Cyber Security Technologies

Session 9 – Cryptography II

Shawn Davis
ITMS 448 – Spring 2015

Slides contain original content from Davis, S., and Lidinsky, W.

Two Part Lecture

- Part I:
  - Cryptography Basics
  - Private Key Crypto
  - Codes
    - Substitution
  - Ciphers
    - Substitution
    - Transposition
- Part II:
  - Public Key Crypto
  - Computational Functions
  - Ciphers
  - Hashing
  - Secure Communications
  - Midterm Answer Review

Note

- You should be logged on to RADISH Win 8.1 VM
- Copy putty, puttygen, pageant, and WinSCP from M:\Tools folder to Win 8 Desktop
- Open up Kali for use later

Overview

Part I – Public Key Cryptography: Origins

Part II – Public Key Cryptography: Current Day

Part III – Secure Email Communications

Part IV – Secure Web Communications

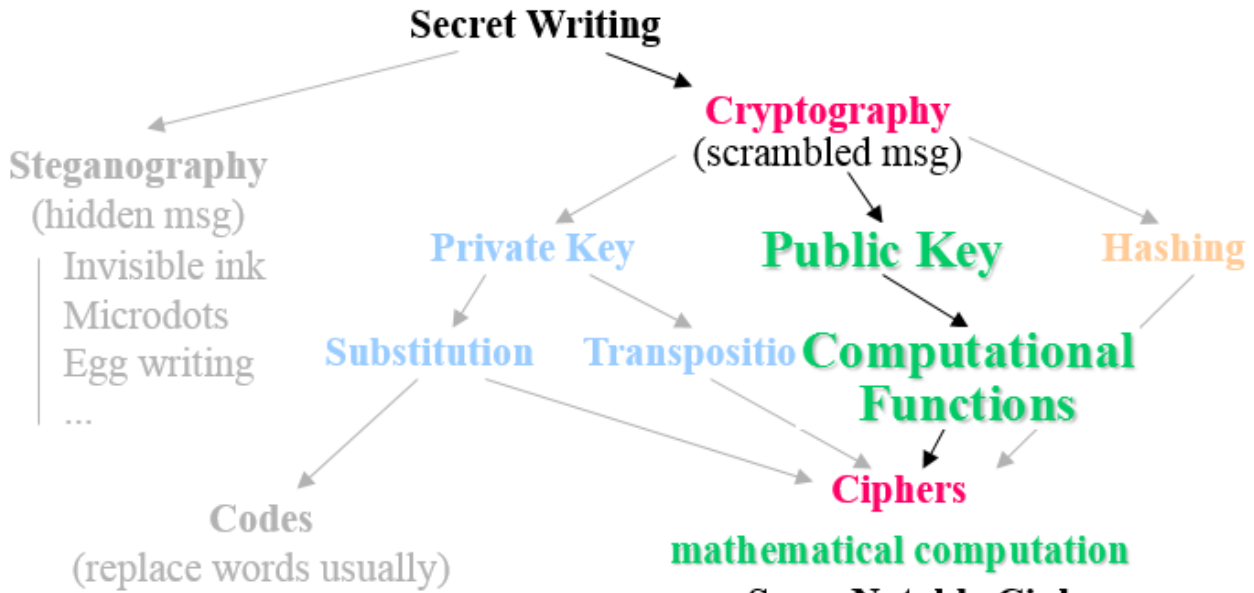
Last Time

- What is a major issue with private (symmetric) key encryption?
  - **Key distribution**
    - How the recipient(s) get the key
- Now we will consider public key encryption
  - Provides a solution to key distribution

Part I

Public Key Cryptography: Origins

A Taxonomy



Cryptography and **Ciphers** usually refer to the same things, with Cryptography being the science and Ciphers being the encoded entities. The only exception is **Codes.**

Private key ciphers use S and T.

Public key systems use CF.

S = substitution
T = transposition
H = hashing
CF = computational functions

Some Notable Ciphers:

Caesar (S)	Monoalphabetic (S)
Polyalphabetic (S)	Vigenère (S)
One-time pad (S)	Enigma (T&S)
DES (T&S)	RSA (T&S)
IDEA	MD5 (H)
RSA Pub (CF)	Diffie-Hellman (CF)
Elliptical (CF)	SHA-1 (H)
AES (T&S)	

Public Key Background

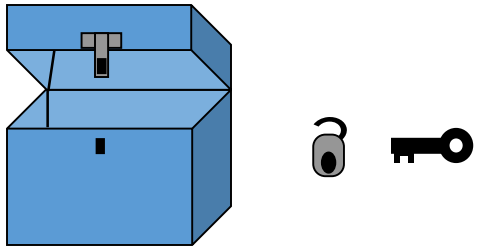
- A major problem with private key systems is the distribution of keys in a secure manner
  - This has plagued cryptographers for millennia
- A solution finally arrived in 1976
  - Diffie and Hellman (Stanford) published the 1st paper describing a public key system
  - Actually Whitfield Diffie figured it out
- It's one of these ideas that is so simple that it's a wonder that it was not thought of centuries ago
- It's a good example of how "thinking out of the box" is very important

Back to Private Keys for a Moment

- Before we go on to public key schemes, let’s go back to private keys in order to give some context
- Let's look at an analogy to encryption using private keys

Private Key Box Analogy

- Alice and Bob wish to privately communicate by putting their messages in a box and sending the box back & forth
- The box has a hasp, padlock and key



Alice has a 2nd key made for the padlock

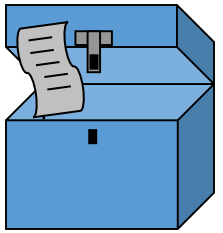
She keeps one

She sends the other key to Bob in a way the ensures that no one can intercept the key & make a copy of it

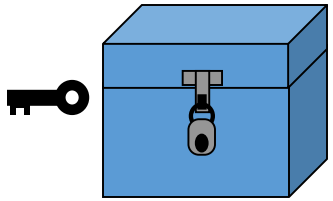
This of course is the key distribution issue. **It's a problem!**

Private Key Box Analogy

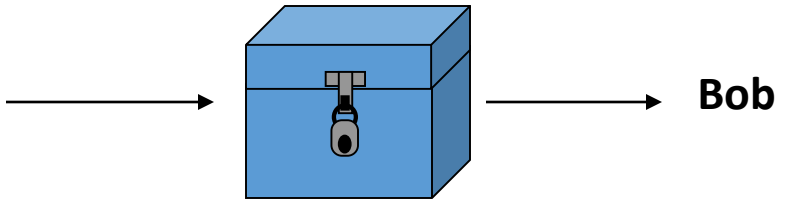
Alice puts a message in the box



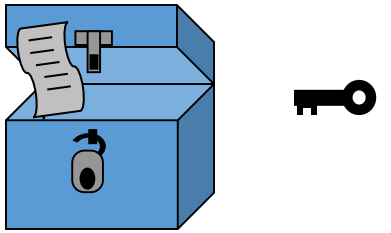
Closes and locks it with her copy of the key



Sends the locked box to Bob



Bob unlocks the box with his copy of the key and reads the message



Private Key Box Analogy

- Bob is sure that the message really came from Alice only if
  - The key was distributed in a secure fashion
- Ditto for Alice receiving messages from Bob

Issues With One Key Solution

- Eve might pick the lock on its way to Bob
  - Eve could then read the message and then reclose the box & forward it to Bob
    - Bob would not know that the message had been read
  - Alternately Eve could read the message and then discard the box and message
    - Bob would never get the message
- Maybe Bob and Alice would eventually decide that the message was either lost or intercepted

Issues With One Key Solution

- Eve might break the padlock or the box
  - Bob might then receive the box or the message or both but would know that it might have been read
  - Alternately Eve might discard the broken box
    - Bob might never get the box or the message or both
- Eve might simply discard the locked box without ever opening it
  - Bob would never receive the message

Issues With One Key Solution

- But even if none of the previous things ever happen, there would still be one major issue
  - Eve might intercept the key on its way to Bob, make a copy and then forward the key on to Bob
  - Bob would receive the key and not know that Eve had made a copy

Key distribution is a
<u>MAJOR</u> problem with private keys

How to Securely Send the Key?

- Sending the key securely perplexed Whitfield Diffie
  - How can Alice send Bob a secure message without sending the key to him first?

- Diffie's answer:

Don't send the key!

Don’t Send the Key?

- Bob sends Alice his box and its padlock in its open position
  - Alice has no Bob key. Bob keeps the only padlock key
- Alice puts the message for Bob in the box
- Alice then locks the box with the padlock and sends it back to Bob
- Bob receives the locked box, unlocks it and reads Alice's message
- This works!

- But Bob can’t send Alice messages

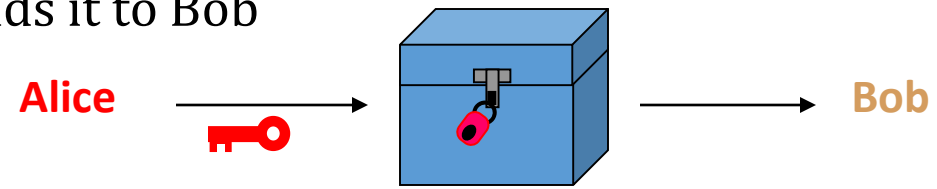
How Can We Send Messages in Both Directions without Distributing a Key?

