FACULTEIT INGENIEURSWETENSCHAPPEN

Network and Computer Security

Chapter 3 – Network and communication security

Prof. dr. ir. Eli De Poorter

Overview

- Network model
- Secure configuration of devices
  - SSH (Secure Shell)
- Exchanging keys
  - Out of band
  - Diffie-Hellman
  - Asymmetric encryption
  - Trusted third party
    - Key Distribution Centre (KDC)
    - Public Key Infrastructure (PKI)
- Secure networking protocols
  - Transport layer: TLS & SSL
  - Network layer: IPSec & VPN
  - Data link layer: WEB & WPA
- Firewalls
  - Packet filter
  - Circuit-level gateway
  - Application-level gateway (proxy)

Overview

- Network model
- Secure configuration of devices
- **Exchanging keys**
  - Out of band
  - Diffie-Hellman
  - Asymmetric encryption
  - Trusted third party
    - Key Distribution Centre (KDC)
    - Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

Key exchange

- **Symmetric encryption is much more efficient**
  - **But how to distribute the shared key securely?**
- **Goal**
  - **Agree on a key that two parties can use for a symmetric encryption, in such a way that an eavesdropper cannot obtain the key**
- **Methods**
  - **Out of band**
  - **Using Diffie-Hellman**
  - **Using asymmetric encryption**
  - **Using trusted third party**
    - **Public-key infrastructure (PKI)**
    - **Kerberos**
    - **Etc.**

4

Given two parties A and B, multiple **key distribution** alternatives exist

1. A can select key and physically deliver to B
2. third party can select & deliver key to A & B
3. if A & B have communicated previously can use previous key to encrypt a new key
4. if A & B have secure communications with a third party C, C can relay key between A & B

Physical delivery (1 & 2) is simplest - but only applicable when there is personal contact between recipient and key issuer. This is fine for link encryption where devices & keys occur in pairs, but does not scale as number of parties who wish to communicate grows. 3 is mostly based on 1 or 2 occurring first.

A third party, whom all parties trust, can be used as a **trusted intermediary** to mediate the establishment of secure communications between them (4). The clients must trust the intermediary not to abuse the knowledge of all session keys. As the number of parties grows, variants of 4 are the only practical solution to the huge growth in number of keys potentially needed.

Overview

- Network model
- Secure configuration of devices
- Exchanging keys
  - **Out of band**
  - Diffie-Hellman
  - Asymmetric encryption
  - Trusted third party
    - Key Distribution Centre (KDC)
    - Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

Out of band key exchange

- Exchanging public keys “out-of-band”
  - E.g. non-electronically
  - Signing public keys of other entities
  - Accepting public keys that were digitally signed by sufficiently trusted entities
- OK for acquaintances
- Less suitable for large organisations

This is more or less the basic principle behind the system used in PGP (Pretty Good Privacy) (see later).

Overview

- Network model
- Secure configuration of devices
- **Exchanging keys**
  - Out of band
  - **Diffie-Hellman**
  - Asymmetric encryption
  - Trusted third party
    - Key Distribution Centre (KDC)
    - Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

Key exchange: Diffie-Hellman

- **g = primitive root mod p**
  - **Example: the number 3 is a primitive root modulo 7**
  - **g = generator of the multiplicative group of integers modulo n**

$$
\begin{aligned}
3^1 &= 3 = 3^0 \times 3 \equiv 1 \times 3 = 3 \equiv 3 \pmod 7 \\
3^2 &= 9 = 3^1 \times 3 \equiv 3 \times 3 = 9 \equiv 2 \pmod 7 \\
3^3 &= 27 = 3^2 \times 3 \equiv 2 \times 3 = 6 \equiv 6 \pmod 7 \\
3^4 &= 81 = 3^3 \times 3 \equiv 6 \times 3 = 18 \equiv 4 \pmod 7 \\
3^5 &= 243 = 3^4 \times 3 \equiv 4 \times 3 = 12 \equiv 5 \pmod 7 \\
3^6 &= 729 = 3^5 \times 3 \equiv 5 \times 3 = 15 \equiv 1 \pmod 7
\end{aligned}
$$

8

In modular arithmetic a number g is a primitive root modulo n if for every integer a coprime to n, there is an integer k such that $g^k \equiv a \pmod n$. In other words, g is a generator of the multiplicative group of integers modulo n.

Key exchange: Diffie-Hellman

- **How can two parties agree on a secret value when all of their messages might be overheard by an eavesdropper?**
  - **The Diffie-Hellman algorithm accomplishes this, and is still widely used.**
  - **First practical method for establishing a shared secret over an unsecured communication channel (1976)**
- **Concept**
  - **Based on the discrete logarithm problem**
    - No efficient general method for computing discrete logarithms on conventional computers is known
    - Exploits the fact that $((g^b \bmod p)^a \bmod p) = ((g^a \bmod p)^b \bmod p)$
      - ✓ if p = prime number and g = primitive root mod p
  - Remember: also the factoring integer problem is challenging and used for the generation of public keys

9

The Diffie-Hellman key agreement protocol (1976) was the first practical method for establishing a shared secret over an unsecured communication channel. It is based on discrete logarithms. Discrete logarithms are fundamental to a number of public-key algorithms, including Diffie-Hellman key exchange and the digital signature algorithm (DSA). Discrete logs (or indices) share the properties of normal logarithms, and are quite useful. The logarithm of a number is defined to be the power to which some positive base (except 1) must be raised in order to equal that number. If working with modulo arithmetic, and the base is a primitive root, then an integral discrete logarithm exists for any residue. However whilst exponentiation is relatively easy, finding discrete logs is not, in fact is as hard as factoring a number. This is an example of a problem that is "easy" one way (raising a number to a power), but "hard" the other (finding what power a number is raised to giving the desired answer). Problems with this type of asymmetry are very rare, but are of critical usefulness in modern cryptography.

While computing discrete logarithms and factoring integers are distinct problems, they share some properties:

• both problems are difficult (no efficient algorithms are known for non-quantum computers),

• for both problems efficient algorithms on quantum computers are known,

• algorithms from one problem are often adapted to the other, and

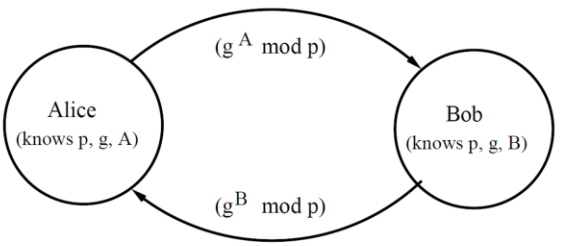
• the difficulty of both problems has been used to construct various cryptographic systems.

Diffie-Hellman Setup

- **Both users agree on global parameters:**
  - **large prime integer or polynomial p**
  - **g being a primitive root mod p**
- **each user (eg. A) generates their key**
  - **chooses a secret key (number): $a < p$**
  - **compute their public key: $y_A = g^a \bmod p$**
- **each user makes public that key y_A**

In the Diffie-Hellman key exchange algorithm, there are two publicly known numbers: a prime number p and an integer “a” that is a primitive root of p. The prime p and primitive root a can be common to all using some instance of the D-H scheme. Note that the primitive root a is a number whose powers successively generate all the elements mod p. Users Alice and Bob choose random secrets x's, and then "protect" them using exponentiation to create their public y's. For an attacker monitoring the exchange of the y's to recover either of the x's, they'd need to solve the discrete logarithm problem, which is hard.

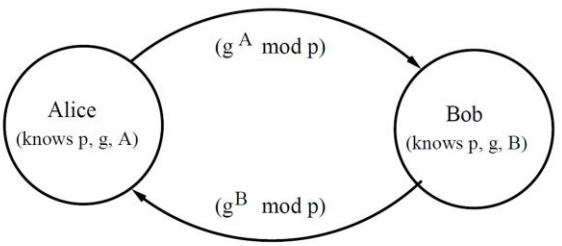
Key exchange: Diffie-Hellman



- **Steps in the algorithm:**
  1. **Alice and Bob agree on a prime number p and a base g**
     - **These do not need to be kept secret**
  2. **Alice chooses a secret number a, and sends $y_A = g^a \bmod p$.**
  3. **Bob chooses a secret number b, and sends $y_B = g^b \bmod p$.**
  4. **Alice computes $(y_B{}^a \bmod p)$.**
  5. **Bob computes $(y_A{}^b \bmod p)$.**

Both Alice and Bob can use this number as their key.
Notice that p and g need not be protected.

