Public key cryptography

Pieter van den Hombergh

Fontys Hogeschool voor Techniek en Logistiek

June 6, 2016

Scheme use and implementation

- In 1976[1] Whitfield **Diffie** and Martin **Hellman** described how a digital signature scheme could be used. The idea of key exchange
- Ronald **Rivest**, Aid **Shamir** and Len **Aldeman** provided an implementaion for such a scheme in the RSA algortihm in 1977.

[1] quite recent compared to other, symmetric schemes

Simplified explanation of RSA

- The concept heavily relies on number theory, some of which goes back a long time, and modulo arithmetic with big numbers.
- The strength lies in the fact that factoring large numbers is a slow process.

The central model is

$$(m^e)^d \equiv m(\text{mod } n)$$

where e, d and n are very large[2] positive integer numbers. All computations, including the exponentiation, are done using modulo n arithmetic.
Note that also in this case, $(m^e)^d = (m^d)^e \equiv m(\text{mod } n)$, which shows the equivalence of the exponents d and e and is also essential in the approach.

[2] think hundreds of bits

Example use

- **A**lice creates key pair, of which (n, e) is the public key and (n, d) is the private key. She shares the public key with **B**ob.
- **B**ob encrypts the message M. He therefor turns M into a large integer m, such that $0 \leq m < m$ and $\gcd(m, n) = 1$ by using a padding scheme. From that he computes the ciphertext c where

$$c \equiv m^e \pmod{n}$$

  which he transmits to **A**lice.
- **A**lice uses exponent d from her private key (n,d) to compute

$$m \equiv c^d \pmod{n}$$

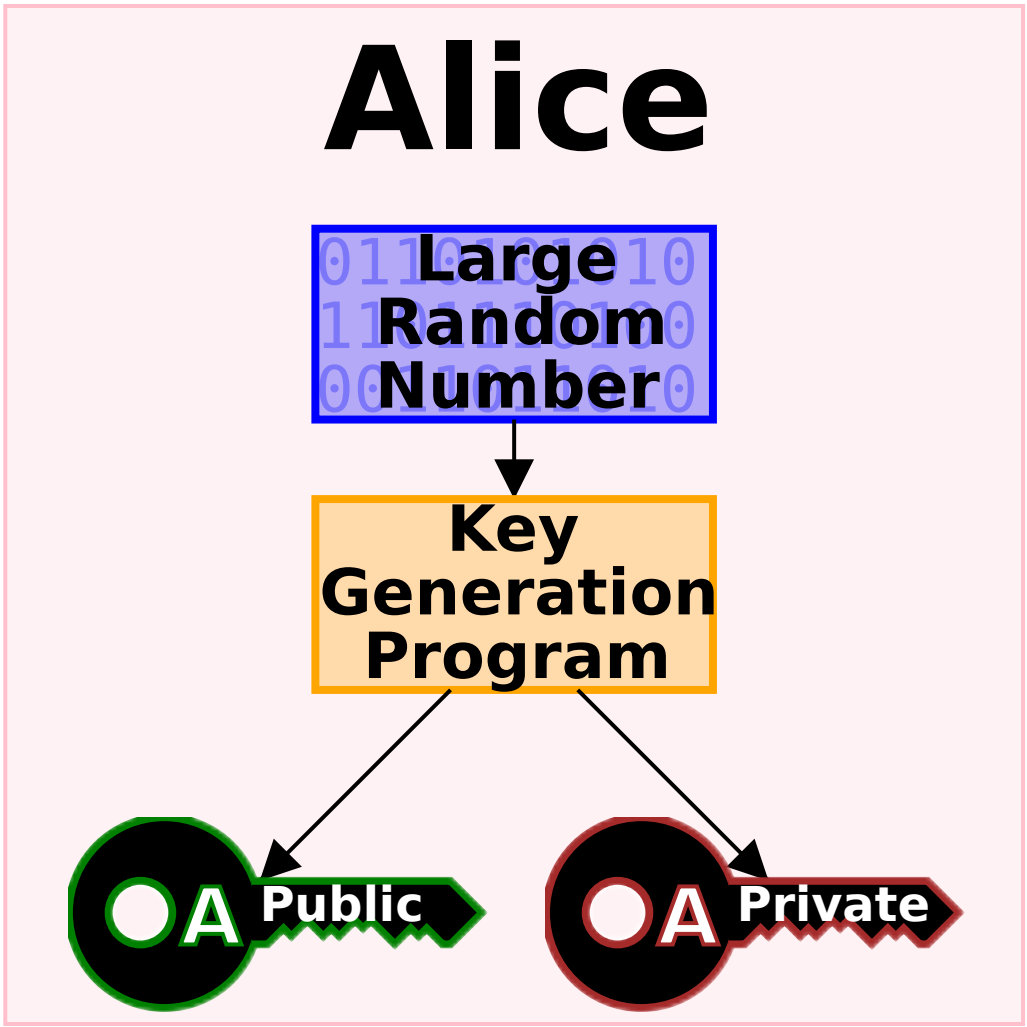
  producing the same m as **B**ob used. She than applies the reverse of the padding scheme, finding the original message.