- What if the there were two locks each with a different key?
  - Alice and Bob each have a different lock with a key
  - Each keeps their own key in a safe place

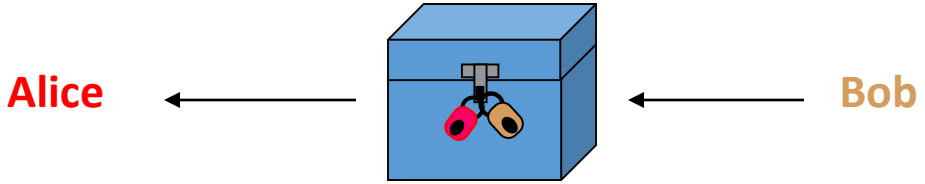
- **They never send any keys to anyone**

Two Locks & Two Keys

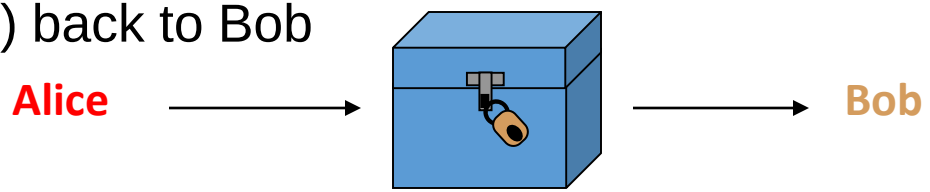
- Alice puts the message into the box, locks it with her padlock & key, and sends it to Bob



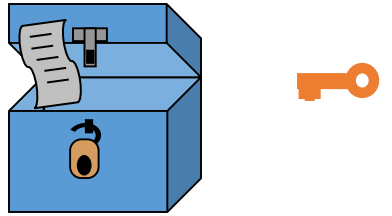
Bob puts his padlock on the box & sends it back to Alice



Alice removes her padlock and sends the box (still locked with Bob's padlock) back to Bob



Bob opens the box and reads the message



Issues With Two Locks & Two Keys

- Box must be sent three times
- Locks might still be picked or broken

But a major problem has been overcome.

Diffie & Hellman's Contribution

- Although there were problems, the implications of the two key scenario were **enormous!**

  It said that two people can securely exchange messages without exchanging keys

  *For the **very first time** ever recorded, it suggested that key distribution was not inevitable*

Rather obvious when one thinks about it

Amazing that it took thousands of years to discover it

But how can one implement such schemes?

Diffie & Hellman investigated a branch of mathematics called one-way functions

Asymmetric (Public Key) Ciphers

- Encrypt by treating the message as numbers and by transforming it using a one-way math function
- Two keys, one public (Kpu) and one private (Kpr)
- Math function requirements:
  - 1. Easy to generate a key pair, Kpr & Kpu for E & D
  - 2. $C = E_{Kpu}(P)$ is easy
    - E_{Kpu} is encryption algorithm using the public key
  - 3. $P = D_{Kpr}(C)$ is easy
    - D_{Kpr} is decryption algorithm using private key
  - 4. $P = D_{Kpu}(C)$ is computationally infeasible
    - D_{Kpu} is decryption algorithm using public key
  - 5. Infeasible to determine Kpr from Kpu, D and E

One-way Functions

- These are mathematical functions that are easy to calculate in one direction but hard to calculate in the reverse direction
- Modular mathematics was the basis for Diffie and Hellman's approach
  - Hellman made the modular math discovery
  - Diffie, Hellman, and Merkle (who later joined them in the work) refined the work

Diffie-Hellman Key Exchange

How can Alice and Bob securely distribute a private key using one-way functions?

Alice and Bob agree on the one-way function $7^x(\text{mod}11)$

They agree about this in public so Eve can know it

ALICE	***BOB***
1. Alice chooses a number A *Let A = 3* *Alice keeps her number secret*	1. Bob chooses a number B *Let B = 6* *Bob keeps his number secret*
2. Alice calculates $a = 7^A(\text{mod}11) = 343(\text{mod}11)$ $= \mathbf{2}$	2. Bob calculates $b = 7^B(\text{mod}11) = 117649(\text{mod}11)$ $= \mathbf{4}$

Diffie-Hellman Key Exchange

3. Alice sends a = 2 to Bob

Transmissions are in the clear.

3. Bob sends b = 4 to Alice

Anyone can see these numbers.

But is very difficult to determine anything even if you know 2, 4, and $7^x(mod11)$ because you don't know the keys A & B that Alice & Bob kept to themselves.

ALICE	***BOB***
Alice now has A=3 & b=4	Bob now has B=6 & a=2
4. Using b & A, Alice calculates	4. Using a & B, Bob calculates
$b^A(mod11) = 4^3(mod11)$	$a^B(mod11) = 2^6(mod11)$
$= 64(mod11)$	$= 64(mod11)$
$= \mathbf{9}$	$= \mathbf{9}$

The number 9 is the private key.

Learning a Symmetric Key Without Exchanging It

- Diffie, Hellman, and Merkle had developed a key "exchange" process that could be done publicly but still be secure
  - But it exchanged symmetric (not asymmetric) keys
- Also it was not really an exchange of keys
  - A way for two sides to determine the same key from
    - Public information that they exchanged
    - Private information that they alone knew

Asymmetric Keys

- Diffie was not satisfied because:
  - Both Alice and Bob needed to be available
    - Alice needed info from Bob
    - Bob needed info from Alice
  - The private key couldn't be determined by either of them without the info from the other
- With symmetric keys, the same key is used to encrypt and decrypt the message
- Diffie developed the idea of asymmetric keys
  - Two related keys: **one to encrypt and one to decrypt**

Part II

Public Key Cryptography: Current Day

One Lock & Two Keys

- Matched pair (Public & Private keys)
  - Public keys are shared and not secret
  - Private keys are never shared and are kept secret
- Confidentiality
  - Public Key encrypts data
    - Then private key decrypts that data
- Authentication
  - Private Key encrypts data
    - Then public key decrypts that data

RSA Algorithm

- RSA stands for Rivest, Shamir and Adleman
  - MIT
  - Published in 1978
- It is widely used and implemented today
- Provided one-way function that is reversible

Confidentiality (but No Authentication)

1: Alice & Bob each create key pairs

2: They exchange their public keys

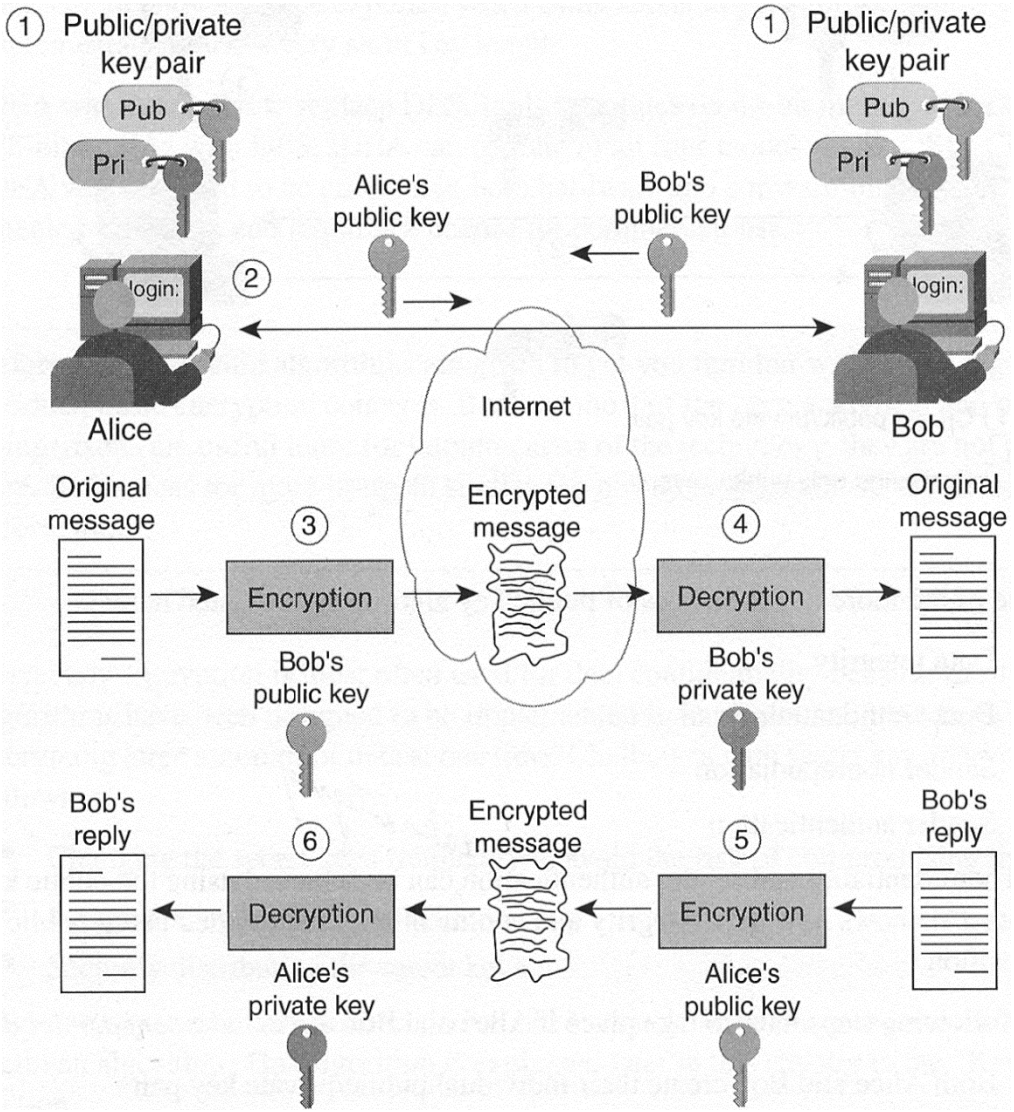
3: Alice sends message to Bob encrypted with Bob's public key

4: Bob receives Alice's message and decrypts it with his private key

5: Bob encrypts a reply to Alice with her public key

6: Alice receives Bob's message and decrypts it with her private key

- Confidentiality achieved!
- Problem: No authentication!
  - *Easy for Eve to send an encrypted message to Bob claiming that she is Alice*



Authentication (but No Confidentiality)