Key exchange: Diffie-Hellman



- **Example**
  - **Alice and Bob agree on p = 23 and g = 5.**
  - **Alice chooses a = 6**
  - **Bob chooses b = 15.**

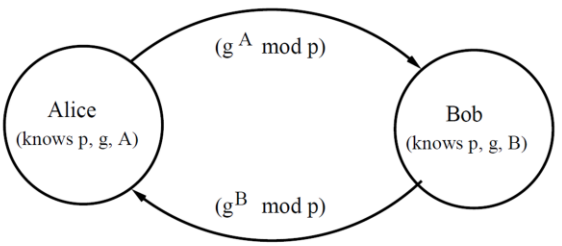
  What are the public keys?
  What is the shared key?

  Clearly, much larger values of a, b, and p are required.
  An eavesdropper cannot discover this value even if she knows p and g and can obtain each of the messages.

12

Alice chooses a = 6 and sends $5^6 \bmod 23 = 8$.
Bob chooses b = 15 and sends $5^{15} \bmod 23 = 19$.
Alice computes $19^6 \bmod 23 = 2$.
Bob computes $8^{15} \bmod 23 = 2$.

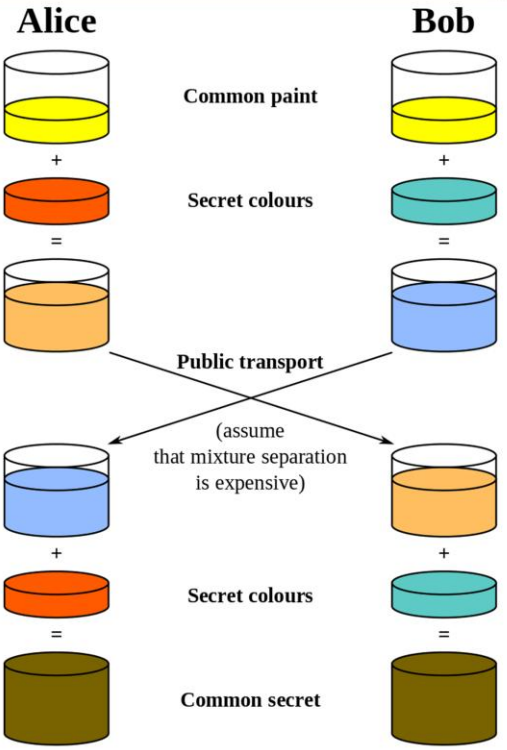
Key exchange: Diffie-Hellman



- Complexity example
  - p is a prime of around 300 digits
  - a and b at least 100 digits each.
- Discovering the shared secret given g, p, $g^a \bmod p$ and $g^b \bmod p$ would take longer than the lifetime of the universe, using the best known algorithm.
  - This is called the discrete logarithm problem

Key exchange: Diffie-Hellman

- Discrete logarithm problem: illustration



Issues with basic DH schemes

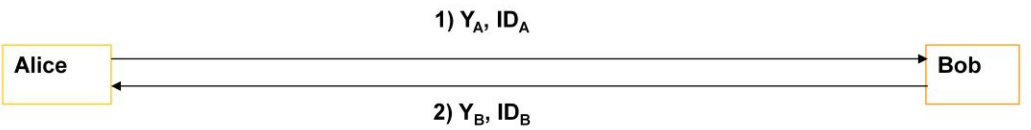
- **Denial-of-Service attacks**
  - **When basic scheme is used**
    - **After receiving attacker's public key the victim will…**
      - ✓ …compute the public key and send it to the attacker
        - » Who may have given a false address to that purpose
      - ✓ …compute the session key (in each configuration)
  - **Computation intensive operations**
    - **May be (ab)used to paralyse a server**
    - **Prior (partial) authentication needed to avoid needless heavy computations**

15

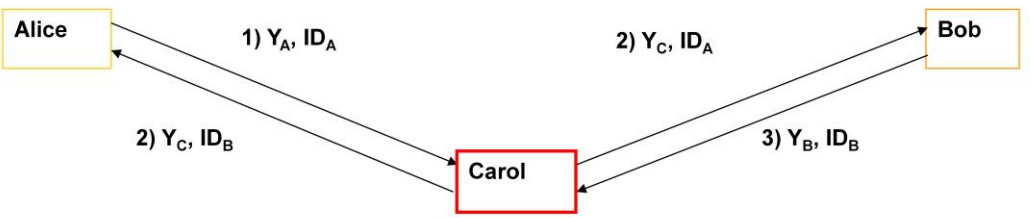
This type of attack is less severe against targets that use *fixed Diffie Helman* (see later), a scheme in which no new keys need to be computed for each session. To prevent this attack from happening, *ephemeral Diffie Helman* (see later) includes a separate prior authentication mechanism that is less computationally intensive.

Issues with basic DH schemes

- "Man-in-the-Middle" attacks
  - Normal operation



  - Attack scheme



16

After the attack Alice will be communicating with Carol using a secret key derived from Y_A and Y_C, and Bob will be communicating with Carol using a secret key derived from Y_B and Y_C, while Alice and Bob falsely assume they're communicating with each other using a mutually agreed upon secret key. Carol will intercept the encrypted traffic between Alice and Bob, decrypt it, and re-encrypt it, without Alice or Bob noticing.

This attack is only possible absent any authentication of the public keys (Y_A, Y_B, and Y_C). This typically is an issue with anonymous or ephemeral DH when no additional authentication mechanism is used. When fixed DH is used it is possible to link the (fixed) key pairs to the communicating entities (see later certificates). Authenticated DH also averts this attack, unless the (fixed) shared secret key is compromised.

Key exchange: Diffie-Hellman

- **Many possible variants**
  - **Diffie-Hellman using ECC**
    - **Better security**
  - **Fixed DH**
    - **Each entity has a fixed private (X_A and X_B) and public key (Y_A and Y_B)**
    - **Public parameters are signed by certification authority (CA)**
    - **Fixed secret key for each pair of entities**
      - ✓ OK if usage of secret key is limited
  - **Anonymous DH**
    - **No built-in authentication mechanism**
      - ✓ No certainty about who owns public key
      - ✓ Useful when 1 entity has no key pair
    - **Purely for exchanging (session) key**
      - ✓ Simple but vulnerable
  - **Ephemeral DH**
    - **Private keys generated for each session**
    - **Different secret session key for each session**
    - **Authentication using different mechanism**
      - ✓ RSA, DSA, etc.
    - **Perfect Forward Secrecy**

17

Elliptic curve Diffie–Hellman (ECDH) is a variant of the Diffie–Hellman protocol using elliptic curve cryptography.

Fixed Diffie-Hellman embeds the server's public parameter in the certificate, and the CA then signs the certificate. That is, the certificate contains the Diffie-Hellman public-key parameters, and those parameters never change.

Anonymous Diffie-Hellman uses Diffie-Hellman, but without authentication. Because the keys used in the exchange are not authenticated, the protocol is susceptible to Man-in-the-Middle attacks.

Ephemeral Diffie-Hellman uses temporary, public keys. Each instance or run of the protocol uses a different public key. The authenticity of the server's temporary key can be verified by checking the signature on the key. Because the public keys are temporary, a compromise of the server's long term signing key does not jeopardize the privacy of past sessions. This is known as Perfect Forward Secrecy (PFS).

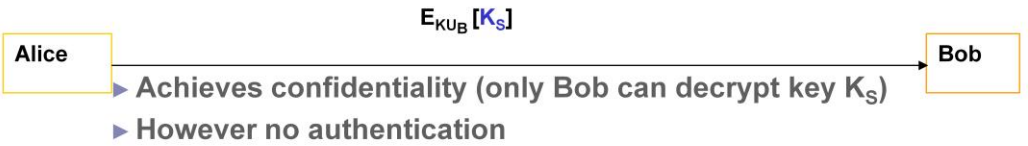
- **Challenge remains:**
  - **To avoid man in the middle attacks, authentication is used.**
  - **But how to do this securely?**

Overview

- Network model
- Secure configuration of devices
- **Exchanging keys**
  - Out of band
  - Diffie-Hellman
  - **Asymmetric encryption**
  - Trusted third party
    - Key Distribution Centre (KDC)
    - Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

Key exchange: asymmetrical approach

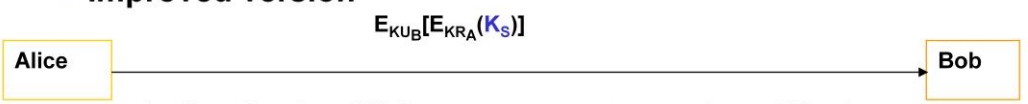
- **Using asymmetric encryption**
  - **Basic scenario**



▸ Achieves confidentiality (only Bob can decrypt key K_S)

▸ However no authentication

  - **Improved version**



▸ Authentication OK (message must come from Alice)

✓ However still vulnerable (how?)

p. 20

An extremely simple scheme was put forward by Merkle in 1979.

