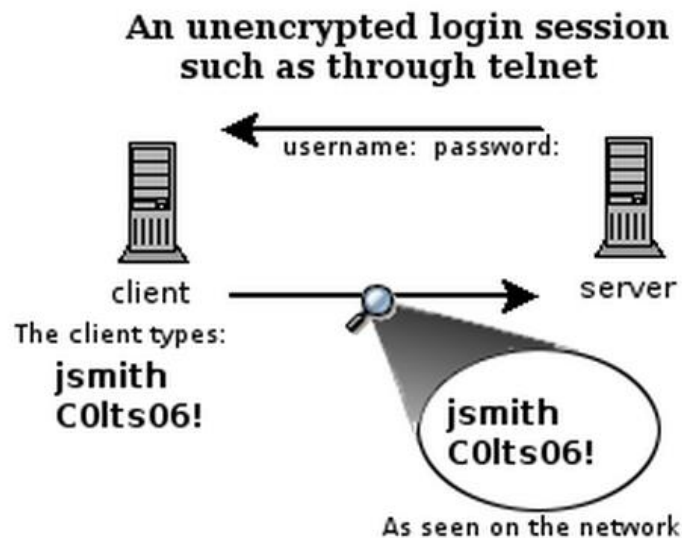
- Identity confirmation

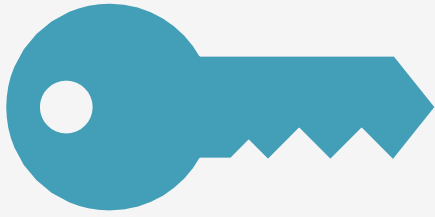
Authorization

- Specifying access rights to resources

SSH

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





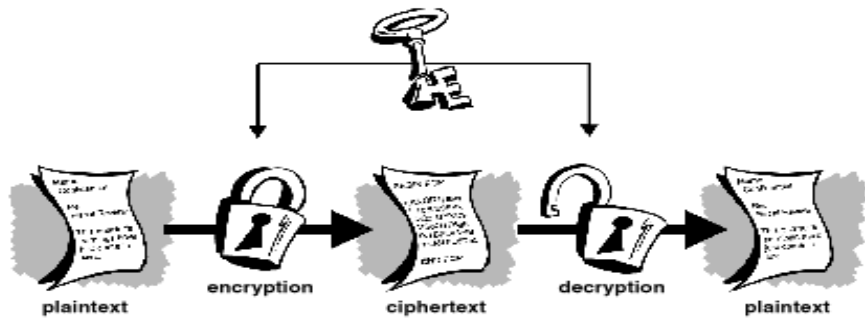
Encryption Types

Symmetric Key Encryption

- aka **shared key** or **secret key**
- Key used to encrypt is the same as key used to decrypt

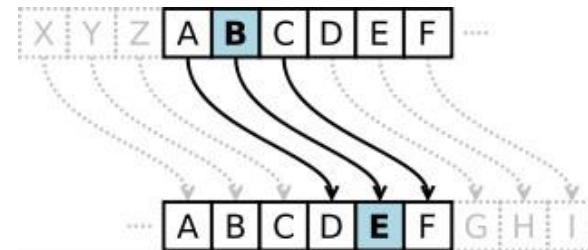
Asymmetric Key Encryption: Public/Private

- Two 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*
- **Private key is *never* published**



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
 - ABCDEFGHIJKLMNOPQRSTUVWXYZ
 - DEF...GHIJKLMNOPQRSTUVWXYZABC
 - Plaintext – SECRET. Ciphertext – VHFUHW
 - Key is 3 (number of shifts of the alphabet)
- **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 Algorithm – Rivest, Shamir & Adleman