1: Alice & Bob each create key pairs

2: They exchange their public keys

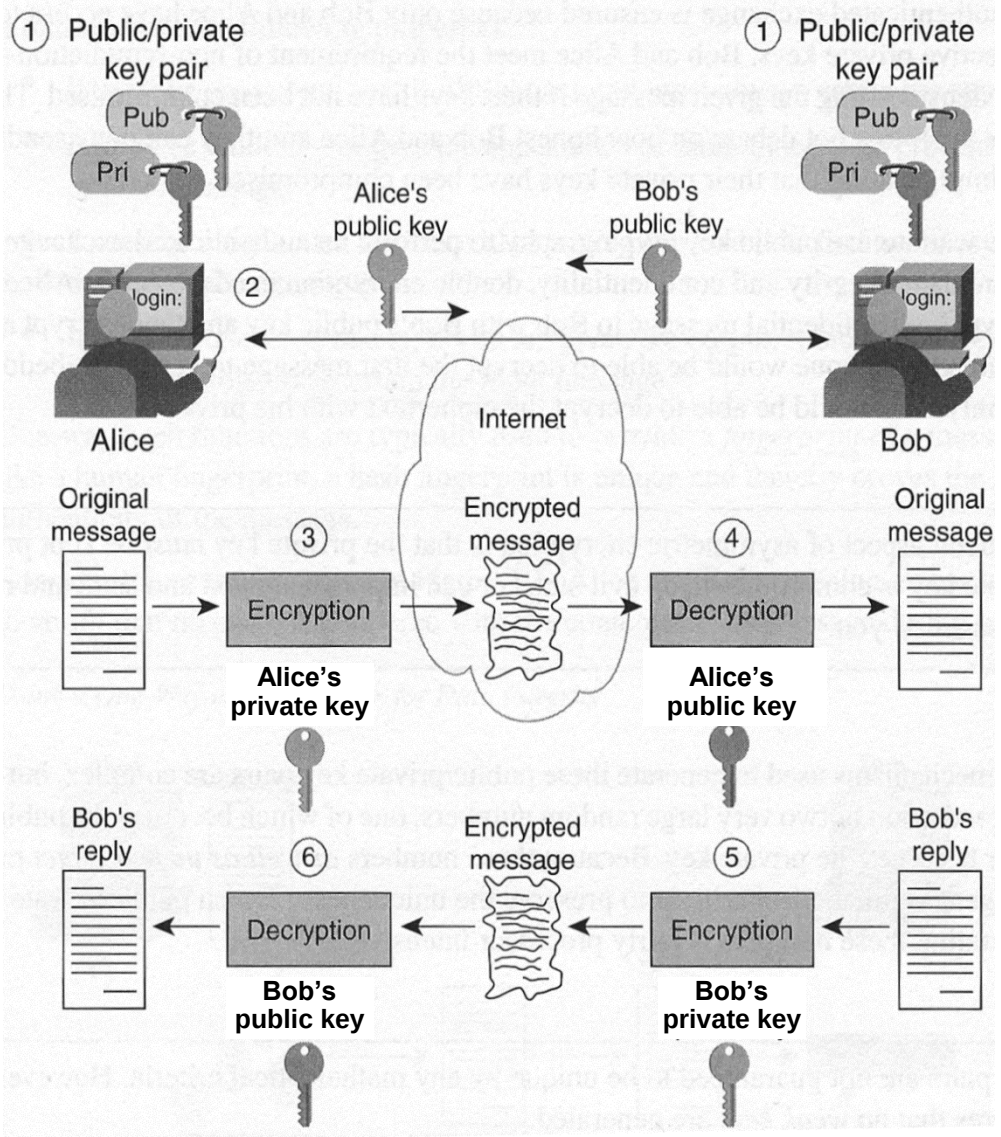
3: Alice sends message to Bob encrypted with Alice's private key

4: Bob receives Alice's message and decrypts it with Alice's public key

5: Bob encrypts a reply to Alice with his private key

6: Alice receives Bob's message and decrypts it with his public key

- Authentication achieved!
- Problem: No confidentiality!
  - *Easy for Eve to decrypt either Alice's or Bob's messages because she (and everyone else) has their public keys*



Integrity?

- The prior two slides showed how a message could be sent that:
  - Kept the message secret (Confidentiality)
  - Verified the sender (Authentication)

- What can we use to ensure the message hasn't changed??? (Integrity)
  - Hashing!

Hash Functions

- Provide input data as a:
  - Message
  - File
  - Etc.
- Perform hash function against input data to generate hash value (aka message digest)

Uses of Hash Functions

- Hash functions are used for message integrity
  - Allows receiver to determine if message has been corrupted
- While a hash function can also be used for message authentication or digital signatures, it is not as secure as if it were combined with encryption
  - Hashes can be used with keys for these purposes
  - Called message authentication codes (MACs)

Properties of a Hash Function

- Input data can be of any length but hash function generates a fixed length hash value
- Input data hashed using a specific hash function will always return the same hash value
- Computationally infeasible to determine input data, given hash value
  - Basically, hashes are resistant to reverse-engineering
- Very low probability that two different sets of data will generate the same hash value

Common Hash Function Algorithms

- MD5
- SHA-1
- SHA-256
- SHA-512
- RIPEMD-160
- HMAC

Example of Hash Uniqueness

- Go to this site:
  - www.hashgenerator.de
  - Select the MD5 algorithm
  - Type a sentence into the top message box while watching the hash value

- Even changing one bit or byte of the input data completely changes the hash value!

Question

- What uses hashing to guarantee message integrity and somewhat provides authentication???
  - Digital Signatures!

Digital Signature

- Allows recipient to verify:
  - Message has not changed (Integrity)
  - Who the message came from (Authentication) ***

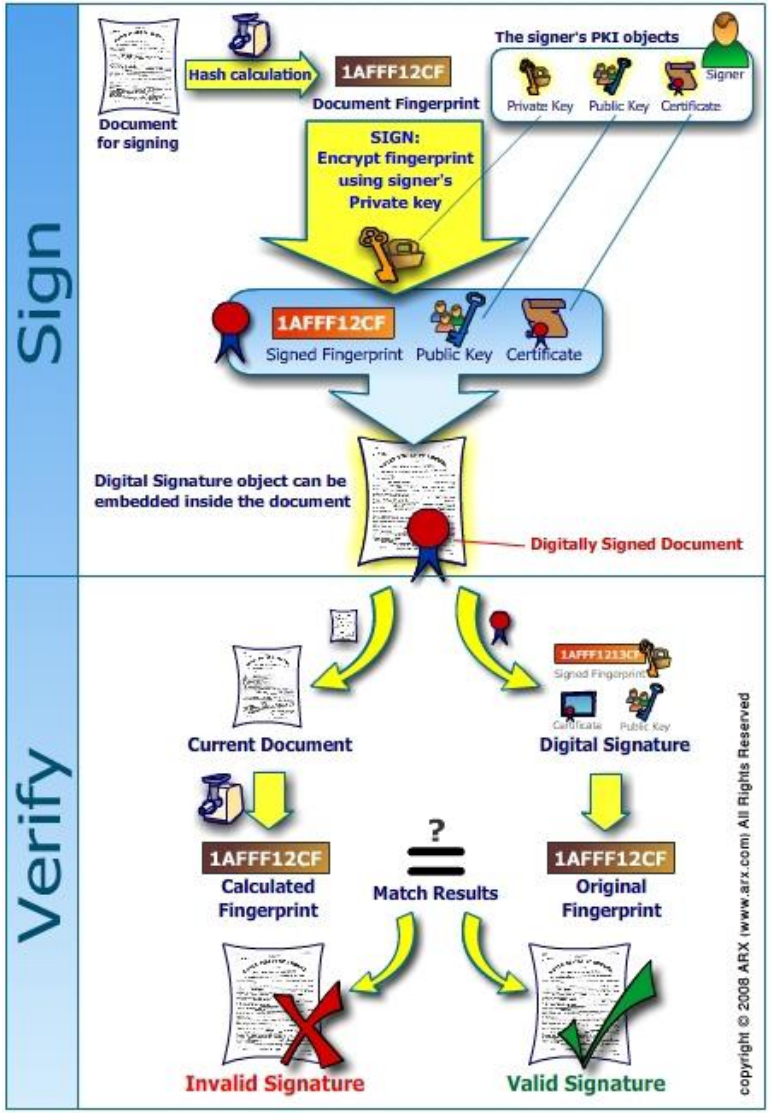
***Really, the Digital Signature just verifies which individual's private key was used to sign the message. It is possible that an attacker is pretending to be the sender after stealing their private key.

How Does a Digital Signature Work?

- Run plaintext message through hash function to form the hash
- Encrypt hash with sender's private key to create the digital signature
- Send plaintext message and digital signature to recipient

Integrity & Authentication* (but No Confidentiality)

1. Alice runs plaintext message through hash function to create the hash
2. Alice encrypts hash with her private key to create digital signature
3. Alice sends plaintext message and digital signature to Bob
4. Bob hashes message to obtain his own hash of it
5. Bob uses Alice's public key to decrypt digital signature to view the hash Alice created
6. Bob verifies that the hash he took matches Alice's hash



http://www.arx.com/files/Brochure-images/Digital-signatures2.jpg

Integrity & Authentication* (but No Confidentiality)

- We know that:
  - Anyone could use Alice's public key to decrypt the digital signature to reveal the hash
  - No one could change it though without knowing Alice's private key (which is secret)
- If Bob's hash that he took and Alice's hash of the message match:
  - The message has not been changed since being signed
  - Whoever signed the message used Alice's private key*

Integrity, Authentication*, & Confidentiality

- How do we get all three???
  - Alice also encrypts message to Bob with Bob's public key
  - Follow all steps two slides ago

Integrity & Authentication*, & Confidentiality

1. Alice runs plaintext message through hash function to create the hash
2. Alice encrypts message to Bob with Bob's public key
3. Alice encrypts hash with her private key to create digital signature
4. Alice sends encrypted message and digital signature to Bob
5. Bob decrypts message with his secret key
6. Bob hashes decrypted plaintext message to obtain his own hash of it
7. Bob uses Alice's public key to decrypt digital signature to view the hash Alice created
8. Bob verifies that the hash he took matches Alice's hash

(Graff, 2001, pp. 69-70)

Issue Remains: Authentication*

- How do we trust that the person who created the digital signature is really Alice and not an imposter???
- Also, how do we enforce non-repudiation so the sender cannot deny they sent the message???
  - Use Digital Certificates!

Digital Certificates

- Helps recipients determine if the digital signature and public key belong to the sender
- Allows exchange of public keys
- Digital Certificate consists of:
  - Serial Number
  - Issuer
  - Dates Valid
  - Sender's Public Key
  - Identity information for the sender
    - Name, User ID, etc.
  - Other users' digital signatures that have vouched that the identify information is bound to the public key
    - Creates trust

How do you distribute the certificates???