Key generation

1. Choose two distinct large primes p and q. These should be chosen random with a true random generator. (As in: unpredictable).
2. Compute $n = p \times q$ This n is the modulus in both public and private key. Its length, in bits, is the key length.
3. Compute the Eulers totient function $\varphi(n)$, which for the primes p and q is simply $n - (p + q - 1)$. This is kept private.
4. Choose e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$, saying e and $\varphi(n)$ are **coprime**.
5. Determine d from $d \equiv e{-}1 (\mathrm{mod}\ \varphi(n))$

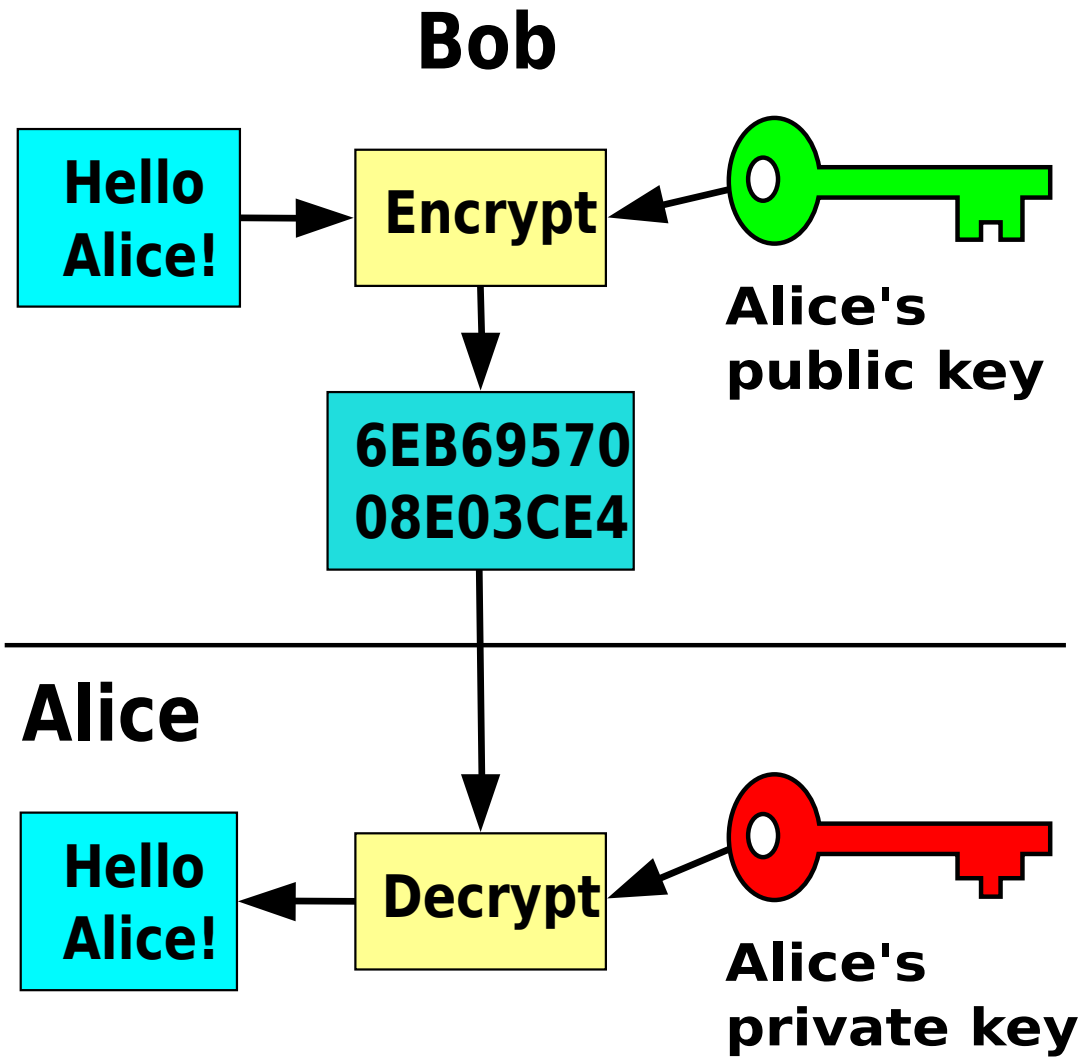
The e and n are combined into the public key, the d and n make up the private key.
source: RSA