Property used - **Difficulty of factoring** large integers to prime numbers

$$N = p * q \quad (3233 = 61 * 53)$$

N is a large integer and p, q are prime numbers N is part of the public key

http://en.wikipedia.org/wiki/RSA_Factoring_Challenge

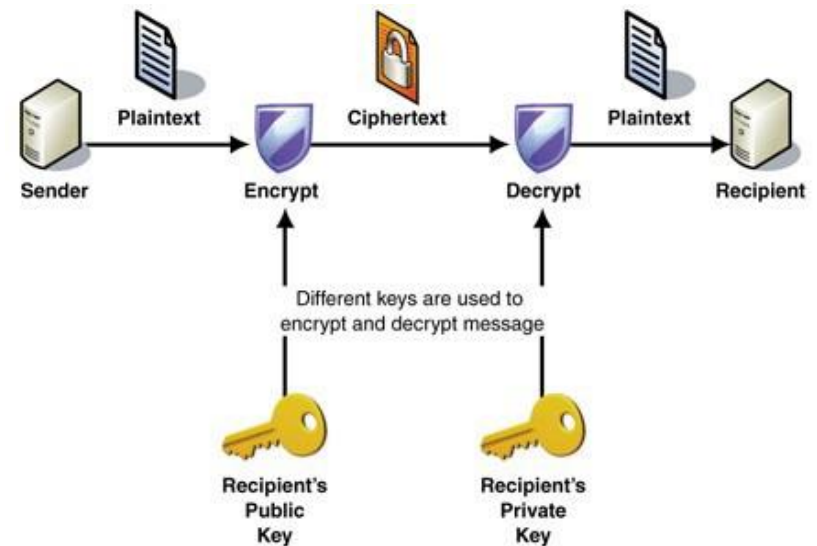



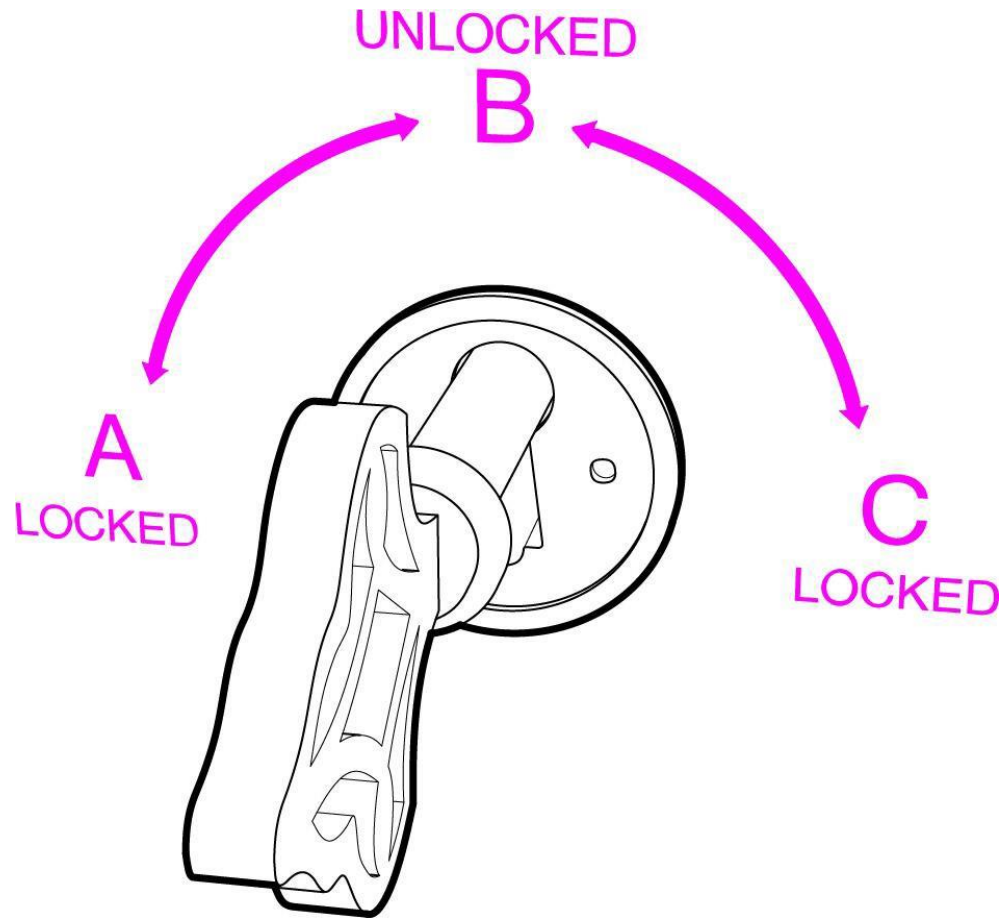
Image Source: MSDN

Unsolved problem in computer science

Can integer factorization be solved in polynomial time on a classical computer?

