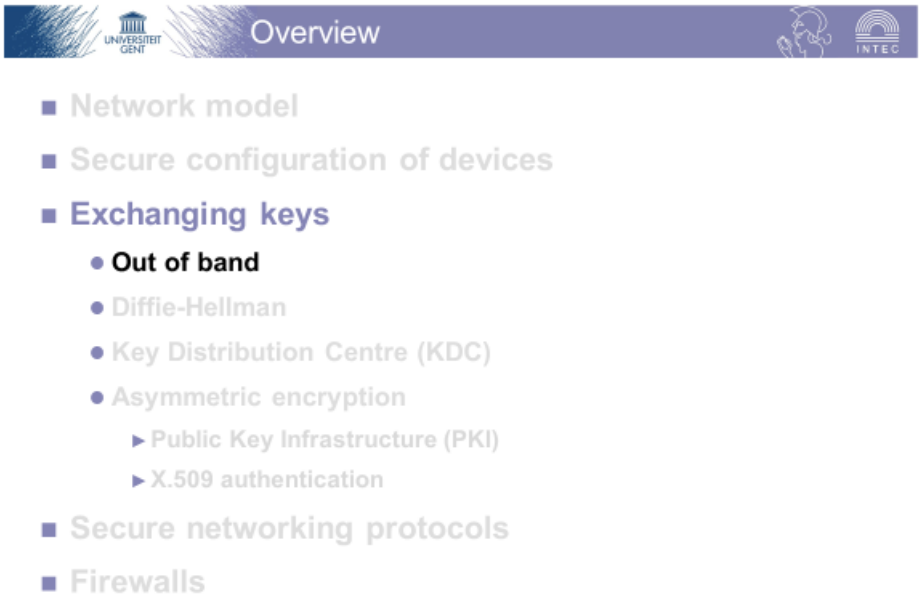
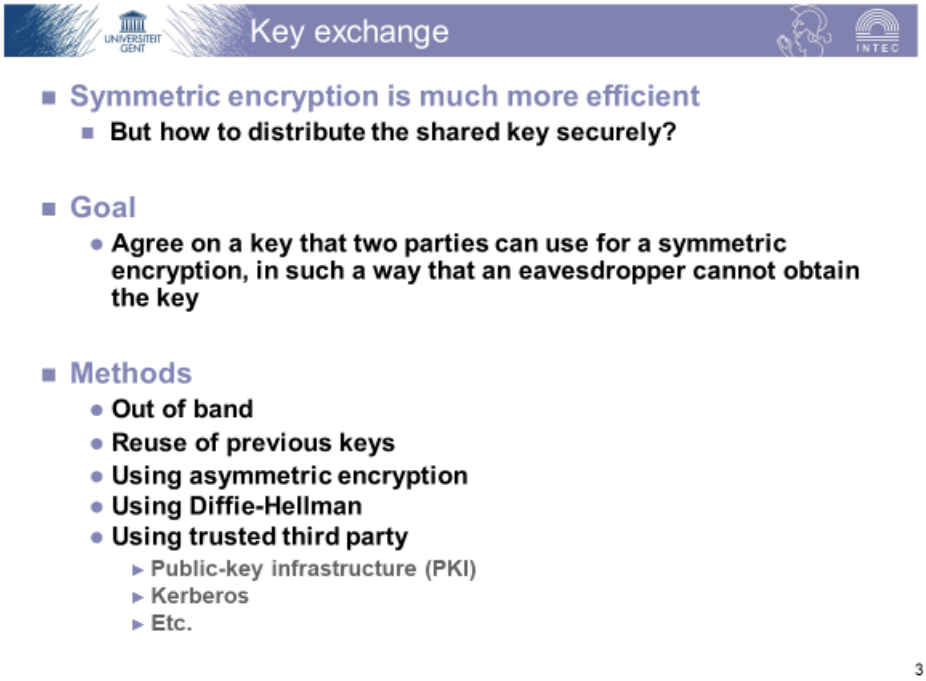




Given two parties A and B, multiple **key distribution** alternatives exist

1. A can select a key and physically deliver it to B. These out of band exchanges (non-electronically) are mostly 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. As such it is mostly suited for acquaintances, but not for large organizations.

2. If A & B have communicated previously can use the previous key to encrypt a new key, as well as to authenticate both parties.
3. Asymmetrical encryption can be used to encrypt a generated key
4. Diffie-Hellman can be used to generate keys (see later), but requires a separate mechanism for authentication.
5. A third party, whom all parties trust, can be used as a **trusted intermediary** to mediate the establishment of secure communications between them
    1. He can select & physically deliver key to A & B
    2. If A & B have secure communications with a third party C, C can relay the key between A & B
    3. The clients must trust the intermediary not to abuse the knowledge of all session keys. As the number of directly involved parties grows, this approach is 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**
  - Key Distribution Centre (KDC)
  - Asymmetric encryption
    - Public Key Infrastructure (PKI)
    - X.509 authentication
- Secure networking protocols
- Firewalls