- Certificate Server
  - Database that allows users to submit their certificates as well as retrieve other individual's certificates

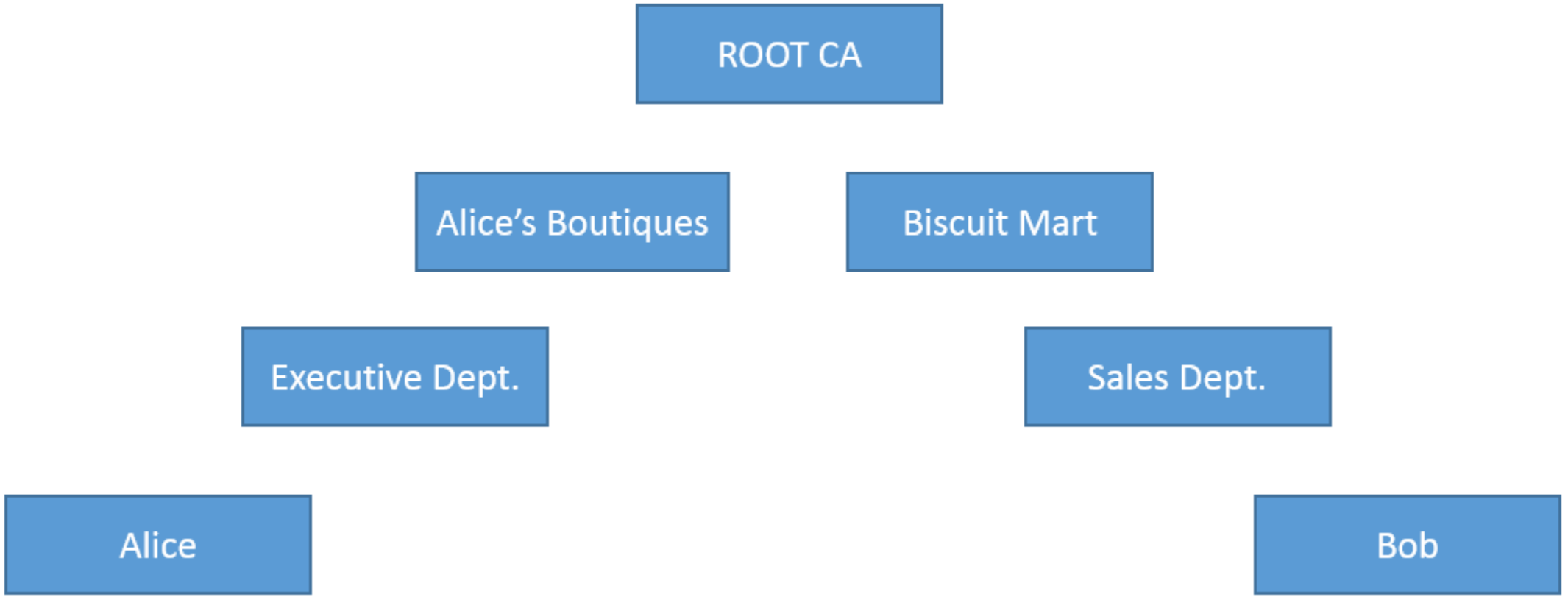
- Public Key Infrastructure (PKI)
  - Allows trust between parties that might not know each other previously
  - X.509 Standard which describes:
    - Ability to issue, revoke, store, retrieve, and trust certificates

http://www.pgpi.org/doc/pgpintro/#p14

Certificate Authority (CA)

- Uses PKI
- CA binds parts of certificate (user's public key, information about them) by using its secret certification key to encrypt them
- Individuals can decrypt the certificate by using the CA's public verification key

Certificate Hierarchies



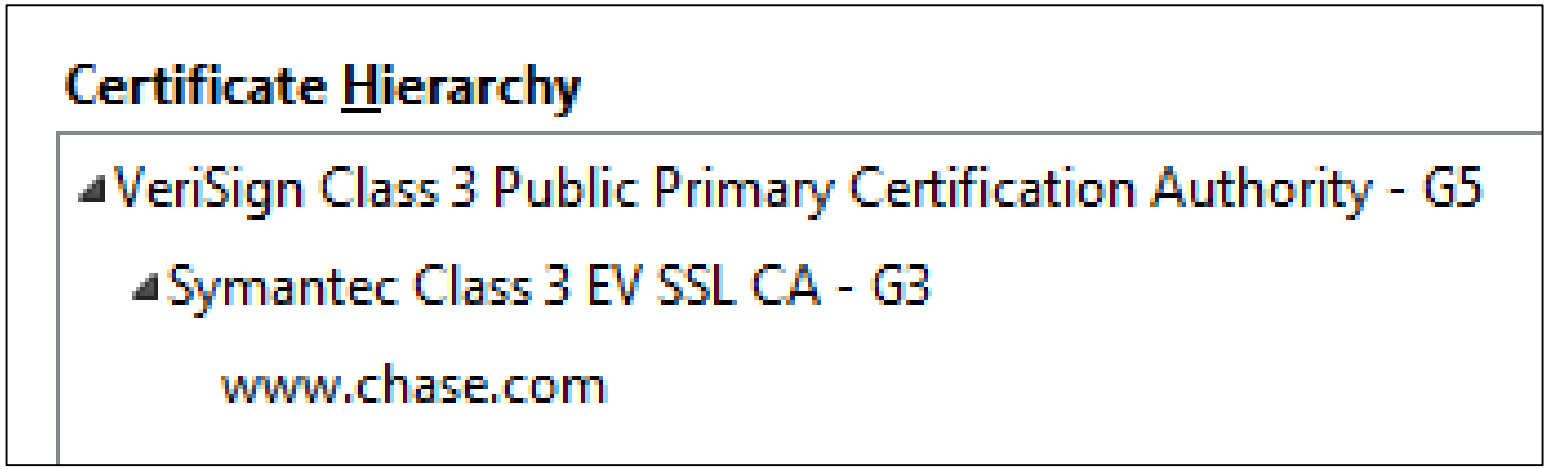
(Graff, 2001, pp. 89)

Certificate Hierarchies

- Root CA issues certificate to Alice's Boutiques and Biscuit Mart
- Alice's Boutiques issues certificate to their Executive Dept.
- Biscuit Mart issues certificate to their Sales Dept.
- Executive Dept. issues certificate to Alice
- Sales Dept. issues certificate to Bob

Certificate Hierarchies

- Example for Chase.com:



Popular Root CAs

- Symantec (bought VeriSign)
- Comodo
- Go Daddy
- GlobalSign

Invalid Digital Certificates

- Let's say Alice loses her private key or her company goes bankrupt
- What can the CA do???
  - Revoke the certificate and place it on a Certificate Revocation List (CRL)

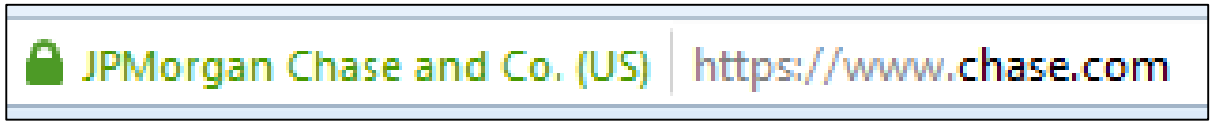
(Graff, 2001, pp. 83)

Example of how Certificates are used when browsing the Web

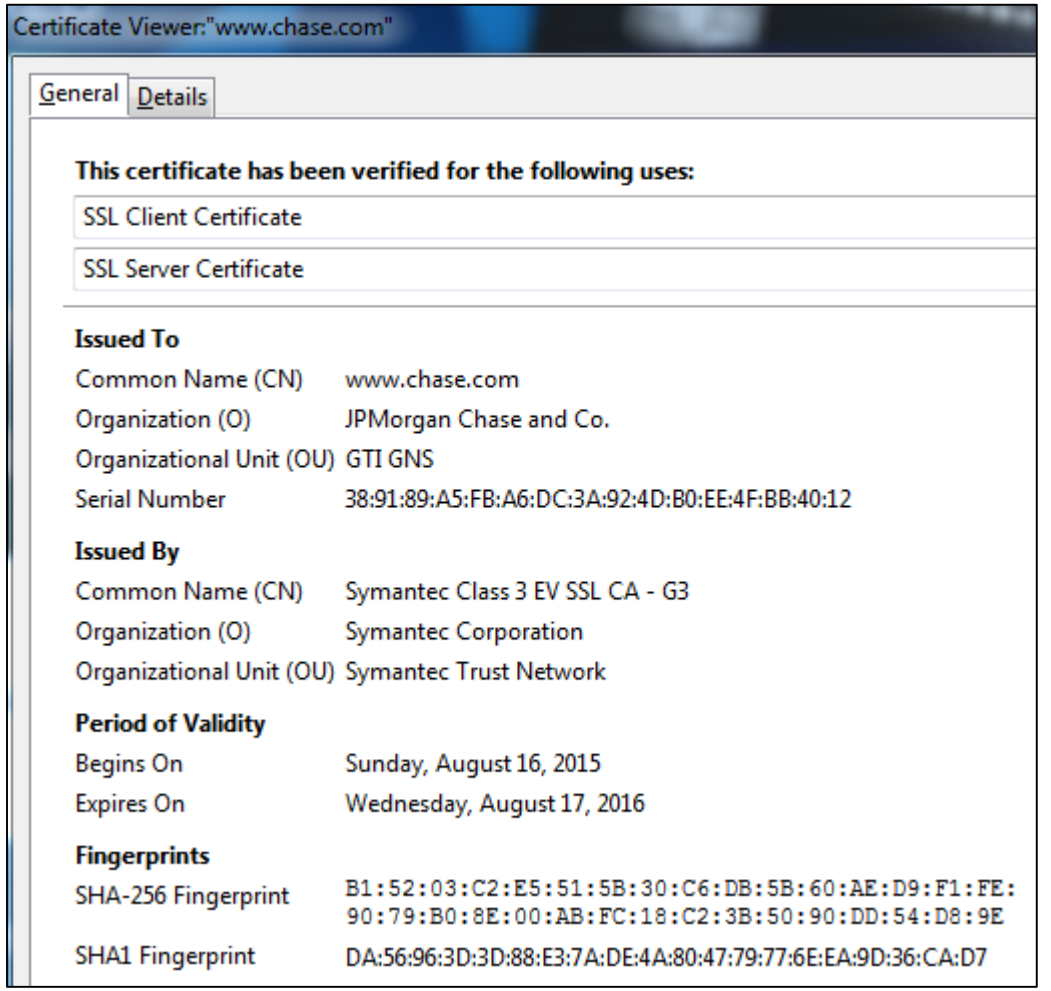
- An employee at Chase applies for a digital certificate from Symantec
- Chase has to prove to Symantec that the employee works for and has authorization from the real Chase Bank through various verification mechanisms
- Once Symantec is satisfied that the employee's request is legitimate:
  - Symantec creates a digital certificate for Chase
  - Symantec signs that certificate with their private key