Public-key cryptography



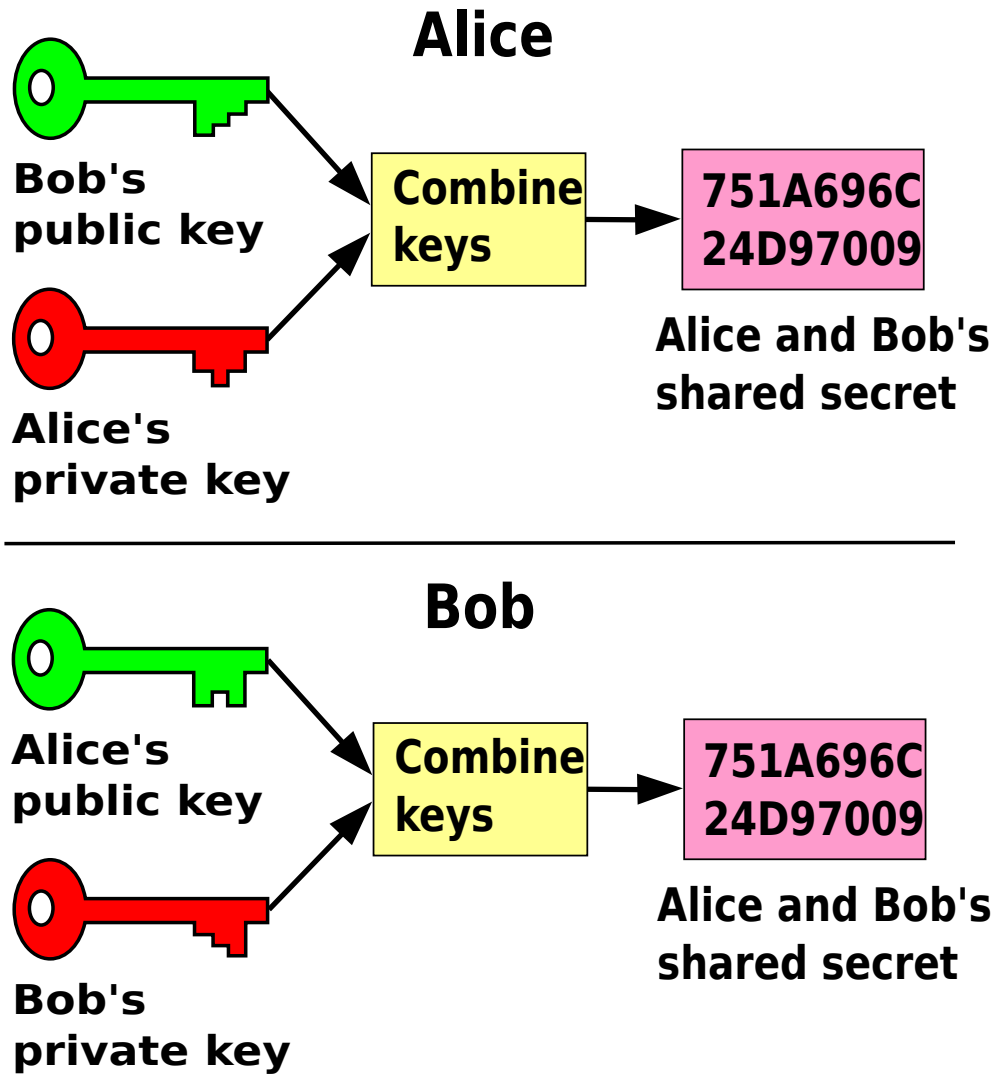
The key pair, simplified.