- The most difficult to factor in practice using existing algorithms are those that are products of two primes of similar size. For this reason, these are the integers used in cryptographic applications.
 - The largest such semiprime yet factored was [RSA-768](#), a 768-bit number with 232 decimal digits, on December 12, 2009.
 - This factorization was a collaboration of several research institutions, spanning two years and taking the equivalent of almost 2000 years of computing on a single-core 2.2 GHz [AMD Opteron](#).
- 

An Analogy





High-Level SSH Protocol

- Client ssh's to remote server
 - `$ ssh username@somehost`
 - **If first time talking to server; host validation occurs**

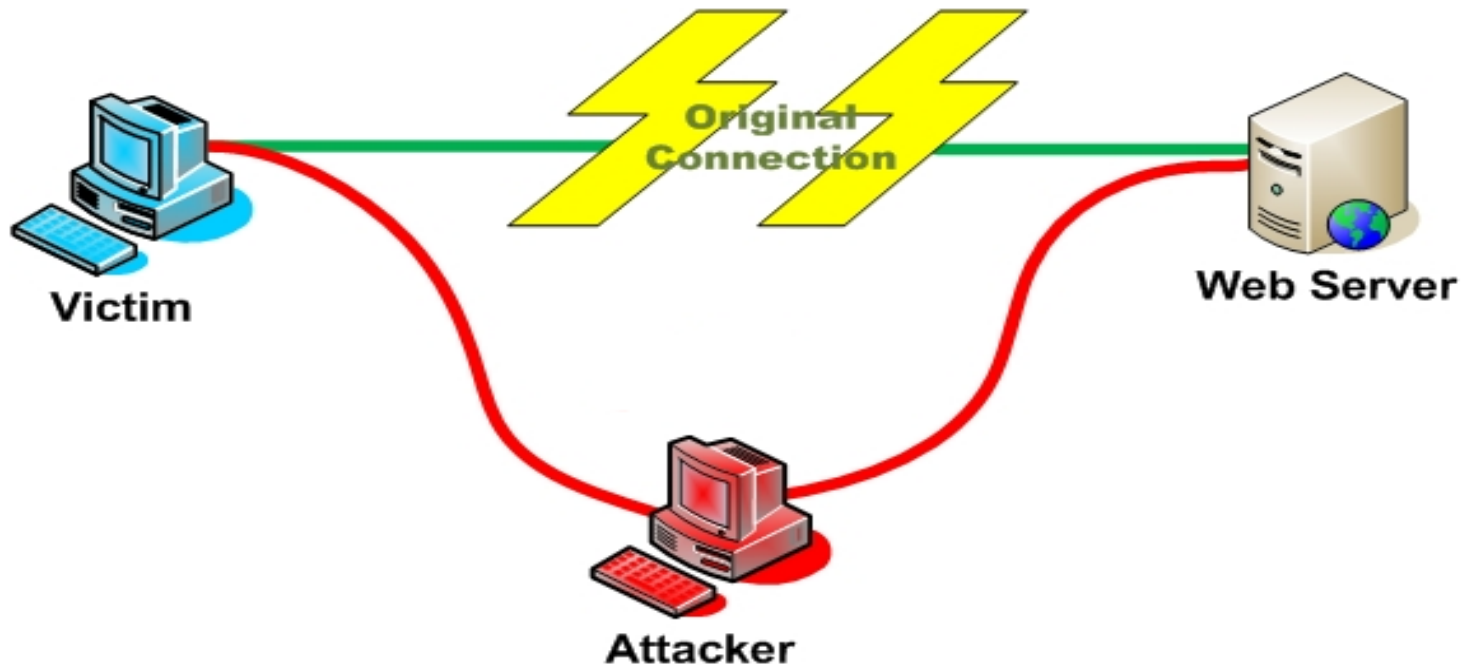
```
The authenticity of host 'somehost
(192.168.1.1)' can't be established.
RSA key fingerprint is
90:9c:46:ab:03:1d:30:2c:5c:87:c5:c7:d9:13:5d:75.
```

```
Are you sure you want to continue connecting
(yes/no)? yes
Warning: Permanently added 'somehost' (RSA) to
the list of known hosts.
```