Example of how Certificates are used when browsing the Web

- You visit https://www.chase.com and notice that a lock symbol appears:



- You see that the website's SSL certificate was signed by a CA's Private Key, in this case, Symantec



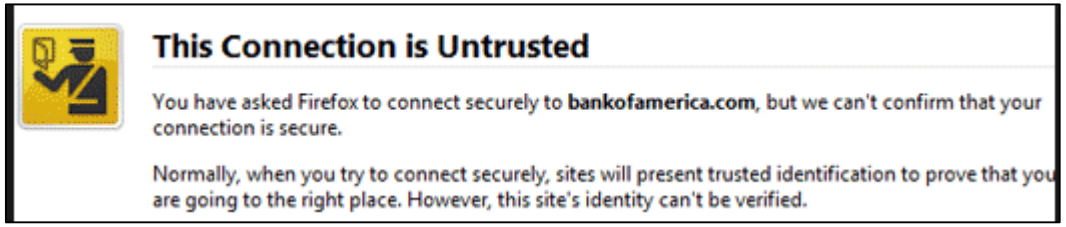
Example of how Certificates are used when browsing the Web

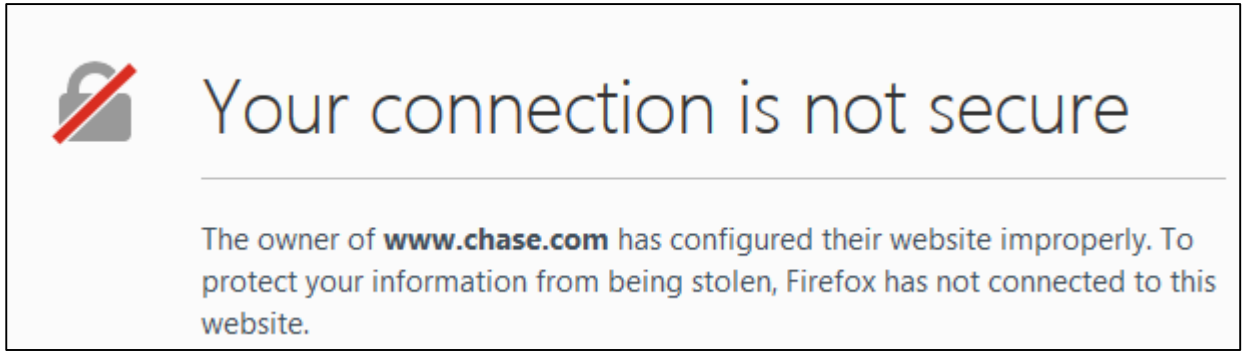
- When installing an OS or a browser, your computer is loaded with the public keys of various CAs that are known to be trusted
  - These are stored in a Certificate Store
- In the prior slide, Chase sent its certificate which was signed by Symantec's private key
- Your browser will check its certificate store for Symantec's public key and try to decrypt Chase's certificate
- If it can, then the public key is valid

Example of how Certificates are used when browsing the Web

- If the browser cannot decrypt Chase's certificate, then the public key is invalid
- Your browser will then display a warning

Example of how Certificates are used when browsing the Web





Invalid Digital Certificates

- You may receive a warning if:
  - The certificate is on a revocation list (CRL)
  - The website self signed their certificate and didn't use a trusted CA loaded into your computer's certificate store
  - An attacker is using a MiTM attack against your computer and is trying to sign the certificates with their own private key

Invalid Digital Certificates

- Your browser will often give you a warning to leave the site or to proceed and store a security exception
  - The exception is stating you are allowing the disabling of the warning
- Do not disable these warning or proceed unless you are the owner of the site and are using a self-signed certificate

Invalid Digital Certificates

- If you are using a self-signed certificate or want to use a proxy like Burp and don't want to receive warnings, what can you do???
- When you self-sign a certificate you also create a CA certificate
- You can install that CA certificate as a trusted root certificate into your host's certificate store
  - The Lenovo Superfish vulnerability occurred due to Lenovo installing Superfish's CA certificate into the host's certificate store which then trusted all websites

Uses of CA Digital Certificates

- Website certificates (as in the prior example)
- Software Code signing
- Software Update signing
  - MS, Mac OS X, Linux all use CAs

One Potential Issue Remains

- The Root CA is at the top of the heirarchy
- Who issues and signs the certificate for the Root CA???
  - The Root CA itself (since nothing is above it in the hierarchy)
- Root CA must have strongest protection on its certificate key and its procedures for generating certificates!

(Graff, 2001, pp. 85)

What Happens if Root CA Certification Key is Compromised???

- All trust is lost!!!

Example of a Huge Root CA Breach:

- DigiNotar
  - Attacker started on outside of network and worked inward
  - Attacker eventually accessed certificate generating mechanism and issued rogue certificates such as for google.com
  - User's visiting who thought they were visiting Google sub-domains were victims of MiTM interception
    - Great investigation report:
      - http://www.rijksoverheid.nl/ministeries/bzk/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update.html

Everything Together

- Now let's look at what happens when the following are all provided:
  - Confidentiality
  - Integrity
  - Authentication
  - Non-Repudiation

- The following scenario is for Alice sending a messages to Bob

Integrity & Authentication, Confidentiality, & Non-Repudiation

1. Alice sends information about herself and her public key to the CA
2. The CA issues her a digital certificate
3. Alice runs plaintext message through hash function to create the hash
4. Alice encrypts message to Bob with Bob's public key
5. Alice encrypts hash with her private key to create digital signature
6. Alice sends encrypted message, digital signature, and her digital certificate to Bob

Integrity & Authentication, Confidentiality, & Non-Repudiation

7. Bob decrypts message with his secret key
8. Bob decrypts Alice's digital certificate to obtain Alice's public key
9. Bob hashes decrypted plaintext message to obtain his own hash of it
10. Bob uses Alice's public key to decrypt digital signature to view the hash Alice created
11. Bob verifies that the hash he took matches Alice's hash

(Graff, 2001, pp. 73-74)

Integrity & Authentication, Confidentiality, & Non-Repudiation

- We now know that:
  - The message has not been changed since being signed
  - Whoever signed the message used Alice's private key
  - Bob obtained Alice's public key from a trusted digital certificate that bound the public Key to Alice's identity
  - Alice can't deny she signed the message because her secret key was used to create the digital signature

(Graff, 2001, pp. 74)

Question???

- What are some email and web standards and schemas that offer:
  - Confidentiality
  - Integrity
  - Authentication
  - Non-Repudiation

Part III

Secure Email Communications (PGP & S/MIME)

Examples:

- Secure Email
  - PGP/GPG
  - S/MIME
- Secure Web Browsing
  - HTTPS with SSL/TLS

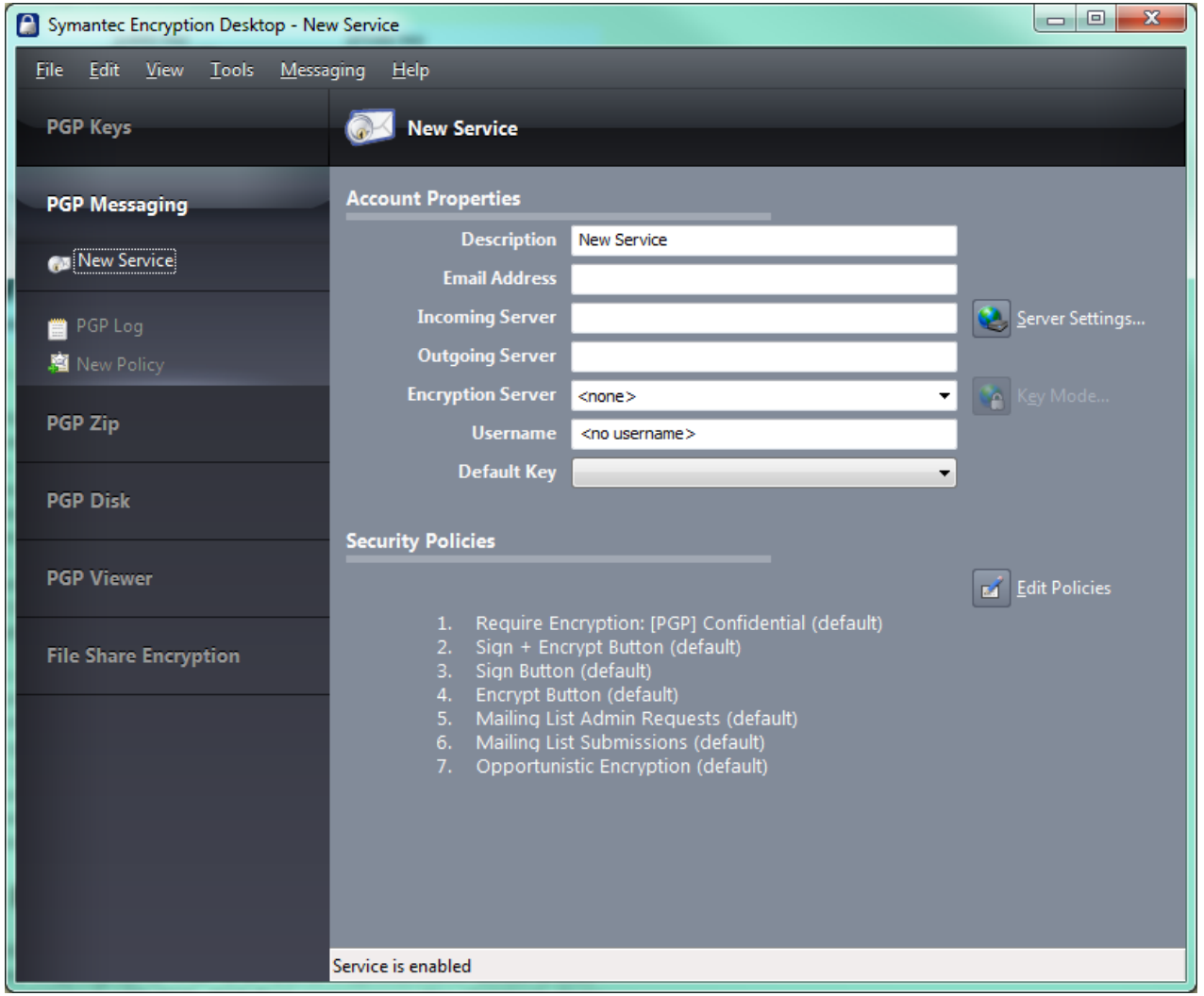
PGP (Pretty Good Privacy)