Encrypt-decrypt



The use of the keys

Shared secret



A shared secret can be use as key for a symmetric encryption. (Diffy-Hellman)

Signing

- To sign a message, a signature can be added, by encrypting with the **public** key. (Signed and sent by Alice to Bob).

$$X = encrypt(\text{pubkey}_{\text{Bob}}, encrypt(\text{privkey}_{\text{Alice}}, M))$$

  where M is the message and X the result of encrypting, then signing it.
- The encrypted X is sent along an insecure path.
- Decryption *and* verification is done with the reverse use of *opposite* keys.

$$M = decrypt(\text{pubkey}_{\text{Alice}}, decrypt(\text{privkey}_{\text{Bob}}, X))$$

  Note that the intermediate product is still encrypted (with Alice's pubkey).
- Remember the evaluation order of the function calls!

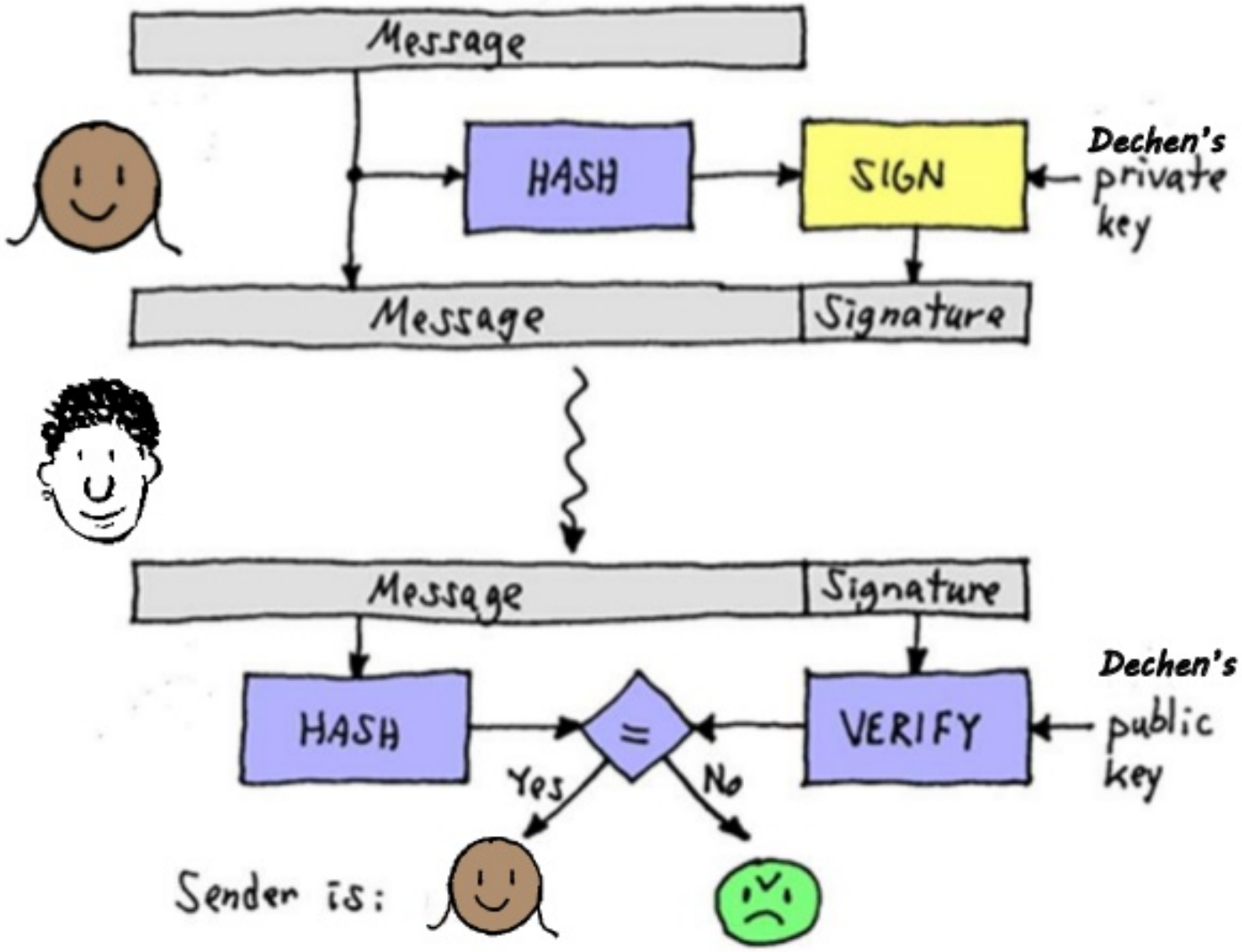
Signing with public key technology

- We have seen that encryption uses the public key of the recipient to encrypt, and the private key of same to decrypt.
- The Owner of the key has the private key and is (or should be the only one that can decrypt.

