Cryptography Part 2

Public Key Distribution

After we have decided on using public and private keys, one issue remains – there needs to be a way to transfer the public key securely. A secure channel is needed.

If a public key is distributed insecurely, we may face a man-in-the-middle attack, where Mallory may intercept communications between Alice and Bob, intercepting one party's public key and providing his own public key to the other party instead. The attacker can then read all messages between Alice and Bob and modify them as required.

Only when a public key is securely distributed can we use it for encryption (confidentiality) and signature verification (authenticity). As such, there is seemingly no difference between symmetric-key setting and public-key setting, since there is still a need for a secure channel to send a key from Alice to Bob (secret key for symmetric-key setting, public key for public-key setting).

Nevertheless, the public-key setting is arguably easier to handle:

Requirement Aspect	Symmetric-Key Setting	Public-Key Setting
No. of times a secure channel is required	For every pair of entities: $n(n-1)/2$ $O(n^2)$	Each entity just needs to securely broadcast its public key: n $O(n)$
Item to be transmitted	Shared secret key	(Publicly-published) public key
Secure channel timing requirement (e.g. when a new entity needs to securely talk to another entity)	A secure channel is needed to deliver both parties' newly-set secret key	Previously announced public key(s) just need to be made accessible to a party requiring the key(s)

Three Different Methods for Public Key Distribution

1. Public Announcement

The owner of the public key can broadcast their public key. Many list their Pretty Good Privacy (PGP) public key on their websites. However, this method is not very standardized and there is no systematic way to find or verify the public key when needed.

Furthermore, there is a need to trust the entity distributing the public key. For example, we need to trust that the website holding someone's key can be trusted.

2. Publicly Available Directory

We can potentially list all names/email addresses and public keys in a public-key directory server. By querying the name or email address, we can access the public key.

However, it is not easy to have a "secure" public directory. How does the server verify that the information provided in a request to post a public key is correct? Eventually, some entity will need to be trusted.

There is also a possibility of the server that the user is accessing being a non-authentic one i.e. the user is actually visiting a spoofed server.

3. Public Key Infrastructure

Public Key Infrastructure (PKI) is a standardized system that distributes public keys. The objectives of PKI are:

- To make public-key cryptography deployable on a large scale
- To make public keys verifiable without requiring any two communicating parties to directly trust each other
- To manage public and private key pairs throughout their entire key lifecycle

There is thus a concept of trust that needs to be managed.

Trust
<Insert book info here>

Public Key Infrastructure is centered around two important components – Certificate and Certificate/Certification Authority (CA). PKI provides a mechanism for trust to be extended in a distributed manner, starting from the root CA. It is an arrangement that binds public keys with the respective identities of entities e.g. people and organizations.

Certificates and Trust

Certificate Authority

A certificate authority (CA) issues and signs digital certificates. The cryptography involved is that of digital signatures. It keeps a directory of public keys, and it also has its own public-private key pair. We assume that the CA's public key has been securely distributed to all entities involved.

Most operating systems and browsers have a few pre-loaded CA's public keys. They are known as the "root" CAs. There are stringent operational requirements for a CA – for example, it must pass the WebTrust audit.