- Hybrid Cryptosystem
  - Uses RSA for asymmetric encryption to share key and IDEA for symmetric encryption of data
  - Requires PKI to distribute and manage digital certificates
- Often used for sending and receiving secure emails

Symantec Encryption Desktop

- Symantec bought PGP
- Allows for:
  - Full disk encryption
  - File share encryption
  - Secure email

Symantec Encryption Desktop



S/MIME

- Secure/Multipurpose Internet Mail Extensions
- Hybrid Cryptosystem
  - Uses RSA for asymmetric encryption to share key and AES for symmetric encryption of data
  - Requires PKI to distribute and manage digital certificates
- Often used for sending and receiving secure emails

Part IV

Secure Web Communications (HTTPS)

HTTPS

- Uses TLS/SSL for encryption
  - Uses asymmetric encryption to share keys and symmetric encryption for data (various ciphers and algorithms used)
  - PKI with X.509 Digital Certificates

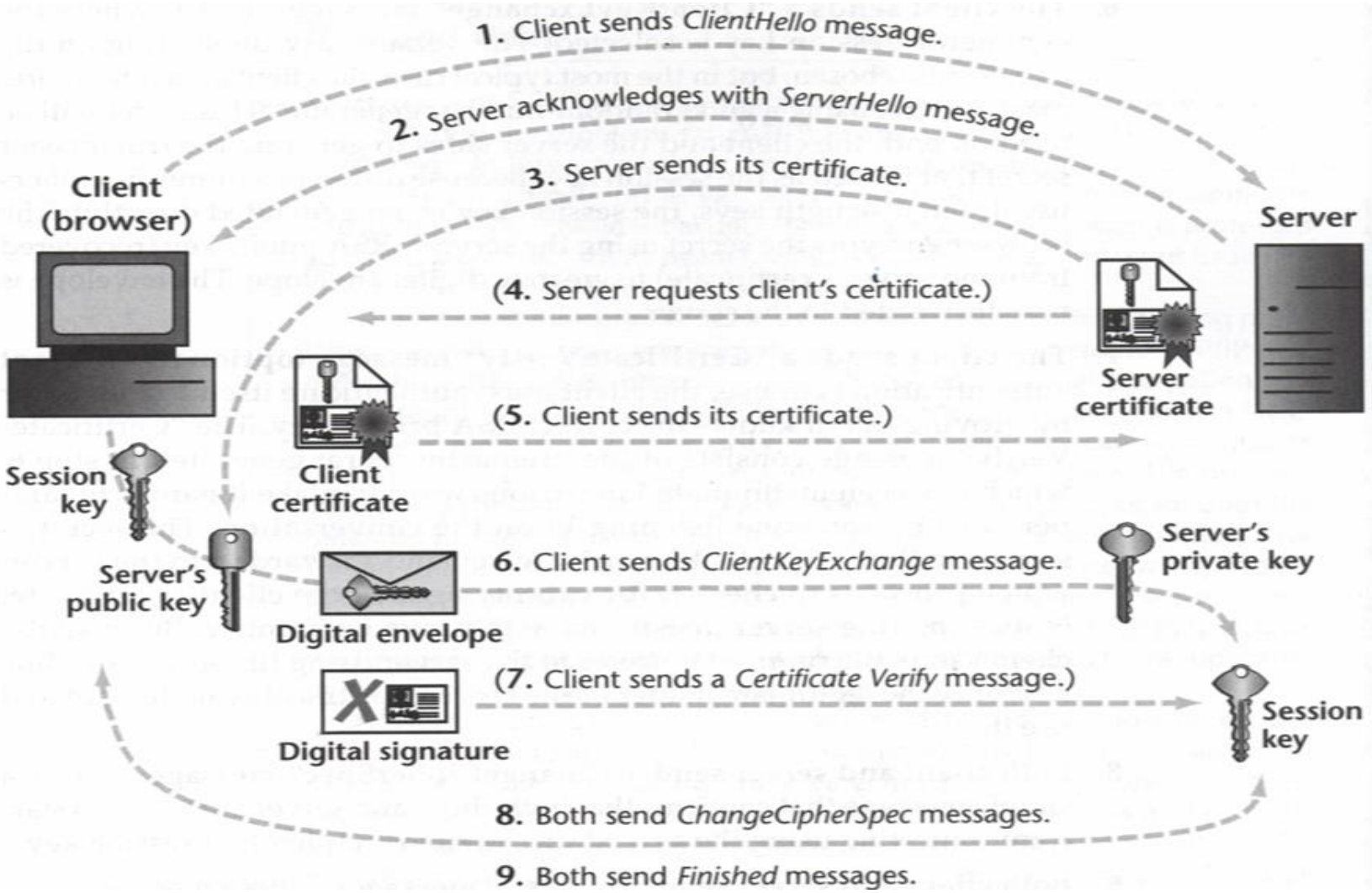
Difference between TLS and SSL???

- Transport Layer Security (TLS) is basically a newer version of Secure Sockets Layer (SSL)
- Most browsers use TLS today

HTTPS

- Ensures you are communicating with intended web server
- Ensures messages aren't changed in transit
- Ensures data is confidential
- Ensures non-repudiation

SSL Full Handshake



SSL Full Handshake

1. Client sends Server a *ClientHello* saying that it wants to start a session

This message lists the capabilities of the client, including SSL version, what cipher suites it supports, and data compression methods it supports

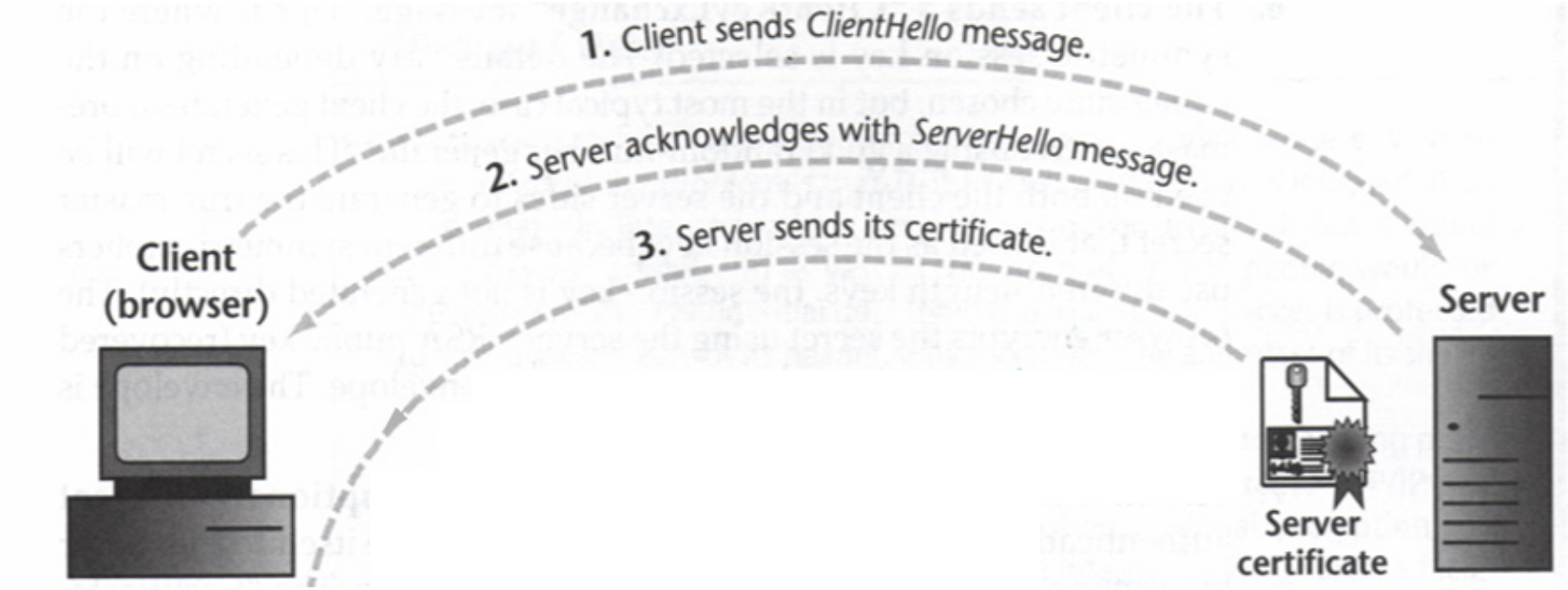
2. Server responds with a *ServerHello*

This is a hash of the Client message encrypted using the Server's private key

Msg. includes SSL version, cipher algorithm, data compression methods & a session ID

If client can meet the Server's specs., continue. Otherwise, Client terminates connection