A generates a new temporary public key pair

A sends B the public key and their identity

B generates a session key K sends it to A encrypted using the supplied public key

A decrypts the session key and both use the session key

However, this approach is insecure against an adversary who can intercept messages and then either relay the intercepted message or substitute another message. The opponent can thereby impersonate both halves of protocol (i.e. a man-in-the-middle attack).

Key exchange: asymmetrical approach

- **Distribution of public keys can be considered using one of:**
  - **public announcement**
  - **publicly available directory**
  - **public-key authority**
  - **public-key certificates**

Several techniques have been proposed for the distribution of public keys, which can mostly be grouped into the categories shown.

Each of them has several advantages and disadvantages in terms of privacy, scalability, complexity, etc.

Key exchange: asymmetrical approach

- **Public Announcement**
  - **users distribute public keys to recipients or broadcast to community at large**
    - eg. append PGP keys to email messages or post to news groups or email list
  - **major weakness is forgery**
    - anyone can create a key claiming to be someone else and broadcast it
    - until forgery is discovered can masquerade as claimed user

The point of public-key encryption is that the public key is public, hence any participant can send his or her public key to any other participant, or broadcast the key to the community at large. Its major weakness is forgery, anyone can create a key claiming to be someone else and broadcast it, and until the forgery is discovered they can masquerade as the claimed user.

Key exchange: asymmetrical approach

- **Publicly Available Directory**
  - **registering keys with a managed public directory**
  - **directory must be trusted with properties:**
    - **contains {name,public-key} entries**
    - **participants register securely with directory**
    - **participants can replace key at any time**
    - **directory is periodically published**
    - **directory can be accessed electronically**
  - **still vulnerable to tampering or forgery**

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization. This scheme is clearly more secure than individual public announcements but still has vulnerabilities to tampering or forgery.

Overview

- Network model
- Secure configuration of devices
- Exchanging keys
  - Out of band
  - Diffie-Hellman
  - Asymmetric encryption
  - **Trusted third party**
    - **Key Distribution Centre (KDC)**
    - Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

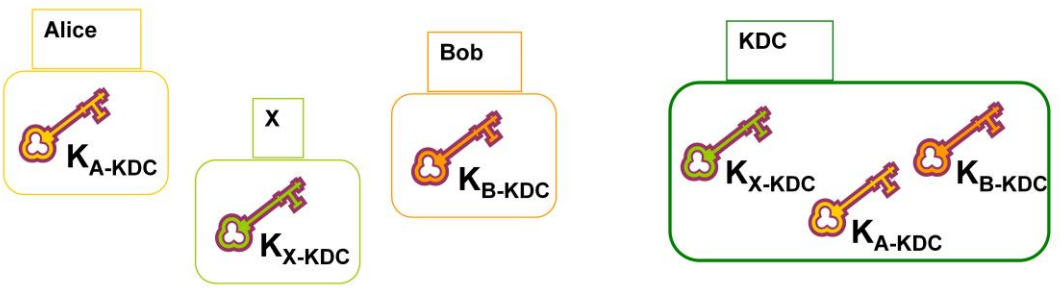
Key exchange: KDC

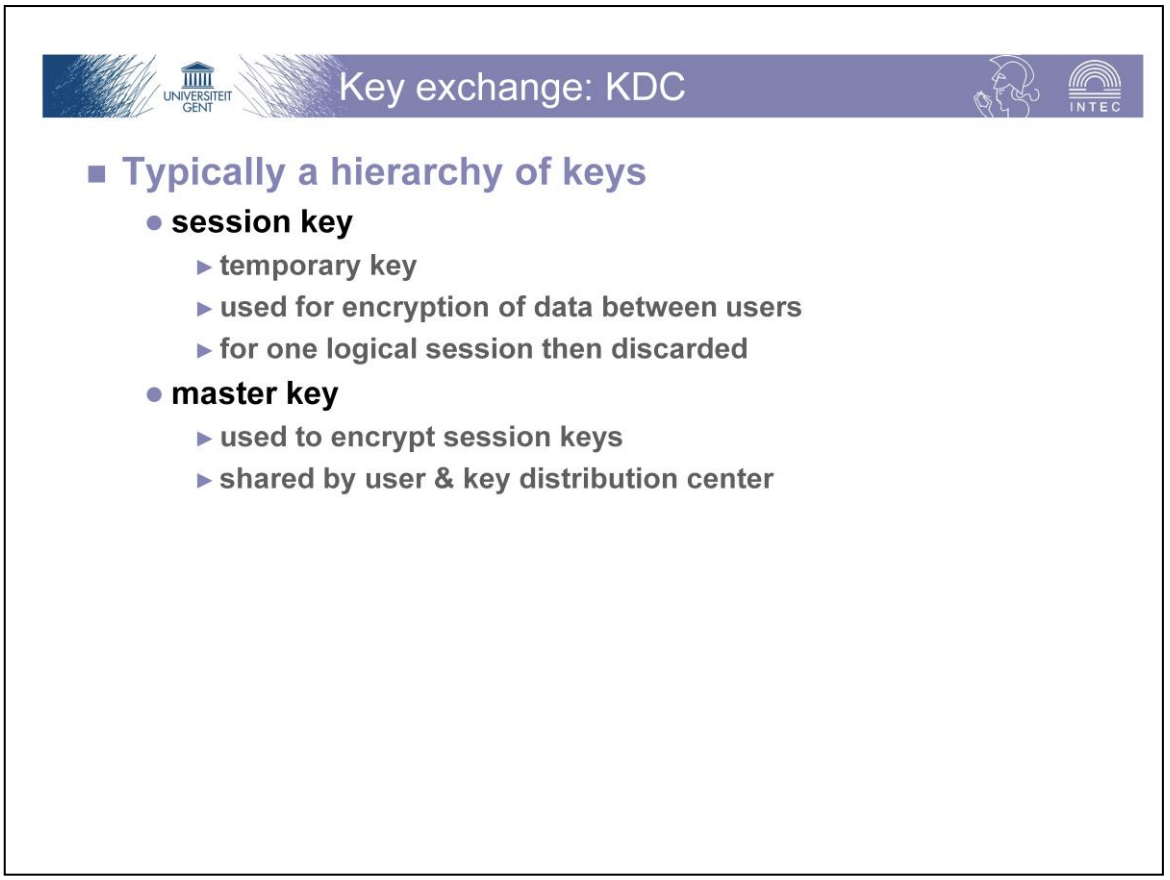
- **Key Distribution Centre (KDC) or Public-Key Authority**
  - **improve security by tightening control over distribution of keys from directory**
  - **has properties of directory**
  - **and requires users to know shared key for the directory**
  - **then users interact with directory to obtain any desired key securely**
    - **does require real-time access to directory when keys are needed**

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of keys from the directory.