Certificate

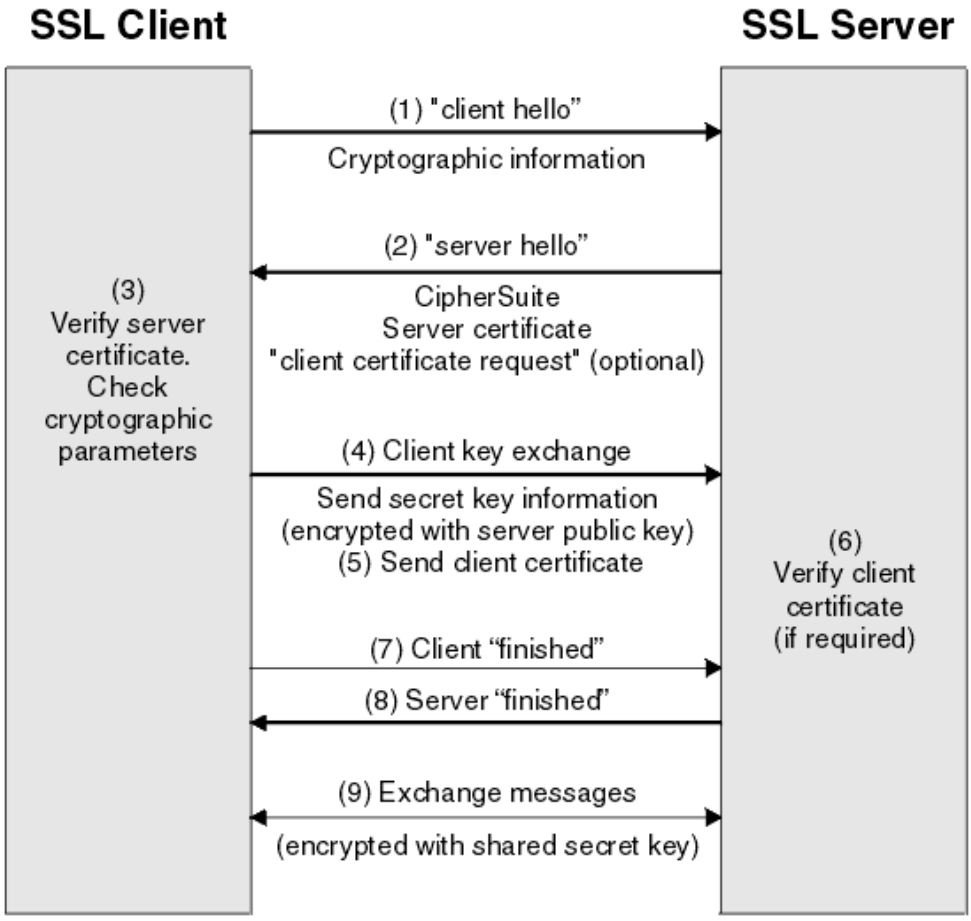
A certificate is a digital document that contains at least the following main items:

1. Identity of the owner
2. Public key of the owner
3. Valid time window of this certificate
4. Digital Signature of the Certificate Authority (entity that issued the certificate)

There is other additional information based on the intended purpose of the certificate.

A certificate is widely used by Internet applications – Secure Sockets Layer (SSL) / Transport Layer Security (TLS), Secure/Multipurpose Internet Mail Extensions (S/MIME), Secure Shell (SSH), etc.

This is how the SSL or TLS handshake looks like:



In general, these are the steps:
- ClientHello from client to server
- ServerHello from server to client
- Certificate from server to client
- ServerHelloDone from server to client

Advantages of Certificate-based PKI
In a certificate-based PKI, the CA acts as the directory server. Instead of constantly needing to query for the public key from the server, the CA issues a certificate to entities to bind their public keys with them. This overcomes two problems:
- The directory server is a bottleneck
- The verifier needs to have online access to the directory server at the verification point

Previously, if I were to send an email signed with my private key to someone, that someone would need to query the directory server to ask for my public key.

With certificates, I will send an email signed with my private key along with my certificate. The recipient would verify that the signature in the certificate is indeed signed by the CA, and since no one except the CA can produce the valid signature, the authenticity of the information in the certificate is as good as coming from the CA.

This certificate that I would send is obtained from the "offline" CA beforehand. There is still a need to check that the certificate has not been revoked, which is done with the Online CRL Distribution Point or OCSP Responder.

<u>X.509 Digital Certificate Standard</u>
The bodies that can decide the X.509 Standard are:
- International Telecommunications Union's Standardisation Sector (ITU-T)
    - Was here before the Internet era
    - Specifies formats for certificates, certificate revocation lists, and a certification path validation algorithm
- Public-Key Infrastructure Working Group (PKIX)
    - IETF working group that creates Internet standards on issues related PKI based on X.509 certificates

The structure of an X.509 v3 digital certificate is as such:
- Certificate
    - Version Number
    - Serial Number
    - Signature Algorithm ID (Note Signature Algorithm below too)
    - Issuer Name
    - Validity period
        - Not Before
        - Not After
    - Subject Name
    - Subject Public Key Info
        - Public Key Algorithm
        - Subject Public Key
    - Issuer Unique Identifier (optional)
    - Subject Unique Identifier (optional)
    - Extensions (optional)
- Certificate Signature Algorithm
- Certificate Signature

This certificate is bound to a particular Distinguished Name (DN). A DN has these common attribute types:

- Country (C)
- State (S)
- Locale (L)
- Organization name (O)
- Organizational unit name (OU)
- Common name (CN) - Common name can be an individual user or any other entity, e.g. a web server

How do I get a certificate?