In signing we do the (almost) reverse:

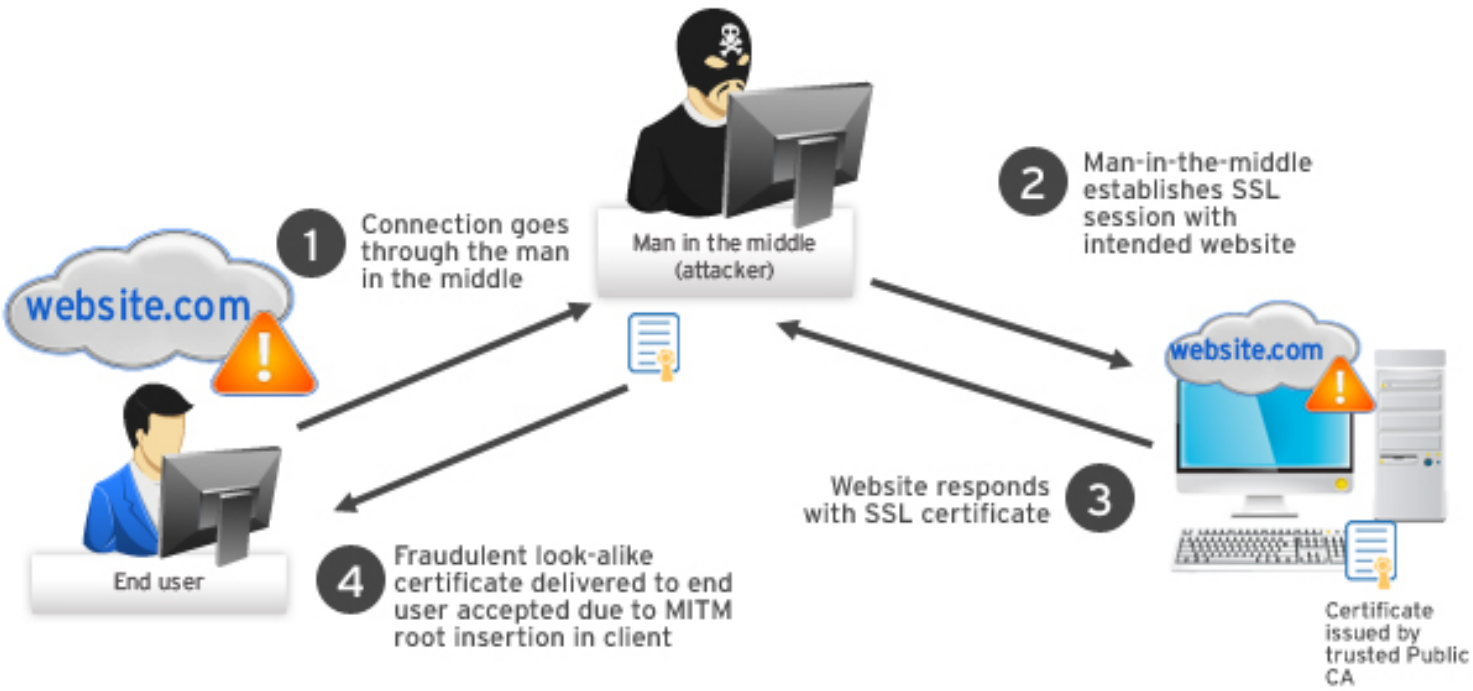
- From the message to be signed, we compute a hash h.
- The sender encrypts this hash with his private key, appends it to the message.
- some steps left out... (quiz which one?)
- The receiver decrypts the crypted hash, and if successfull, *knows* that the sender must have encrypted it, because he is the only one with access to the private key.

Signing in a picture



In place of the wiggly vertical, we could add encryption to keep the information confidential. Quiz: What will the

Man in the Middle attack



If the MITM can make the end user trust HIS certificate, he can decrypt the communication from the end user to him, and because he communicates to the server "on behalf" of the end user, also the traffic from the server intended for the end user. He can also modify the information.

Authenticity

A major concern is authenticity.