Key exchange: KDC

- Each user owns secret key allowing him to communicate with key distribution centre (KDC)

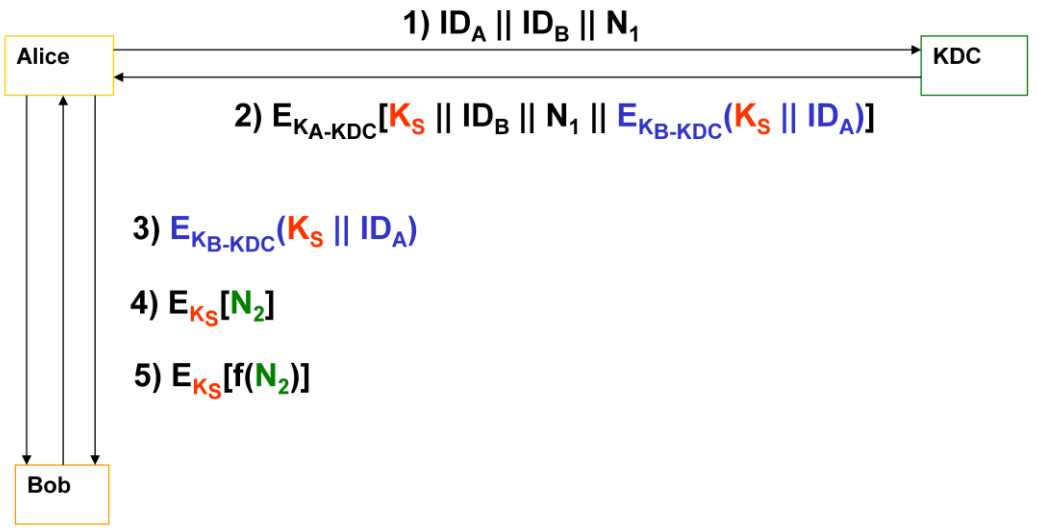




The use of a key distribution center is based on the use of a hierarchy of keys. At a minimum, two levels of keys are used: a session key, used for the duration of a logical connection; and a master key shared by the key distribution center and an end system or user and used to encrypt the session key.

For communication among entities within the same local domain, the local KDC is responsible for key distribution. To balance security & effort, a new session key should be used for each new connection-oriented session. A new session key is used for a certain fixed period only or for a certain number of transactions. An automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other, provided they trust the system to act on their behalf. The use of a key distribution center imposes the requirement that the KDC be trusted and be protected from subversion.

Key exchange: KDC



p. 28

N_1: nonce, which is used only once.

ID_A, ID_B: identities of Alice, resp. Bob

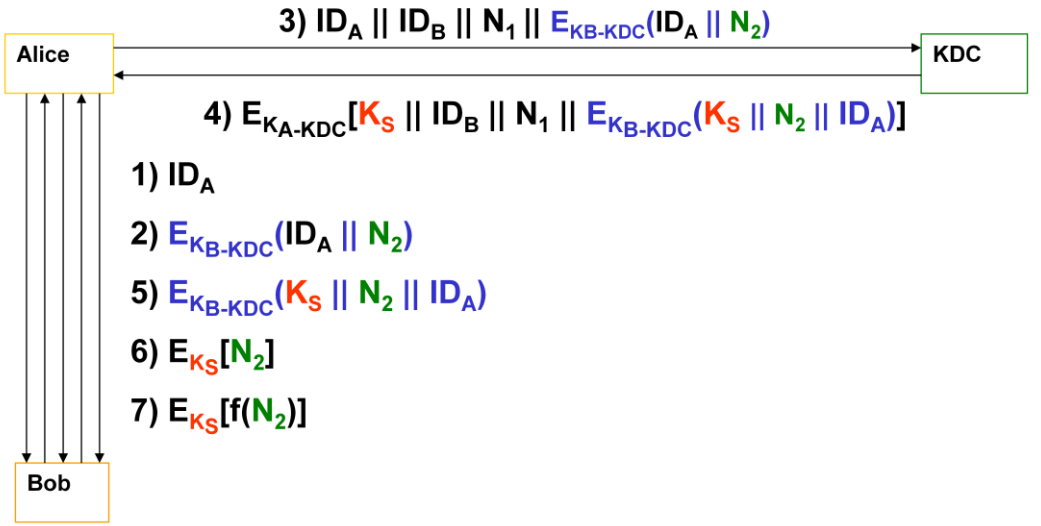
1) Alice asks KDC to initiate a communication with Bob and sends “nonce” N_1
2) KDC answers with encrypted (using Alice's key, K_{A-KDC}, which authenticates KDC w.r.t. Alice and achieves confidentiality) message consisting of the session key, Bob's identity, “nonce” N_1 (excluding replay), encrypted (using Bob's key, K_{B-KDC}) message (session key and Alice's identity); only Alice can decrypt this message and thus decrypt the session key
3) Alice sends session key and her own identity, encrypted with Bob's key (as received from KDC), after which Bob (and nobody else) can also decrypt the session key
4) Bob answers with second “nonce” N_2, encrypted with session key, authenticating himself w.r.t. Alice (by his knowledge of K_S)
5) Alice answers with processed (e.g. adding 1 to N_2) “nonce” N_2, encrypted with session key, authenticating herself w.r.t. Bob (using challenge-response mechanism)

Key exchange: KDC

- **Needham-Schroeder protocol**
  - **Security depends on:**
    - **Secret key $K_{A\text{-}KDC}$**
    - **Secret key $K_{B\text{-}KDC}$**
    - **Secret session key K_S**
      - ✓ Even after session has expired

- **Needham-Schroeder protocol**
  - **Attack against Bob using compromised K_S**
    - Carol replays step 3 using compromised K_S (only requires eavesdropping upon earlier communication)
    - Carol can answer Bob's challenge using compromised key K_S
    - Bob believes he's communicating with Alice, while he's in fact communicating with Carol
  - **OBS. 1: this kind of attack is very unlikely to be successful (recovering K_S is very hard)**
  - **OBS. 2: improved versions of this protocol exist**

Key exchange: KDC (improved)



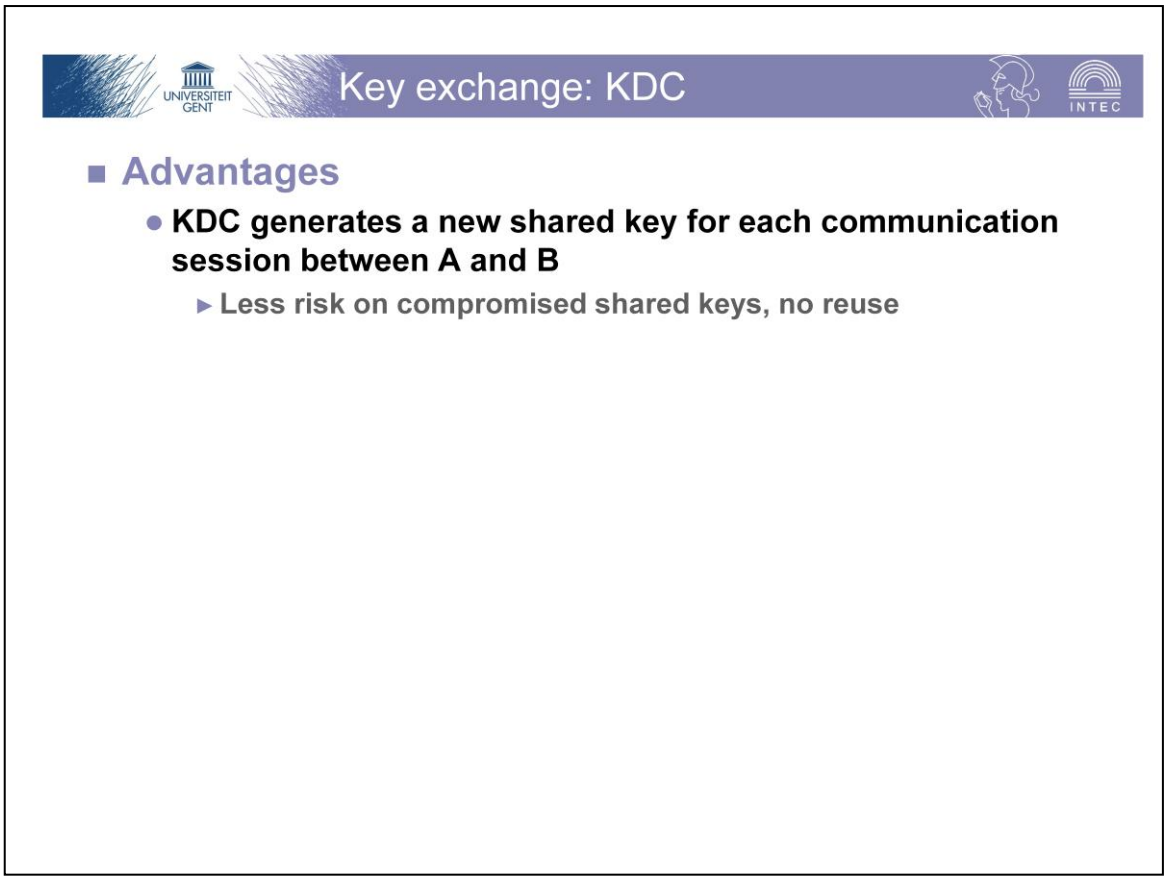
Correct security sometimes proves very subtle…

p. 31

N_1, N_2: nonces, which are used only once.