Get a Root CA to issue you one:

- Paid ones: $10 - $50 / year (not costly)

“Let's Encrypt” provides (basic) TLS certs at no charge:

- Launched in April 2016
- A certificate is valid for 90 days
- Its renewal can take place at anytime
- Automated process of cert creation, validation, signing, installation, and renewal
- No of issued certs: 1M (March 2016) to 380M (Sept 2018)

Firefox Telemetry:

- 77% of all page loads via Firefox are now encrypted
- It is predicted that it will reach 90% by the end of 2019

Certificate Authority and Trust Relationship

The CA is also responsible of verifying that the information in a certificate is correct. For example, when issuing a certificate for a certain website domain, the CA should check that the applicant owns the domain name.

This may involve some manual checking and can be costly, especially for **Extended Validation SSL** (EV SSL) certificates. Those certificates are the highest class of certificates with a lot of stringent checks done, and they activate both the padlock and green address bar in major browsers.

Different SSL Certificates

1. Domain Validation (DV) SSL certificate
    a. Issued if the purchaser can demonstrate the right to administratively manage a domain name
    b. For example, replying to an email sent to the email contact in the domain's whois details, publishing a DNS TXT record
2. Organization Validation (OV) SSL certificate
    a. Issued if the purchaser additionally has an organization's actual existence as a legal entity
3. Extended Validation (EV) SSL certificate
    a. Issued if the purchaser can persuade the certificate provider of its legal identity
    b. This process involves manual verification checks by a human

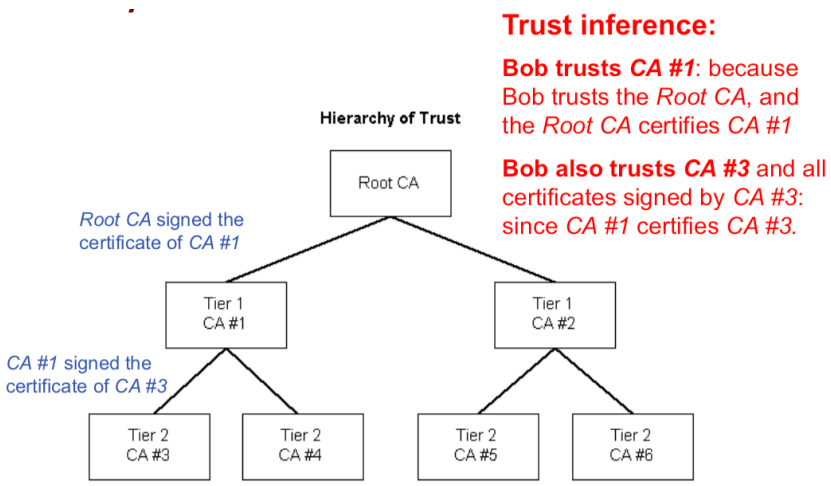
TLS Certificate Level Summaries

Certificate type	HTTPS encrypted?	Padlock displayed?	Domain validated?	Address validated?	Identity validation	Green address bar?
DV	Yes	Yes	Yes	No	None	No
OV	Yes	Yes	Yes	Yes	Good	No
EV	Yes	Yes	Yes	Yes	Strong	Yes

Types of Certificate Authority

There are three different types of CA:

- Root CA
    - Self-signed Certificate
- Subordinate / Intermediate CA
    - Tier 1, 2 etc.
- Leaf CA



There is a hierarchy of trust when it comes to the CAs. The Root CA signs the certificates of Tier 1 CAs, and the Tier 1 CAs sign the certificates of Tier 2 CAs. Suppose Alice's certificate is issued and signed by a Tier 1 CA, CA_1 and she sends a message, signed, with her certificate to Bob. But Bob does not have the public key of CA_1. What happens?