- Being sure that the party we are talking to is the one he claims to be.
- Successful authentication is often a prerequisite of authorization.
- This works both ways:
  - A computer needs to verify the authenticity of a user to allow certain operations.
  - A user needs assurance to be certain she is connected to the proper service, not a fake one.

The problem is aggravated when the two parties do not personally know each other and have to trust the opposite anyway.

A helping hand

- In particular the problem of the user trusting the system is problematic, since the user cannot "see" the server, only what it communicates.
- To address this, we need a helper we can trust: The Certification Authority. (**CA**).
- The CA verifies that a party is who he claims to be (sometimes even by making them or a representative show up in person), than
- *signs* a Certificate Request (CSR), creating a (server) certificate.
- This CSR is based upon the public key of the requestor (sometimes called the supplicant).

Client certification, example ssh

An even stronger fortification can be achieved by also make the client (end user) present a certificate.

- This is the road that **ssh** (Secure Shell) can taken when configured to use a client or user certificate. Often this certificate is created by the end user and safely transferred to the server in a end user specific store. (Typically `$HOME/.ssh` under Unixoids).
- In this use case the end user is known to the server AND the server can verify the authentication on use AND when the best practice of setting a non empty **key phrase** on the private key part, the security level is quite good.
- Currently ssh is the tool of choice to do remote work or deployment on a server.
- ssh can also provide a "poor mans" VPN.

Questions to ask

When considering network security you might want to ask the following questions:

- How about data safety (eavesdropping)
- How about identity theft
- Data integrity
- Authenticity
- Privacy

Sometimes the answers to these question are at odds with each other.

The problem of trust

- A **public key certificate** is used to prove ownership of a public key. The certificate includes:
  - information about the key
  - information about the owner's identity
  - a digital signature of a trusted entity
  - This entity is the key to trust and must be trusted by all parties involved

- You need a trusted channel to transport the certificate. Why?
- To create a trusted channel you need a certificate. Why?

source:
`http://en.wikipedia.org/wiki/Certificate_authority`

A social solution: Web of trust

- Trust is in fact a human emotion.
- An emotion but nonetheless a crucial one to make “Business” possible.
- A web of trust is based on: **I trust whom you trust.**
- What implications do we have here?

- This is what PGP (and GNU PG) do: people who trust one another sign each others key, thereby casting this trust in a certificate.
- The certificate contains personal details (name according to e.g. passport AND drivers license, DOB etc) and the public key.
- A web of trust has **NO** central authority.

Digitally distributing trust X.509

- X.509 is the ITU-T standard for a **public key infrastructure** (**PKI**)
- In X.509 a **certification authority** issues certificates to bind a public key to a **distinguished name**[3] or alternate name (e.g. email or dns entry).
- A CA can create/have sub ordinate or intermediate (level 2, level 3) registration authorities (RA), which can sign and issue end user (server) certificates.
- This builds a hierarchy of trust, with the CA a the top, the server at the bottom and the intermediate, if any, in between.
- Typically the root CA certificates are installed with the operating system and/or the browser. Why?
- Can you think of some weak spots in this whole circus?

source: `http://en.wikipedia.org/wiki/X.509`

[3] See LDAP, ADS

Protocols ans standards using X.509

- TLS/SSL
- S/MIME (Secure Multipurpose Internet Mail Extensions)
- IPsec
- SSH
- Smart card
- HTTPS
- EAP
- LDAP
- Trusted Computing Group (TNC TPM NGSCB)
- CableLabs (North American Cable Industry Technology Forum)
- WS-Security
- XMPP
- Microsoft Authenticode
- OPC UA

X.509 Certificate structure

- Certificate
  - Version
  - Serial Number
  - Algorithm ID
  - Issuer
  - Validity
    - Not Before
    - Not After
  - Subject
  - 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

Everlasting trust?

- As can be inferred from the structure, a certificate has a validity time span or period. Can you come up with some reasons?

Reasons for revocation: (from CRL)

- unspecified (0)
- keyCompromise (1)
- CACompromise (2)
- affiliationChanged (3)
- superseded (4)
- cessationOfOperation (5)
- certificateHold (6)
- removeFromCRL (8)
- privilegeWithdrawn (9)
- AACompromise (10)

- It must be possible to revoke a certificate. Possible is:
  - Using a certificate revocation list.
  - Use the Online Certificate Status Protocol (OCSP)
  - OCSP is favored by some browsers.