SSL Full Handshake

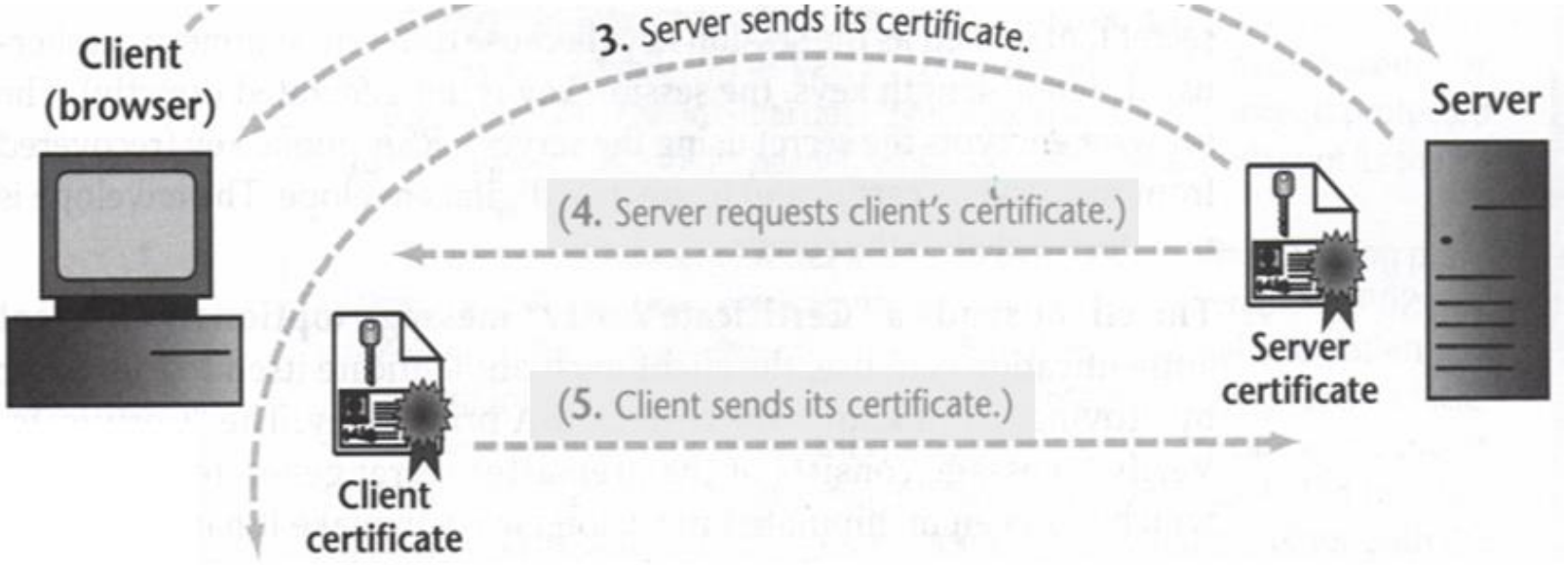


Next the Client decrypts the message using the Server's public key and compares it against the actual message in ciphertext

If they match, the Client is reasonably certain that it is communicating with the Server

3. If the Server's key is certified, it will send its *Certificate*.

SSL Full Handshake

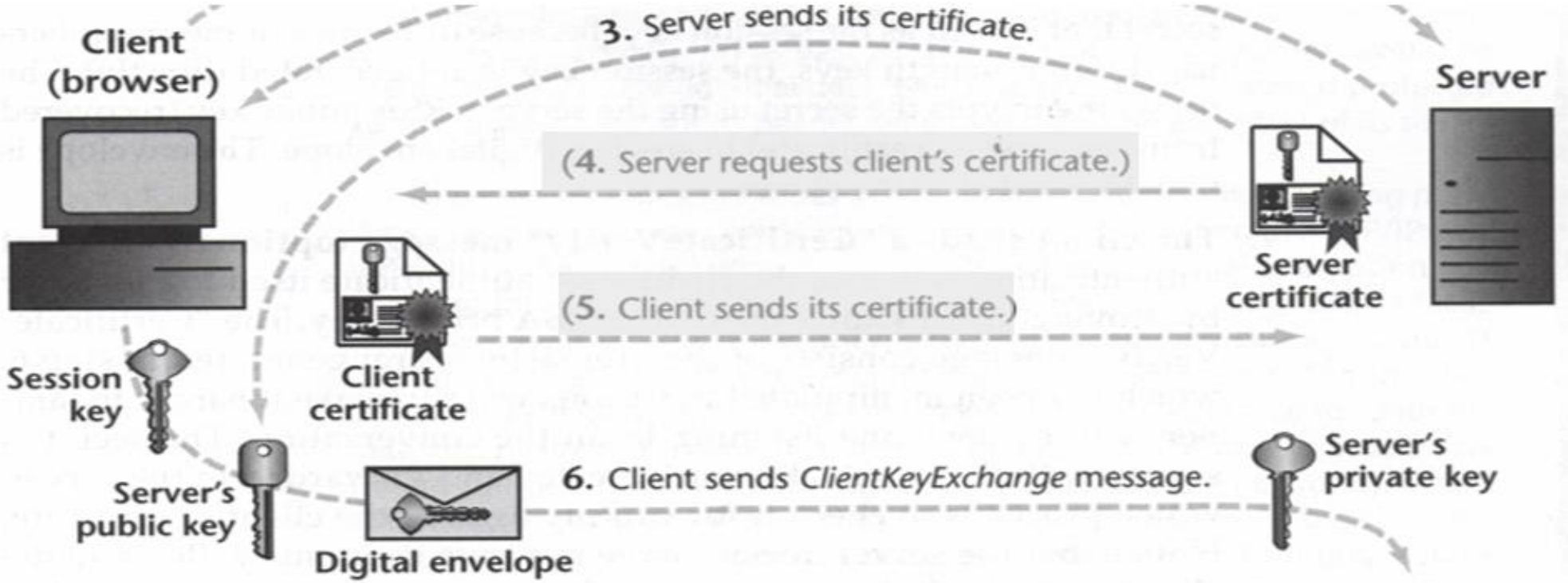


4. Server requests Client's Certificate (if enabled)

5. Client responds with its Certificate

If client has no Certificate, it tells the Server
Server then has option of terminating the session
Server will usually continue, with or without a Certificate

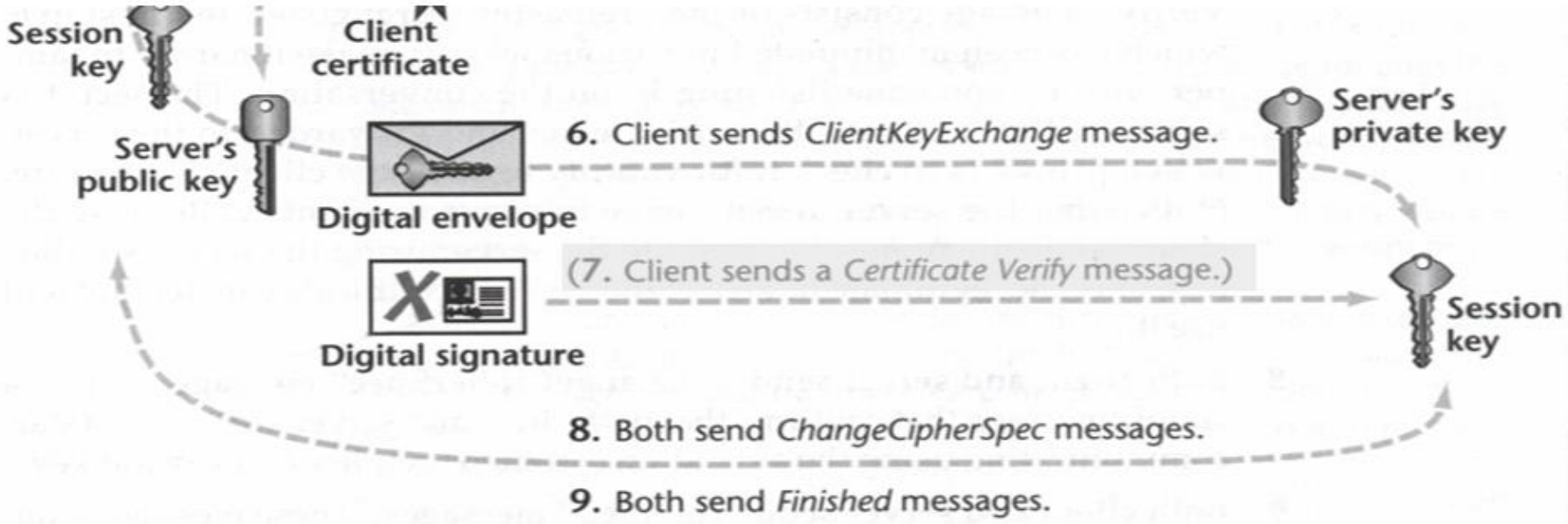
SSL Full Handshake



6. Client sends *ClientKeyExchange* msg. encrypted with Server's public key

Includes a symmetric private "session key" chosen by the Client
This is called the "digital envelope"

SSL Full Handshake



7. Client sends *CertificateVerify* msg if the Client has a Certificate.
This certifies to the Server that the Client is who it really is

8. Client & Server both send *ChangeCipherSpec* messages
Confirms to both sides that the session will now start

Now the application above SSL can proceed
e.g., https, ftps, telnets…

9. At the end of the session, both send *Finish* messages

Example: Client Hello

```
159.53.116.62                    TLSv1.2    259 Client Hello

⊟ Secure Sockets Layer
  ⊟ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 200
    ⊟ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 196
        Version: TLS 1.2 (0x0303)
      ⊞ Random
        Session ID Length: 0
        Cipher Suites Length: 30
      ⊞ Cipher Suites (15 suites)
        Compression Methods Length: 1
      ⊞ Compression Methods (1 method)
        Extensions Length: 125
```

Example: Client Hello (Available Cipher Suites)

```
Cipher Suites (15 suites)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc14)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc13)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
```

Example: Server Hello

```
192.168.1.232                    TLSv1.2  1434 Server Hello
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 85
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 81
      Version: TLS 1.2 (0x0303)
      Random
      Session ID Length: 32
      Session ID: f109115d357ce0415e24e76bc659793c880b53a2ccadfd9e...
      Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
      Compression Method: null (0)
      Extensions Length: 9
```

Cipher Suite

```
Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256
```

- TLS 1.2 = Cipher Suite Version
- RSA = Asymmetric key exchange algorithm (RSA)
- AES_128_GCM = Symmetric Encryption algorithm for data transfer (GCM with 128 bit key size)
- SHA256 = Message Authentication Code (256 bit SHA hash for integrity

Example: Certificate Exchange

```
192.168.1.232                    TLSv1.2    410 Certificate

Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 3012
    Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 3008
      Certificates Length: 3005
      Certificates (3005 bytes)
        Certificate Length: 1672
        Certificate: 308206843082056ca003020102021038918 9a5fba6dc3a92... (id-at-commonName=www.chase.com,id-at-organiz
        Certificate Length: 1327
        Certificate: 3082052b30820413a003020102021 07ee14a6f6feff2d37f... (id-at-commonName=Symantec Class 3 EV SSL CA
          signedCertificate
            version: v3 (2)
            serialNumber: 0x7ee14a6f6feff2d37f3fad654d3adab4
            signature (sha256WithRSAEncryption)
            issuer: rdnSequence (0)
            validity
            subject: rdnSequence (0)
            subjectPublicKeyInfo
              algorithm (rsaEncryption)
              Padding: 0
              subjectPublicKey: 3082010a0282010100d8a1657423e82b64e232d733373d8e...
```