ID_A, ID_B: identities of Alice, resp. Bob

1) Alice informs Bob about communication
2) Bob answers with encrypted “nonce” (only readable to Bob and KDC)
3) Alice asks KDC to initiate a communication with Bob and sends “nonce” N_1 together with encrypted nonce N_2
4) KDC answers with encrypted (using Alice's key, K_{A-KDC}, which authenticates KDC w.r.t. Alice and achieves confidentiality) message consisting of the session key, Bob's identity, “nonce” N_1 (excluding replay), encrypted (using Bob's key, K_{B-KDC}) message (session key, “nonce” N_2 and Alice's identity); only Alice can decrypt this message and thus decrypt the session key
5) Alice sends session key, N_2, and own identity, encrypted with Bob's key (as received from KDC), after which Bob (and nobody else) can also decrypt the session key (“nonce” N_2 is a guarantee for the freshness of key K_S)
6) Bob answers with second “nonce” N_2, encrypted with session key, authenticating himself w.r.t. Alice (by his knowledge of K_S)
7) Alice answers with processed (e.g. adding 1 to N_2) “nonce” N_2, encrypted with session key, authenticating herself w.r.t. Bob (using challenge-response mechanism)



A typical operation with a KDC involves a request from a user to use some service. The KDC will use cryptographic techniques to authenticate requesting users as themselves. It will also check whether an individual user has the right to access the service requested. If the authenticated user meets all prescribed conditions, the KDC can issue a ticket permitting access.

KDCs mostly operate with symmetric encryption.

In most (but not all) cases the KDC shares a key with each of all the other parties.

The KDC produces a ticket based on a server key.

The client receives the ticket and submits it to the appropriate server.

The server can verify the submitted ticket and grant access to the user submitting it.

Security systems using KDCs include Kerberos. (Actually, Kerberos partitions KDC functionality between two different agents: the AS (Authentication Server) and the TGS (Ticket Granting Service).)

Key exchange: KDC

- **Cerberus**
  - **Greek methodology**
  - **Fitting name for a KDC**
    - 3-headed hellhound
    - Guardian of the underworld

KDC: Kerberos

- **User - password based authentication based on late-70's Needham -Schroeder algorithms.**
  - **Kerberos Authentication Server aka KDC (Key Distribution Center) shares long-term secret (password) with each authorized user.**
  - **User logs in and established a short term session key with the AS which can be used to establish his identity with other entities, e.g. file system, other hosts or services each of which trusts the authority server.**
- **The authorization mechanism needs to be integrated with the each function, e.g. file access, login, telnet, ftp, ...**
- **The central server is a single point of vulnerability to attack and failure.**
- **Been in use for 20 years. We are now at version 5.**

A typical operation with a KDC involves a request from a user to use some service. The KDC will use cryptographic techniques to authenticate requesting users as themselves. It will also check whether an individual user has the right to access the service requested. If the authenticated user meets all prescribed conditions, the KDC can issue a ticket permitting access.

KDCs mostly operate with symmetric encryption.

In most (but not all) cases the KDC shares a key with each of all the other parties.

The KDC produces a ticket based on a server key.

The client receives the ticket and submits it to the appropriate server.

The server can verify the submitted ticket and grant access to the user submitting it.

Security systems using KDCs include Kerberos. (Actually, Kerberos partitions KDC functionality between two different agents: the AS (Authentication Server) and the TGS (Ticket Granting Service).)

Overview

- Network model
- Secure configuration of devices
- Exchanging keys
  - Out of band
  - Diffie-Hellman
  - Asymmetric encryption
  - **Trusted third party**
    - Key Distribution Centre (KDC)
    - **Public Key Infrastructure (PKI)**
- Secure networking protocols
- Firewalls

Key exchange: PKI

- **Public key management**
  - **Issue:**
    - How does one know who owns some public key?
    - Building a trust relationship
  - **Solution:**
    - Simple system
    - PKI ("Public Key Infrastructure")

Key exchange: PKI

- Certificates allow key exchange without real-time access to public-key authority
- a certificate binds identity to public key
  - usually with other info such as period of validity, rights of use etc
- with all contents signed by a trusted Public-Key or Certificate Authority (CA)
- can be verified by anyone who knows the public-key authorities public-key

A further improvement is to use certificates, which can be used to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. A certificate binds an **identity** to **public key**, with all contents **signed** by a trusted Public-Key or Certificate Authority (CA). This can be verified by anyone who knows the public-key authorities public-key.

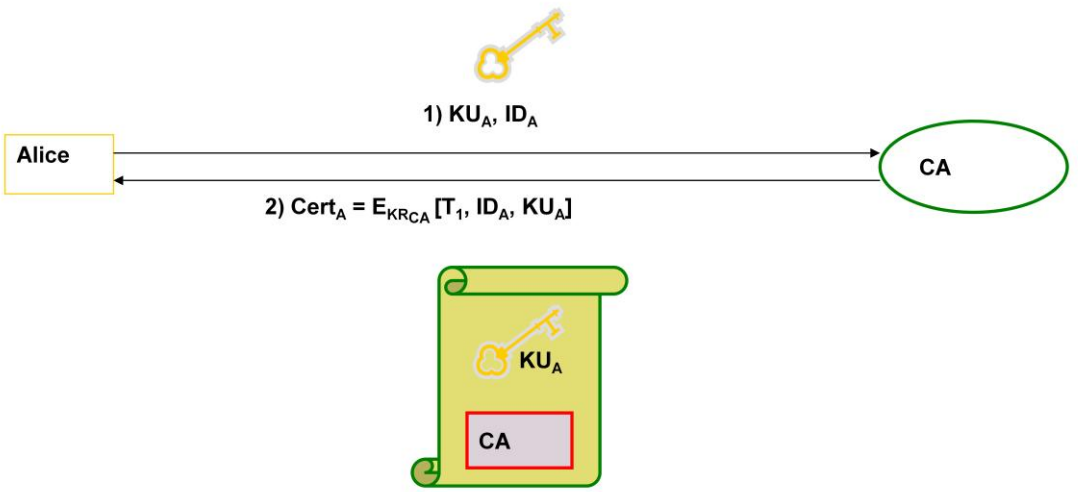
Key exchange: PKI

- **Using certification authority (CA)**
  - **Guarantees identity of the user of the public key**
    - Entity registers key at CA, where entity proves its identity
    - CA verifies identity and delivers certificate binding this identity to the public key
    - Entity can use certificate (with CA's guarantee) to prove its ownership of the public key
  - **For more about certificates, see also later (X.509)**

p. 38

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.

Certificates

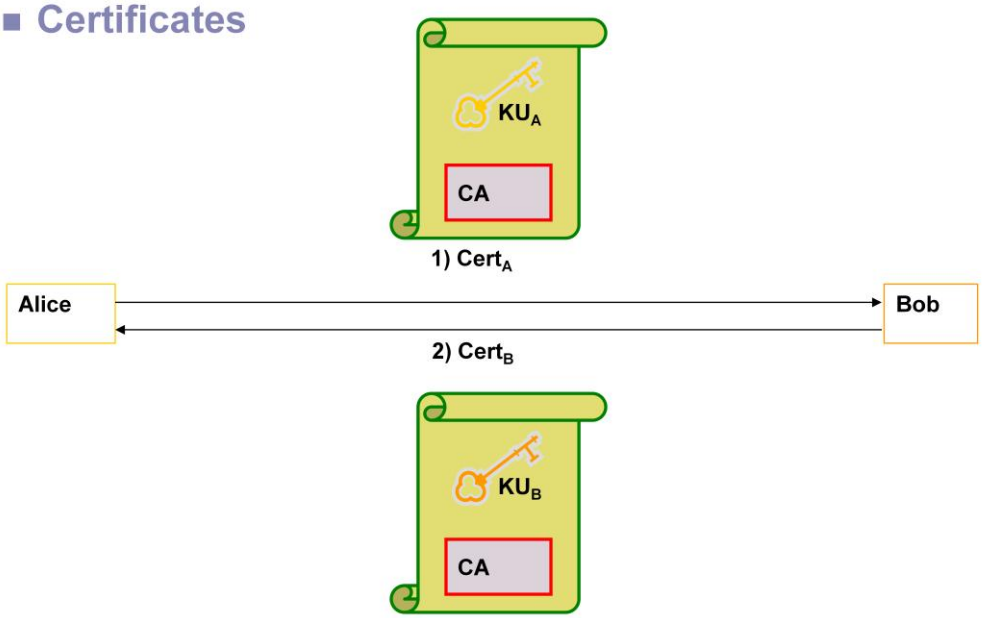


ID_A: Alice's identity

- Alice registers public key at CA and proves her identity
- CA creates a certificate binding Alice's public to her identity, together with some time value (validity of the certificate), using a digital signature. Certificate can be verified using the public key of the CA (KU_{CA}).