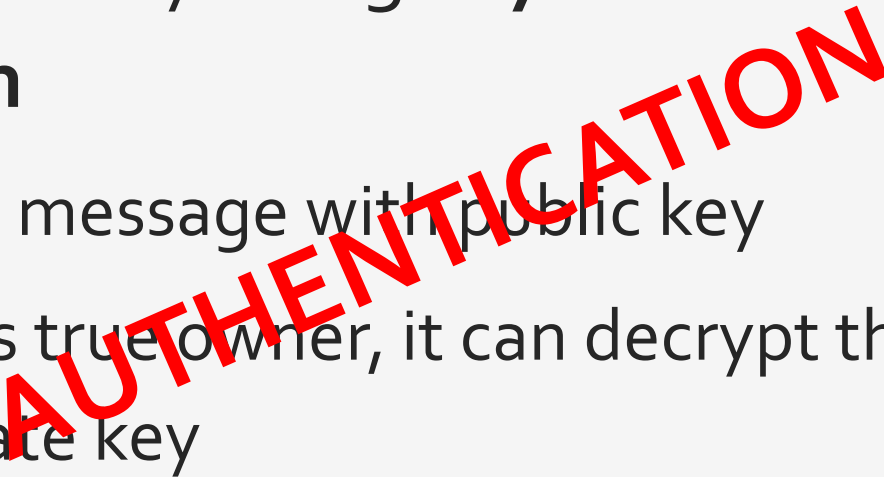

- ssh doesn't know about this host yet
- shows hostname, IP address and fingerprint of the server's public key, so you can be sure you're talking to the correct computer
- After accepting, public key is saved in `~/.ssh/known_hosts`

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
- 
- 
-

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
- 

- **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!)

User Authentication

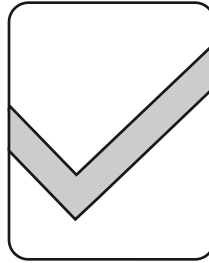
ssh-agent (passphrase- less ssh)

- 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 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)