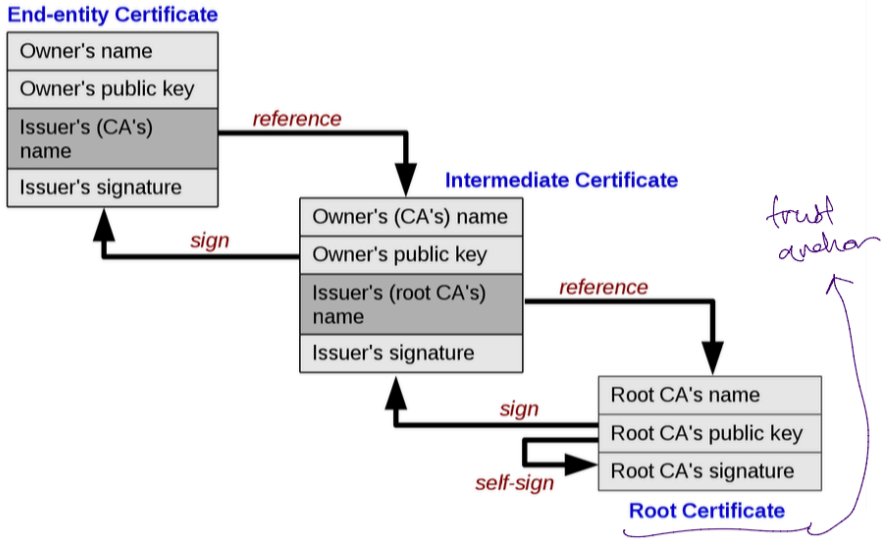
This is the **certification chain or path verification.** Alice will actually send CA_1's certificate along with her own (along with the Root CA's certificate as well). Bob will then verify CA_1's certificate using the Root CA's public key, Alice's certificate with the verified CA_1's public key, then verify Alice's message using Alice's verified public key.

Certification Chain
This is a list of certificates starting with an end-entity certificate followed by one or more CA certificates (with the last one being a self-signed root certificate).

For each certificate (except the last one):
- The issuer matches the subject of the next certificate in the list
- It is signed by the secret key of the next certificate in the list

The last certificate in the list, i.e. the root CA's, is the trust anchor**.**

There must also be a note in the certificate issued that states that the certificate owner is able to issue certificates to others, before non-root CAs are able to issue certificates.



Certificate Revocation
When certificates expire, they will naturally be revoked. But non-expired certificates can also be revoked for various reasons:
- Private key was compromised
- Issuing CA was compromised
- Entity left an organisation
- Business entity closed

Thus, a verifier needs to check if a certificate is still valid, even if it is not expired yet. There are different approaches to certificate revocation:
- Certificate Revocation List (CRL)
    - CA periodically signs and publishes a revocation list
- Online Certificate Status Protocol (OCSP)
    - OCSP Responder validates a certificate in question

An online CRL Distribution Point or OCSP Responder is thus needed.

However, there are issues with OCSP:

- Privacy
  - The OCSP Responder knows which certificates that you are validating
- Soft-fail Validation
  - Some browsers proceed in the event of no reply to an OCSP request (no reply *is* a “good” reply)

Solutions

- OCSP Stapling
  - Allows a certificate to be accompanied or “stapled” by a (time-stamped) OCSP response signed by CA
- Part of TLS handshake
  - Clients do not need to contact CA or OCSP Responder
- Drawback of the above
  - Increased network cost

Limitations and Attacks

There are numerous security issues when it comes to certificate authorities:

1. Compromise of CAs

As the CAs ultimately rely on the Root CA, if the Root CA is compromised, a lot of issues will occur.

Examples:

DigiNotar (Netherlands)

- Resulted in 500+ fraudulent certificates, including for *.google.com, *.mozilla.com, *.windowsupdate.com, *.torproject.org, in Sept 2011
- Immediately removed by major browsers
- Declared bankrupt within the same month

Turktrust (Turkey)

- Its subordinate CA, *.EGO.GOV.TR, issued *.gmail.com certificates
- Fraudulent certificates were used against Google Web properties

2. Abuse by CAs

There are so many CAs; some of them may be malicious. A rogue CA can practically forge any certificate. For example, Trustwave issued a "subordinate root certificate", which can then issue other certificates, one of which went to an organization that monitored the network. With this certificate, the organization can "spoof" X.509 certificates, and hence is able to act as the man-in-the-middle of any SSL/TLS connection.