Key exchange: PKI

- Certificates

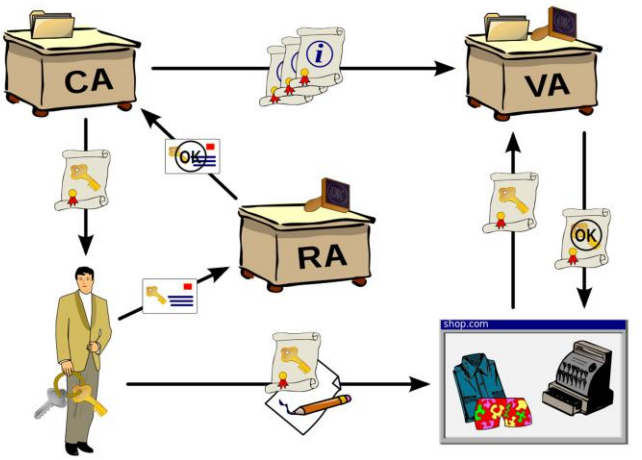


p. 40

1) Alice sends certificate she has obtained from CA to Bob, allowing Bob (using CA's public key) to verify the public key Alice uses really belongs to Alice
2) Bob sends his certificate back to Alice for verification

Key exchange: PKI

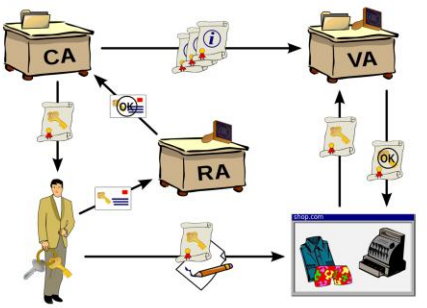
- CA – Certification Authority
- RA – Registration Authority
- VA – third-party validation authority



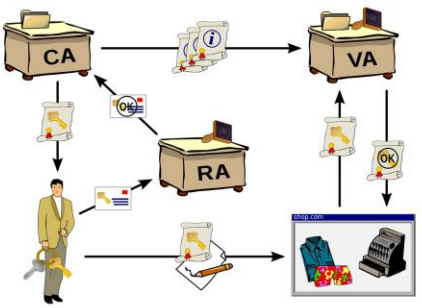
41

In cryptography, a PKI is an arrangement that binds public keys with respective user identities by means of a certificate authority (CA). The user identity must be unique within each CA domain. The third-party validation authority (VA) can provide this information on behalf of the CA. The binding is established through the registration and issuance process. Depending on the assurance level of the binding, this may be carried out by software at a CA or under human supervision. The PKI role that assures this binding is called the registration authority (RA). The RA is responsible for accepting requests for digital certificates and authenticating the person or organization making the request. In a Microsoft PKI, a registration authority is usually called a subordinate CA.

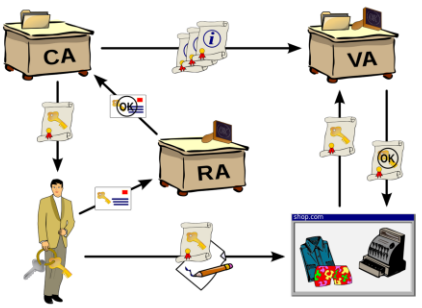
- **CA – *Certification Authority***
  - **Issuer/Signer of the certificate**
    - **Binds public key with identity+attributes**
  - **Provider**
    - **Enterprise CA**
    - **Individual as CA (PGP)**
      - ✓ Web of trust
    - **"Global" or "Universal" CAs**
      - ✓ VeriSign, Equifax, Entrust, CyberTrust, Identrus, …
  - **Trust is the key word**



- **RA – Registration Authority**
  - **Also called LRA – Local RA**
  - **Goal: off-load some work of CA to LRAs**
  - **Support all or some of:**
    - **Identification**
    - **User key generation/distribution**
      - ✓ passwords/shared secrets and/or public/private keys
    - **Interface to CA**
    - **Key/certificate management**
      - ✓ Revocation initiation
      - ✓ Key recovery



- VA – third-party validation authority
  - The third-party validation authority (VA) can provide this information on behalf of the CA.
    - Why is this useful?



Key exchange: PKI

- **Using certification authority (CA)**
  - **Remaining issue: CA's public key**
    - 1 order of magnitude smaller, as there are far fewer CAs than user

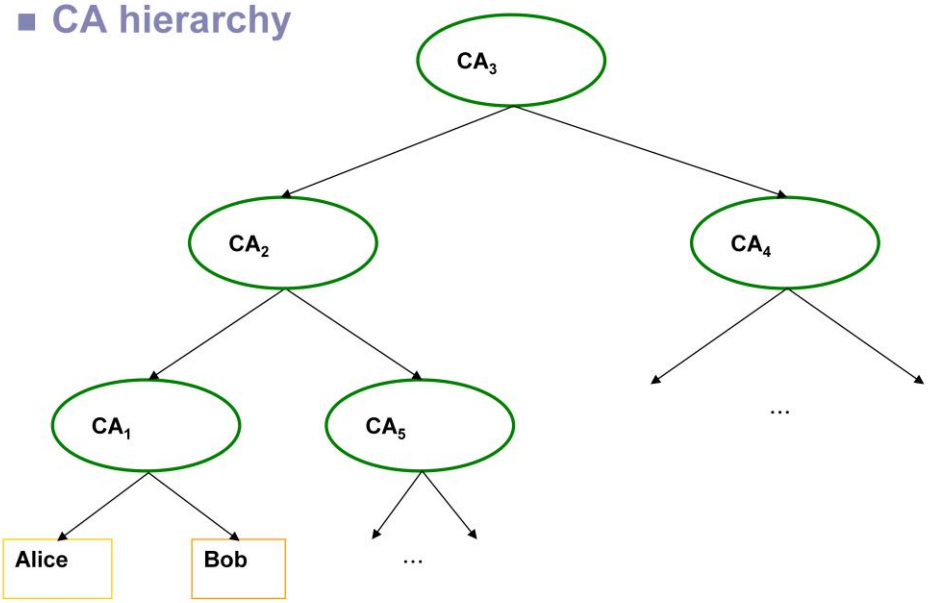
Key exchange: PKI

- Who to trust?
  - Which certificates can be trusted
- Source of Trust
  - How it is established?
- Limiting/controlling trust in a given environment

- Commonly used methods
  - CA Hierarchy
  - Distributed
  - Web
  - User-centric
  - Cross-certification

Key exchange: PKI

- CA hierarchy



p. 47

Have CA's public key signed by more important CA, thereby building a CA hierarchy (see also later X.509)

1) Public keys of Alice and Bob are signed by CA_1
2) Public key of CA_1 is signed by CA_2
3) Public key of CA_2 is signed by CA_3

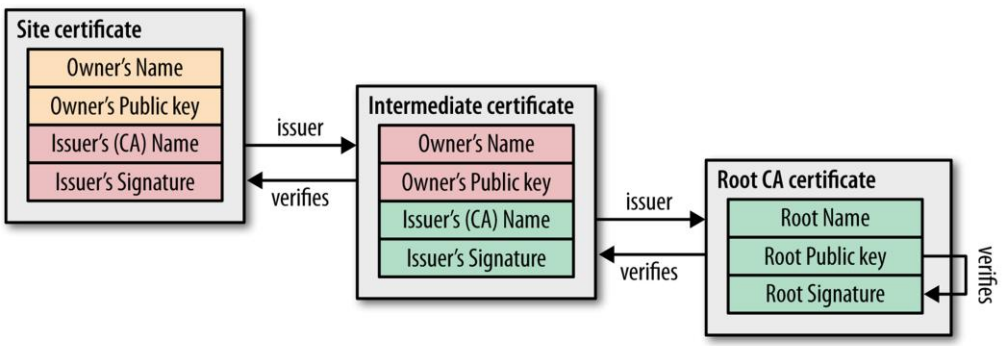
Until number of CAs is sufficiently limited, so that a limited list of trusted public keys exists (e.g. VeriSign, Belgian government, etc.)