Principles of a CRL

- List contains the certificate serial numbers, not the whole certificates
- A list per CA.
- The list has a limited life time, e.g. it support caching, but not indefinite.
- *Usually* the CRL is signed by the CA.
- To validate the signature you need the CA's certificate, which is typically pre-installed. (App, browser, OS service).
- Security issues: What if the revocation list is not available (think of a way to prevent access).

Online (and dynamic) certificate status check

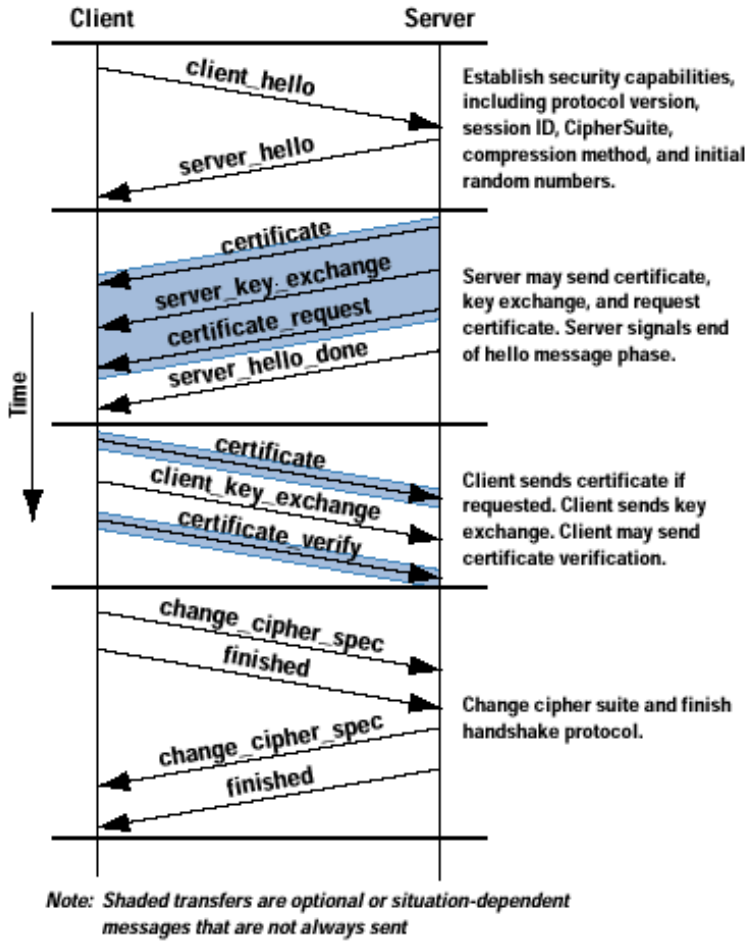
- The **Online Certificate Status Protocol, (OCSP)** is the internet protocol to obtain the status of a certificate (validity, revocation).
- OCSP also has some issues
  - Privacy: The CA can know that Bob and Alice have contact. This might compromise the privacy of one or both.
  - When a OCSP request times out, it is typically further ignored...
  - It does not protect against a *man in the middle* attack. (Think what this MitM can do with the net traffic).
    - The **MustStaple** TLS extension help mitigate this last two problem.
    - In a stapling scenario, the certificate holder queries the OCSP server themselves at regular intervals, obtaining a **signed, time-stamped** OCSP response.
    - The client (browser) can request a **Certificate Status Request** , and does not need to contact the CA.

source: `http://en.wikipedia.org/wiki/OCSP_stapling`

Secured HTTP connections (https://)

- The purpose of the protocol is twofold:
  - Authentication (of least of the server).
  - Secrecy (encryption).
  - self signed certificates only support the last.
  - Note that client authentication is also possible, but less used
- SSL and TLS use two phases to set up a secure connection.
- The data connection (payload) is encrypted using a symmetric key.
- Such key is created anew for each session.
- This is done in the key exchange phase. The server and client use public key encryption to set derive a *session* key.

SSL/TLS Key exchange



After this key exchange, the actual communication (HTTP payload) can start. source: cisco
`http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_1-1/ssl.html`

Open connection

A side effect of the expense of setting up a connection is that with SSL/TLS the TCP connection typically is kept open, where in HTTP (without s), a new connection is established for each request/response pair.
You can see this effect if you reconfigure your Apache server and then reload. The SSL connected child processes stay alive.