3. MITM Attack by Rogue CA

A rogue CA can perform a man-in-the-middle attack by doing proxy re-encryption. These are the steps that will happen:

a) First, Mallory intercepts Alice's ClientHello to Bob.
b) Mallory pretends to be Bob. As a CA, Mallory is able to issue a certificate with Bob's details, and the digital signature is signed with Mallory's private key.
c) Alice will accept the information listed in the certificate issued by Mallory, because Alice has accepted Mallory's certificate.
d) Alice and Mallory will establish communication
e) Mallory then sends ClientHello to Bob. Bob will send over his certificate, which Mallory will accept. He will now have Bob's public key as well.
f) Mallory now can see all traffic and also modify all data.

4. Weak Browser Trust Model

Certification is as strong as the weakest root CA. The Browser Trust Model is a pre-loaded list of widely-used root CAs compiled by web browser developers. This is a form of Certificate Trust List (CTL) approach, where a list of CAs' certificates are compiled by a "trusted" authority.

The trust anchor in this situation would be the **union** of all root CAs. But there is always the question of which root CA should we use?

Implementation Bugs

There are also limitations in implementations that lead to severe vulnerability. These are not so much related with Certificates and CAs, but is related to web security. Below are some examples:

1. Null-byte Injection Attack

Some browsers ignore the substrings in the entity's identity/name field after null characters when displaying it in the address bar, but include them when verifying the certificate.

For example, www.comp.nus.edu.sg\0.hacker.com will be displayed as www.comp.nus.edu.sg.

2. Social Engineering

A social engineering attack is typosquatting. Basically, hackers will register for domain names that look visually similar to an actual URL e.g. luminus.nvs.edu.sg, get a valid certificate for that domain, then employ a phishing attack to trick a victim to click on the above link. The victim does not notice the incorrect URL and ends up giving their credentials to the site.

Strong Authentication

As mentioned before, the password is a weak authentication system as an eavesdropper can easily replay the transmission to get authenticated. There is another way for the user to prove their identity i.e. she knows the secret, without revealing the secret.

Secret-Key-Cryptography-Based (SKC-Based) Challenge-Response

Suppose Alice and Bob have a shared secret k, and they agree on an encryption scheme e.g. AES.

The following steps will happen:

1. Alice sends to Bob a hello message
    a. "Hi, I'm Alice"
2. (Challenge) Bob randomly picks a plaintext m.
    a. Bob computes $y = E_k(m)$
    b. and sends y to Alice
3. (Response) Alice decrypts y to get m and then sends m to Bob.
4. Bob verifies that the message received is indeed m. If not, any further communication is rejected.

Even if the eavesdropper Eve can obtain all communication between Alice and Bob, Eve still cannot get the secret key k. Replay is also impossible as the challenge m is randomly chosen and will likely be different in the next authentication session. The m ensures the freshness of the authentication process.

The protocol only authenticates Alice; it is called a unilateral authentication. There are also protocols to verify both parties, which are called mutual authentication.

Public-Key-Cryptography-Based (PKC-Based) Challenge-Response

Suppose Alice wishes to authenticate Bob.

The following steps will happen:

1. (Challenge) Alice chooses a random number r and sends to Bob
    a. "Bob, here is your challenge, r"
2. (Response) Bob uses his private key to sign r
    a. Bob also attaches his certificate sign(r)
3. Alice verifies Bob's certificate and extracts Bob's public key from the certificate before verifying that the signature is correct

An eavesdropper can't derive Bob's private key and replay the response because the challenge is likely to be different. The value r is also known as the cryptographic nonce (or simply nonce).

Session Keys

Strong authentication by itself is unable to ensure that Mallory cannot interrupt the session. This is because Mallory can still interrupt and take over the channel after strong authentication is carried out i.e. Mallory can pretend to be Alice.

We thus need something more. The outcome of the authentication process is a new secret key i.e. session key established between Alice and Bob. This process is called **key exchange** or **key agreement.**

Authenticated Key Exchange

If the process is incorporated with authentication, then this process is called authenticated key exchange or station-to-station protocol (if the key-agreement used is Diffie-Hellman). To carry out the protocol, Alice and Bob needs to know each other's public key, such as using PKI. After the protocol is completed, a set of session keys would be established: e.g. 1 for encryption and 1 for MAC, 1 key for each direction of encryption etc.