4) CA_2 can also sign public keys of other CAs (such as CA_5), which in turn can sign the keys of other users/CAs

5) CA_3 at a higher level in the hierarchy can also in turn sign the public keys of other CAs (such as CA_4), etc.

Key exchange: PKI

- **Web model**
  - **A number of root CAs pre-installed in browsers**
  - **The set of root CAs can be modified**
    - **But will it be?**
  - **Root CAs are unrelated (no cross-certification)**
    - **Except by "CA powers" of browser manufacturer**
    - **Browser manufacturer = (implicit) Root CA**



Whom does your browser trust, and whom do you trust when you use the browser? There are at least three answers to this question:

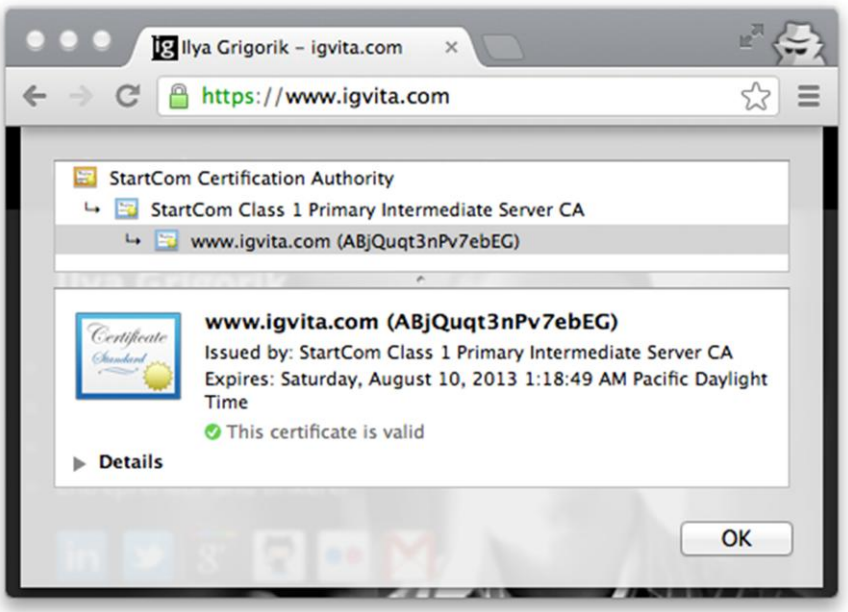
• Manually specified certificates: Every browser and operating system provides a mechanism for you to manually import any certificate you trust. How you obtain the certificate and verify its integrity is completely up to you.

• Certificate authorities: A certificate authority (CA) is a trusted third party that is trusted by both the subject (owner) of the certificate and the party relying upon the certificate.

• The browser and the operating system: Every operating system and most browsers ship with a list of well-known certificate authorities. Thus, you also trust the vendors of this software to provide and maintain a list of trusted parties.

In practice, it would be impractical to store and manually verify each and every key for every website (although you can, if you are so inclined). Hence, the most common solution is to use certificate authorities (CAs) to do this job for us: the browser specifies which CAs to trust (root CAs), and the burden is then on the CAs to verify each site they sign, and to audit and verify that these certificates are not misused or compromised. If the security of any site with the CA's certificate is breached, then it is also the responsibility of that CA to revoke the compromised certificate.

Web model



Every browser allows you to inspect the chain of trust of your secure connection, usually accessible by clicking on the lock icon beside the URL.

- igvita.com certificate is signed by StartCom Class 1 Primary Intermediate Server.
- StartCom Class 1 Primary Intermediate Server certificate is signed by the StartCom Certification Authority.
- StartCom Certification Authority is a recognized root certificate authority.

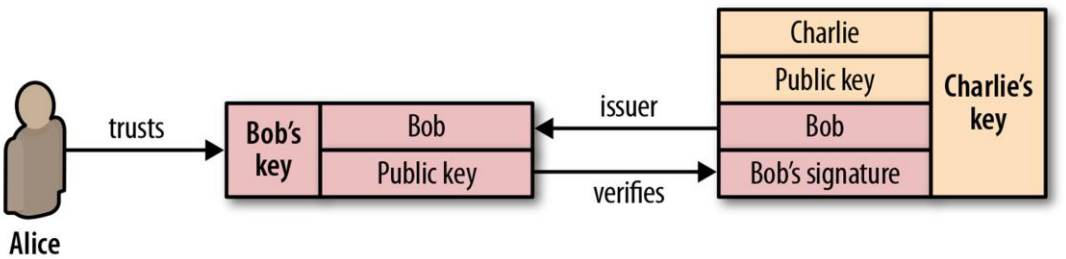
The "trust anchor" for the entire chain is the root certificate authority, which in the case just shown, is the StartCom Certification Authority. Every browser ships with a pre-initialized list of trusted certificate authorities ("roots"), and in this case, the browser trusts and is able to verify the StartCom root certificate. Hence, through a transitive chain of trust in the browser, the browser vendor, and the StartCom certificate authority, we extend the trust to our destination site.

Every operating system vendor and every browser provide a public listing of all the certificate authorities they trust by default. Use your favorite search engine to find and investigate these lists.

In practice, there are hundreds of well-known and trusted certificate authorities, which is also a common complaint against the system. The large number of CAs creates a potentially large attack surface area against the chain of trust in your browser.

Key exchange: PKI

- **User centric**
  - **PGP**
  - **User = her own Root CA**
    - **Webs of trust**
  - **Good**
    - **User fully responsible for trust**
  - **Bad**
    - **User fully responsible for trust**
    - **Corporate/gov/etc. like to have central control**
      - ✓ User-centric not friendly to centralized trust policies



A user centric model (also referred to as web of trust (WoT) model) relies on graph superconnectivity. With a WoT everybody is a CA, and each actor is its own and unique root CA; to cope with gullible or downright malicious "CA", WoT users (i.e. Web browsers) accept a target certificate as valid only if they can verify it through many paths which go through distinct CA and all concur to posit the target server key. WoT security is very dependent on critical mass: it will not give you much until sufficiently many people collaborate.

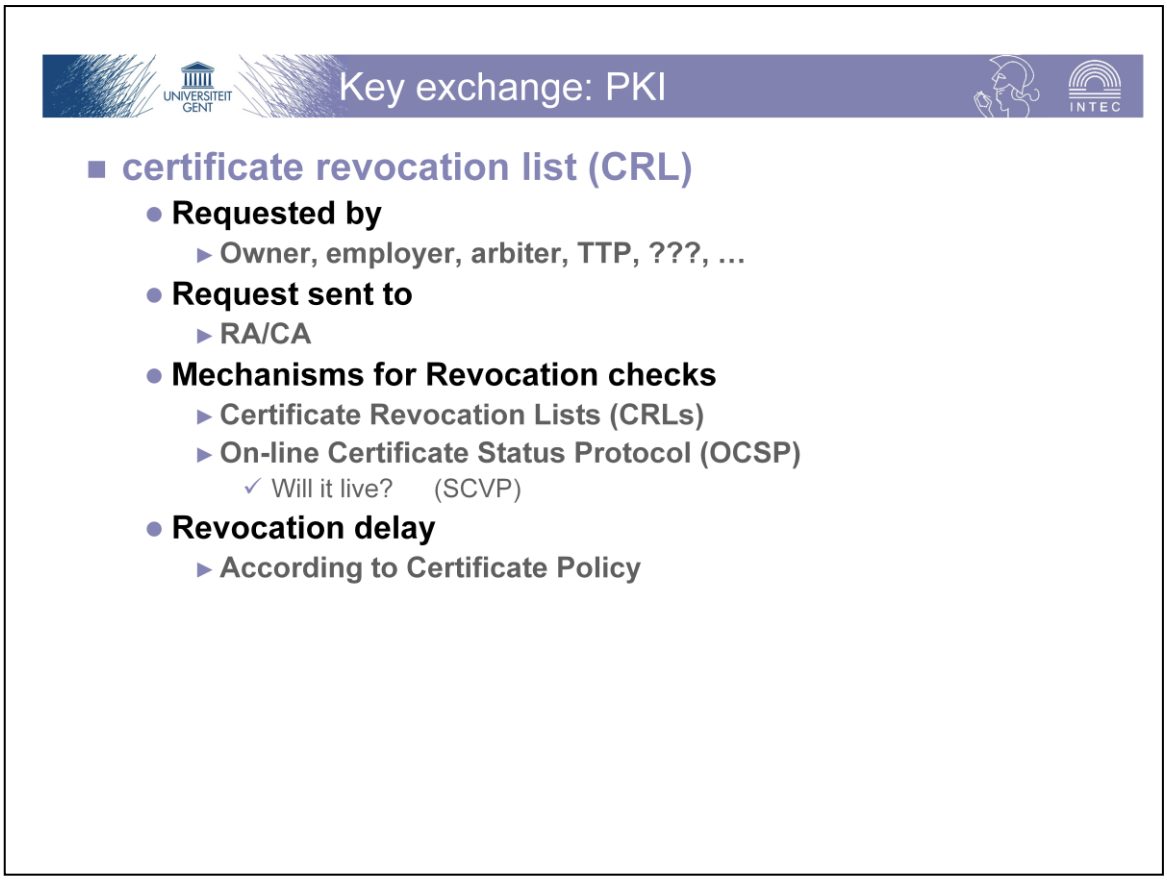
Alice and Bob could have exchanged their public keys when they met in person, and because they know each other well, they are certain that their exchange was not compromised by an impostor—perhaps they even verified their identities through another, secret (physical) handshake they had established earlier! Next, Alice receives a message from Charlie, whom she has never met, but who claims to be a friend of Bob's. In fact, to prove that he is friends with Bob, Charlie asked Bob to sign his own public key with Bob's private key and attached this signature with his message. In this case, Alice first checks Bob's signature of Charlie's key. She knows Bob's public key and is thus able to verify that Bob did indeed sign Charlie's key. Because she trusts Bob's decision to verify Charlie, she accepts the message and performs a similar integrity check on Charlie's message to ensure that it is, indeed, from Charlie. What we have just done is established a chain of trust: Alice trusts Bob, Bob trusts Charlie, and by transitive trust, Alice decides to trust Charlie. As long as nobody in the chain gets compromised, this allows us to build and grow the list of trusted parties.