Example: Key Exchange

```
159.53.116.62                    TLSv1.2    372 Client Key Exchange, Change Cipher Spec

Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 262
    Handshake Protocol: Client Key Exchange
      Handshake Type: Client Key Exchange (16)
      Length: 258
      RSA Encrypted PreMaster Secret
        Encrypted PreMaster length: 256
        Encrypted PreMaster: 64f07d91e67ab050ba55f0c37268a6f08c0edbcb9bc4b740...
  TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
```

Example: Encrypted Data Channel Open

```
159.53.116.62                    TLSv1.2  1164 Application Data

Secure Sockets Layer
  TLSv1.2 Record Layer: Application Data Protocol: http
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 1105
      Encrypted Application Data: 00000000000000001f060d028191dd8db0099d79bc28b1ffa...

0000  a4 4c 11 ef 1a 25 d4 c9  ef 53 55 8b 08 00 45 00   .L...%.. .SU...E.
0010  04 7e 27 ee 40 00 80 06  00 00 c0 a8 01 e8 9f 35   .~'.@... .......5
0020  74 3e 17 a9 01 bb fe da  82 09 a3 c1 6c f5 50 18   t>...... ....l.P.
0030  fb c5 da 74 00 00 17 03  03 04 51 00 00 00 00 00   ...t.... ..Q.....
0040  00 00 01 f0 60 d0 28 19  1d d8 db 00 99 d7 9b c2   ....`.(. ........
0050  8b 1f fa d1 c3 7d 59 8c  2b dd a1 89 de ef 35 3a   .....}Y. +.....5:
0060  8a a0 b4 3f 44 15 a7 5d  20 85 a4 4a 3c dd b9 d6   ...?D..]  ..J<...
0070  b2 62 8b 0c 74 77 29 e2  b4 c2 44 bd 0f 7c 79 e1   .b..tw). ..D..|y.
0080  7b 5c 39 db eb 5e 40 f0  6f 42 4a c3 30 06 b2 71   {\9..^@. oBJ.0..q
0090  72 a6 d1 a2 93 d1 b3 64  3f 93 e4 01 22 c3 81 2b   r......d ?..."..+
00a0  bf 32 44 fa bf 9a 06 54  a9 0c 78 b2 59 c2 9a ae   .2D....T ..x.Y...
00b0  86 31 e6 f5 68 b3 c7 bc  c6 62 d0 cb 78 3e b7 db   .1..h... .b..x>..
00c0  00 fa 3c 93 2d 8d f6 0b  2e 0b a0 00 e4 57 94 e9   ..<.-... .....W..
00d0  65 0a f2 e7 ff 25 6f 9c  00 31 31 67 b7 88 37 c8   e....%o. .11g..7.
00e0  2a cb 1f 62 b5 9e 67 79  73 e1 e8 6d 41 ef 89 76   *..b..gy s..mA..v
00f0  cb 54 b5 df db 20 a9 ed  72 98 24 d4 2d 9f eb ff   .T... .. r.$.-...
0100  63 0f 2a 31 b8 06 20 42  62 21 4b cf e7 ae e0 85   c.*1.. B b!K.....
0110  20 16 38 82 04 bf 97 8a  29 0c a9 77 b8 b5 57 34    .8..... )..w..W4
0120  da 64 de 05 ef 02 16 46  bb 53 5f 79 56 9c 0d 30   .d.....F .S_yV..0
0130  c7 06 d7 35 4c d4 0f 96  67 f5 bd e8 c7 b4 f9 ea   ...5L... g.......
0140  58 05 46 0b 0c 00 74 55  3c 83 a5 0b 2f 68 20 15   X.F...tU <.../h .
0150  e1 a5 db 48 99 ba 82 03  79 10 03 68 6b 83 5b b6   ...H.... y..hk.[.
0160  bf 1a 9d f4 11 9c 61 76  c6 e6 5f 36 b5 64 10 f4   ......av .._6.d..
```

SSH

- Secure Shell (SSH) provides an encrypted tunnel to remotely administer a system, move files, etc.
- Secure replacement for telnet, ftp, rlogin, rcp, and rsh

Hands On – Logging in with SSH Key/Pair

- If you haven't already:
  - Copy putty, puttygen, pageant, and WinSCP from M:\Tools folder to Win 8 Desktop
- Open terminal in Kali Linux
- Install OpenSSH Server (Might already be installed)
  apt-get install openssh-server
- Start the OpenSSH service and then confirm
  service ssh start
  service ssh status

SSH Public/Private Keys

- Users can either simply enter an IP address into Putty and use their password to authenticate
- Another option is to use an RSA key pair

- We are going to set up a key pair so that we can logon securely with Putty from Win8 to Kali

Hands On – Logging in with SSH Key/Pair

- Create btaylor user
  - **adduser btaylor**
  - Select password of notsecure2
  - Verify password
  - Enter through defaults
  - Confirm info correct with y

- Change to btaylor user
  - **su btaylor**

```
root@KLY-IR105:~# su btaylor
btaylor@KLY-IR105:/root$
```

Hands On – Logging in with SSH Key/Pair

- Change to btaylor's home directory
  - **cd ~**
- Create a public/private key pair for btaylor
  - **ssh-keygen –t rsa**
  - Enter for defaults where files saved
  - Use cryptopass as passphrase
- Look at private/public key pair in ssh folder
  - **cd .ssh**
  - **ls**

Hands On – Logging in with SSH Key/Pair

- Copy public key to file called authorized_keys
  - **cp id_rsa.pub authorized_keys**
  - **ls** You should now see three files
- Change permissions on .ssh to 700 and authorized_keys to 600
  - **chmod 700 ~/.ssh**
  - **chmod 600 ~/.ssh/authorized_keys**
- Write down your eth0 IP address
  - Open new terminal as root and run **ifconfig**

Hands On – Logging in with SSH Key/Pair

- Now we need to get the private key onto the windows host so we can use it with Putty to logon.
- Any idea how to do that???
- Use WinSCP (Secure File Copy)
- Open WinSCP in Win 8
- Choose SCP as protocol
- Enter IP address you wrote down in “Host name” box

Hands On – Logging in with SSH Key/Pair

- Enter btaylor in “User name” box
- Enter notsecure2 in “Password” box
- Click “Login” button
- Chose “yes” if you get a key warning
- Left side is Win8, Right side is Kali
- On Left:
  - Double-click folder with .. to change up one level
  - Double-click Desktop folder

Hands On – Logging in with SSH Key/Pair

- On Right:
  - Double-click .ssh folder
  - Drag id_rsa from right side to bottom of left side and hit "OK"
    - You just copied the private key from the Kali .ssh folder to the Win8 Desktop
- Close WinSCP
- At this point we have one problem. Any idea??
- The private key is not in a format Putty can use.

Hands On – Logging in with SSH Key/Pair

- Open puttygen in Win8
- Click “Load” button and move dropdown on bottom right to “All Files.”
- Select “Desktop” on left side
- Double-click the id_rsa file
- Enter cryptopass as the passphrase and hit “OK”
- Click “OK” again
- Click “Save private key”
- Name the key kali and click “Save”

Hands On – Logging in with SSH Key/Pair

- Close puttygen and open putty
- Expand the “SSH” option on the bottom left
- Click on “Auth” and “Browse” to select the new converted private key we just created on the Desktop called kali.ppk and click “Open”
- Scroll the left bar to the top and click “Session”
- Type the kali IP address you wrote down in the “Host Name” box and click “Open”
- Hit “Yes” to the Alert

Hands On – Logging in with SSH Key/Pair

- Enter btaylor at the login as prompt
- Enter passphrase as cryptopass
- You are now logged in as btaylor via an encrypted SSH connection
- You would now normally need to make sure you put your private key in a secure non visible spot
  - Possibly an encrypted volume on Win 8.1
- Close Putty

Hands On – Logging in with SSH Key/Pair

- Some people who need to access remote systems multiple times in a day will not specify a passphrase when generating the key pair
- This is not secure

- If you don't generate a passphrase, anyone who finds your private key can use it to login to the remote system

Hands On – Logging in with SSH Key/Pair

- Any idea what we could use to avoid entering the passphrase but still be secure???
  - pageant

- Pageant lets you import your private key and only asks for your passphrase once. Then, as long as pageant is running, you won't have to type in your passphrase.

Hands On – Logging in with SSH Key/Pair

- Open pageant in Win8
- Click the up arrow on the right side of the bottom task bar in Win8 and you should see a computer with a black hat. Double-click on that
- Select "Add Key" and choose the kali.ppk file from your desktop
- Select "Open" and enter your passphrase as cryptopass and click "OK"
- Click "Close" on pageant

Hands On – Logging in with SSH Key/Pair

- Open putty again (This time we don't need to browse to the private key file)
- Type your kali IP address in the "Host Name" box again and click "Open"
- Login as btaylor
- This time you should have gotten right in without entering your key passphrase

Hands On – Logging in with SSH Key/Pair

- Exit out of putty
- Right click on pageant in the taskbar and select "Exit"
  - It is always good to exit pageant when you are done using the remote system for the day or will be away from your desk to a period of time

Text References

- Graff, J. (2001). Cryptography and E-Commerce. Ney York, NY: John Wiley & Sons, Inc.

Checkpoint

- Diffie-Hellman Key Exchange (Origins)
- Public & Private Key Pairs (Current Day)
  - Confidentiality
    - Which key encrypts and which decrypts?
  - Authentication
    - Which key encrypts and which decrypts?
- How to get integrity and sometimes authentication?
- How to get non-repudiation?

Checkpoint

- Which standards offer Confidentiality, Authentication, Integrity, and Non-Repudiation?
- What is SSH?
- What does pageant do?

Next Week

- No class next week due to Spring Break
- I will be in Orlando at a SANS Mobile Hacking Course

Homework

- Complete Homework9
  - Due before midnight on Sunday, March 27th
  - There are four questions
  - One of the questions is about attacking your service for your individual project
  - I am giving you almost three weeks to work on this assignment to:
    - Give you more time on your individual project
    - Be nice and not have something due at the end of spring break week

Midterm Answer Review

- Go over Midterm and suggestions