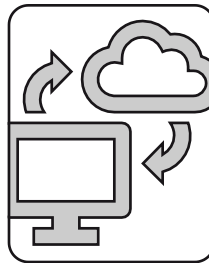
Digital Signature



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

- i.e., document was not changed
during transmission



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

Create a Digital
Signature

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

Attach digital
signature to
message and send
to receiver

Steps for Generating a Digital Signature

Compare digests

- Compare 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*

Generate the Message Digest

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

Recover the *Message Digest*

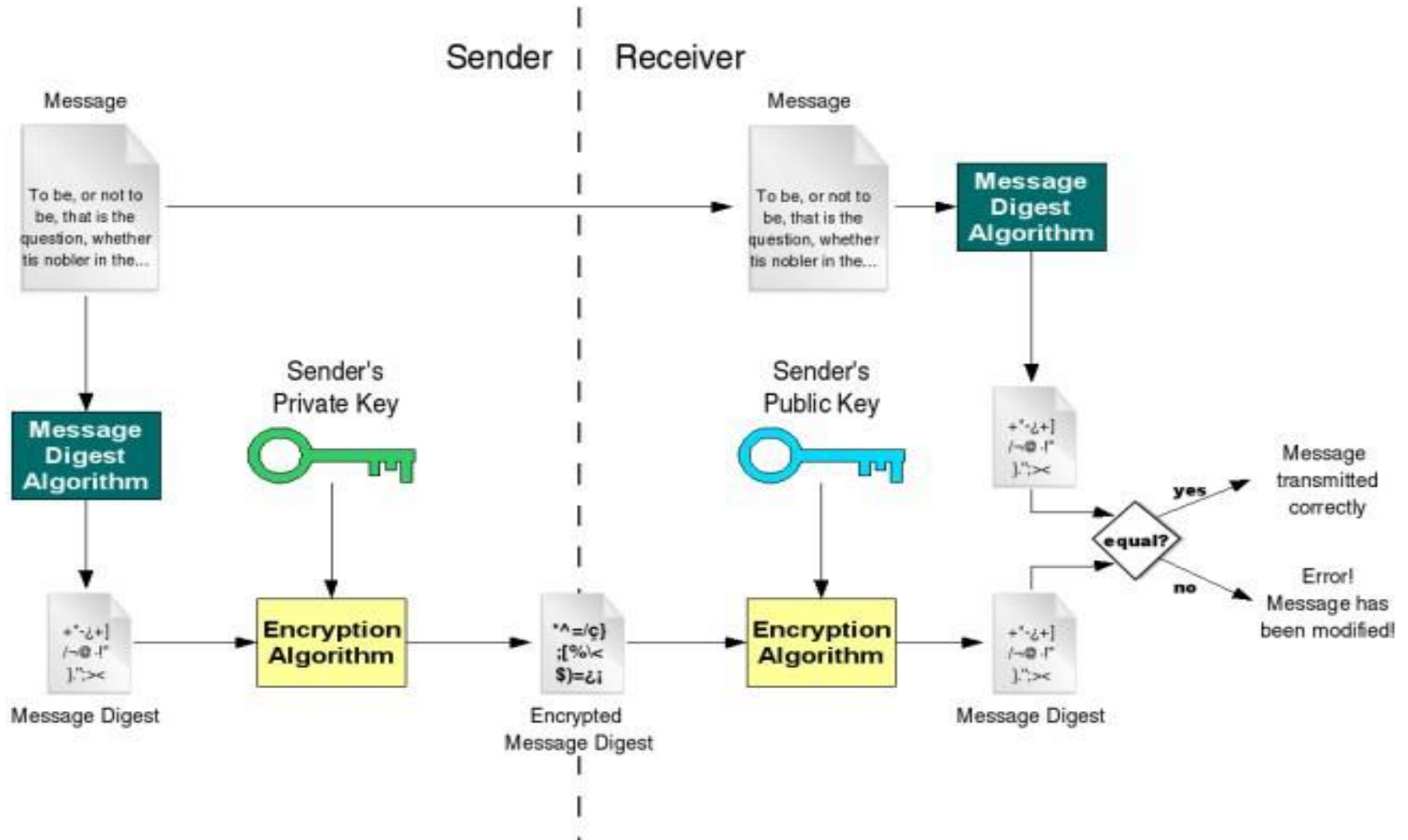
- Decrypt the digital signature using the sender's public key to obtain the message digest generated by the sender

*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.

Steps for Verifying a Digital Signature: Receiver

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
- 

Lab 8

- **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
-

- Debian
 - 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`

Lab Environment Setup

- **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>`
 - `$ sudo mkdir .ssh`
- **Change ownership and permission on .ssh directory**
 - `$ sudo chown -R username .ssh`
 - `$ sudo chmod 700 .ssh`

Server Steps



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`
- **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>`(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`
-

Homework 8

- Answer 2 questions in the file `hw.txt`
- A file `eeprom` that is a copy of the file `/sys/bus/i2c/devices/0-0050/eeprom` on your BeagleBone.
- <https://www.gnupg.org/gph/en/manual.html>
- Generate a key pair with the GNU Privacy Guard's commands (choose default options when prompted)
- Export public key, in ASCII format, into `hw-pubkey.asc`
- Use the private key you created to make a detached clear signature `eeprom.sig` for `eeprom`
- Use given commands to verify signature and file formatting
 - These can be found at the end of the assignment spec