Authentication on the Web and in your browser follows the exact same process as shown.

- **Cross-Certification**
  - **Mechanism:**
    - **Certificates for CAs (not end-entities)**
  - **Intra- vs. Inter- domain**
  - **One or two directions**
    - **CA1 certifies CA2 and/or CA2 certifies CA1**
  - **Control**
    - **Cross-certificate limits trust**
      - ✓ Name, policy, path length, etc. constraints

Key exchange: PKI

- Certificate renewal
  - Same keys, same cert, but new dates
  - Preferably automatic
  - but watch for attributes change!
- Certificate history
  - Key history
    - For owner: eg to read old encrypted msgs
  - Key archive
    - “For public”: audit, old sigs, disputes, etc.
- Certificate cancellation
  - Certificate Expiration
    - Natural “peaceful” end of life
  - Certificate Revocation
    - Untimely death, possibly dangerous causes
    - New keys, new certificate



Occasionally the issuer of a certificate will need to revoke or invalidate the certificate due to a number of possible reasons: the private key of the certificate has been compromised, the certificate authority itself has been compromised, or due to a variety of more benign reasons such as a superseding certificate, change in affiliation, and so on.

Certificate Revocation List (CRL)

Certificate Revocation List (CRL) is defined by RFC 5280 and specifies a simple mechanism to check the status of every certificate: each certificate authority maintains and periodically publishes a list of revoked certificate serial numbers. Anyone attempting to verify a certificate is then able to download the revocation list and check the presence of the serial number within it—if it is present, then it has been revoked. The CRL file itself can be published periodically or on every update and can be delivered via HTTP, or any other file transfer protocol. The list is also signed by the CA, and is usually allowed to be cached for a specified interval. In practice, this workflow works quite well, but there are instances where CRL mechanism may be insufficient:

• The growing number of revocations means that the CRL list will only get longer, and each client must retrieve the entire list of serial numbers.

• There is no mechanism for instant notification of certificate revocation—if the CRL was cached by the client before the certificate was revoked, then the CRL will deem the revoked certificate valid until the cache expires.

Online Certificate Status Protocol (OCSP)

To address some of the limitations of the CRL mechanism, the Online Certificate Status Protocol (OCSP) was introduced by RFC 2560, which provides a mechanism to perform a real-time check for status of the certificate. Unlike the CRL, which contains all the revoked serial numbers, OCSP allows the verifier to query the certificate database directly for just the serial number in question while validating the certificate chain. As a result, the OCSP mechanism should consume much less bandwidth and is able to provide real-time validation. However, no mechanism is perfect, and the requirement to perform real-time OCSP queries creates several problems of its own:

• The CA must be able to handle the load of the real-time queries.

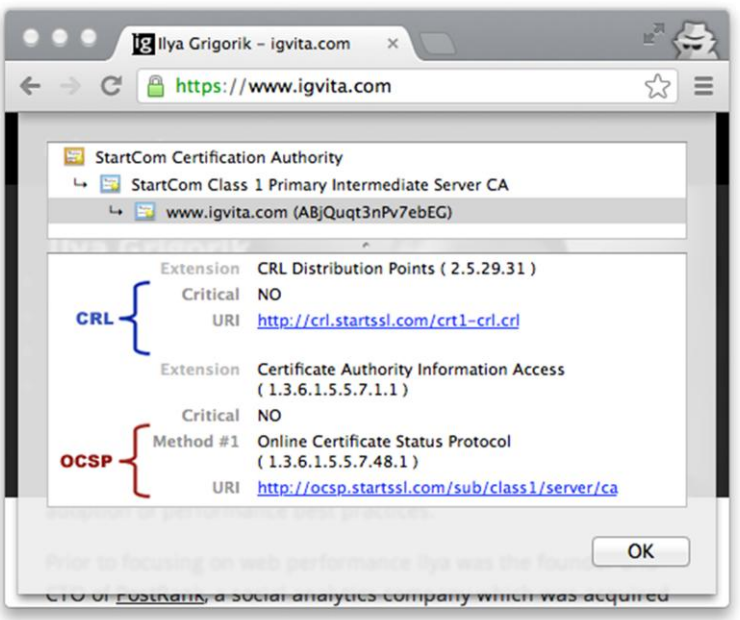
• The CA must ensure that the service is up and globally available at all times.

• The client must block on OCSP requests before proceeding with the navigation.

• Real-time OCSP requests may impair the client's privacy because the CA knows which sites the client is visiting.

In practice, CRL and OCSP mechanisms are complementary, and most certificates will provide instructions and endpoints for both. The more important part is the client support and behavior: some browsers distribute their own CRL lists, others fetch and cache the CRL files from the CAs. Similarly, some browsers will perform the real-time OCSP check but will differ in their behavior if the OCSP request fails. If you are curious, check your browser and OS certificate revocation settings!

- certificate revocation list (CRL)



To address revocations, the certificates themselves contain instructions on how to check if they have been revoked. Hence, to ensure that the chain of trust is not compromised, each peer can check the status of each certificate by following the embedded instructions, along with the signatures, as it walks up the certificate chain.

Overview

- Network model
- Secure configuration of devices
- Exchanging keys
  - Out of band
  - Diffie-Hellman
  - Asymmetric encryption
  - Trusted third party
    - Key Distribution Centre (KDC)
    - Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

Besides the lecturers' own material, many third party, often copyrighted, material is reused within this lecture (e.g. in the notes) under the 'fair use' approach, for sake of educational purpose only, and very limited edition. As a consequence, the current slide set presentation usage is restricted, and is falling under usual copyrights usage.

At the end of every lecture, appropriate references to used materials are included.

References

- This work contains content adapted from, amongst others, the following sources (in no particular order)
  - **Books**
    - William Stallings, "**Cryptography and Network Security, principles and practices**", 6th (international) edition, Prentice Hall, 2010;
    - Matt Bishop, "**Computer Security: Art and Science**", Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7
  - **Lecture slides:**
    - "Informatiebeveiliging", Universiteit Gent, Eric Laermans & Thom Dhaene
    - Stallings, 2014, Lecture slides by Lawrie Brown
  - **Wikipedia**
    - Note page descriptions
  - **Websites**
    - https://crypto.stackexchange.com
    - http://www.roguelynn.com/words/explain-like-im-5-kerberos/
    - http://chimera.labs.oreilly.com/books/1230000000545/ch04.html#COT_CA