4

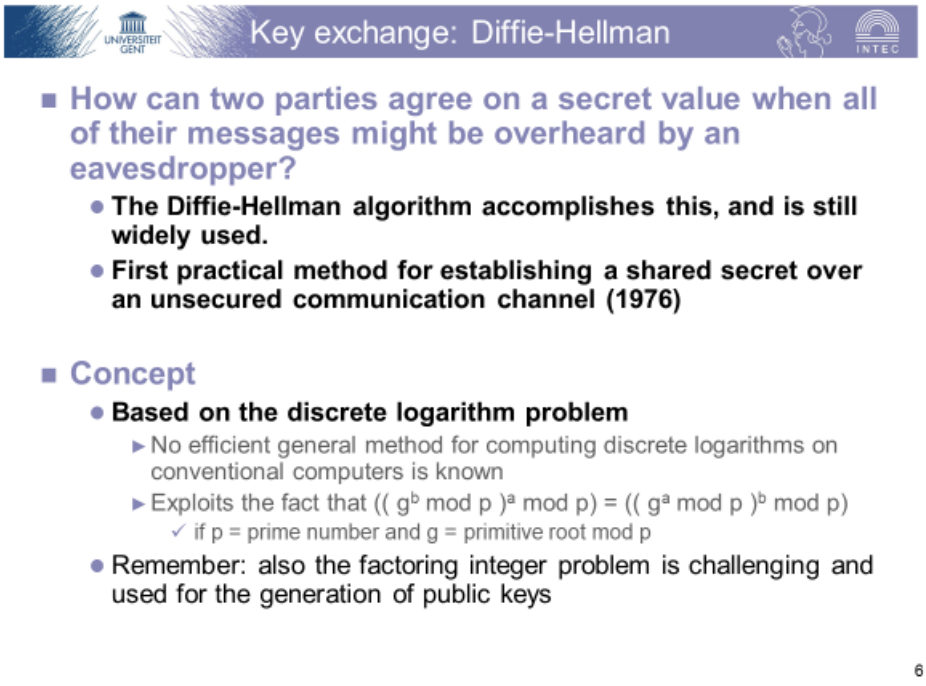
Key exchange: Diffie-Hellman

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

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

5

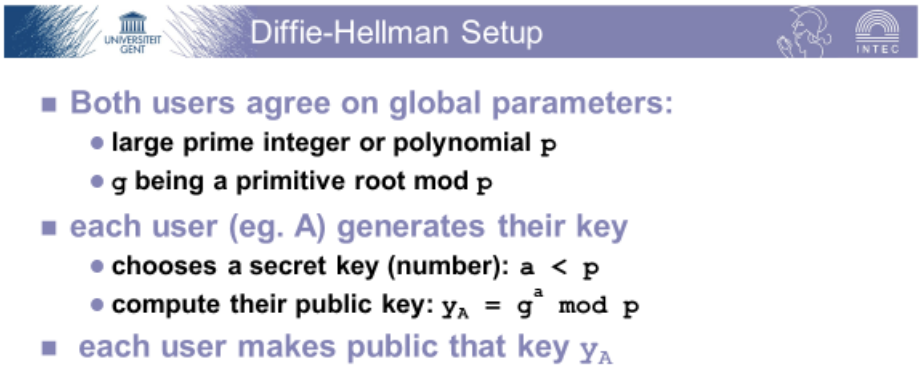
In modular arithmetic a number g is a primitive root modulo p if for every integer a coprime to p, there is an integer k such that $g^k \equiv a \pmod p$. In other words, g is a generator of the multiplicative group of integers modulo p.



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, it is 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.

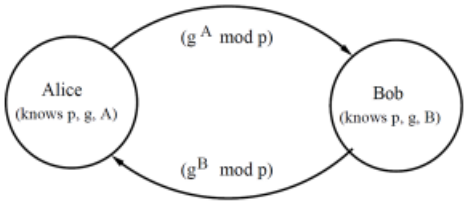
While computing discrete logarithms and factoring integers are distinct problems, they share some properties:

- 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.



In the Diffie-Hellman key exchange algorithm, there are two publicly known numbers: a prime number p and an integer "g" that is a primitive root of p. The prime p and primitive root g can be common to all using some instance of the D-H scheme. Note that the primitive root g 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

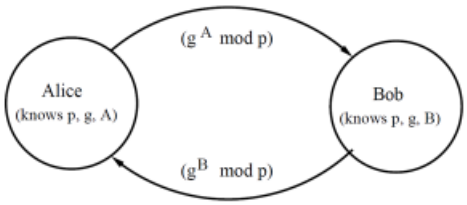


- **Steps in the algorithm:**
  1. **Alice and Bob agree on a prime number p and a base g**
     - These do not need to be kept secret
  2. **Alice chooses a secret number a, and sends $y_A = g^a \bmod p$.**
  3. **Bob chooses a secret number b, and sends $y_B = g^b \bmod p$.**
  4. **Alice computes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

8

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.

9

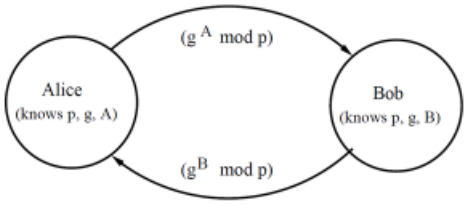
Alice chooses a = 6 and sends $5^6 \bmod 23 = 8$.

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

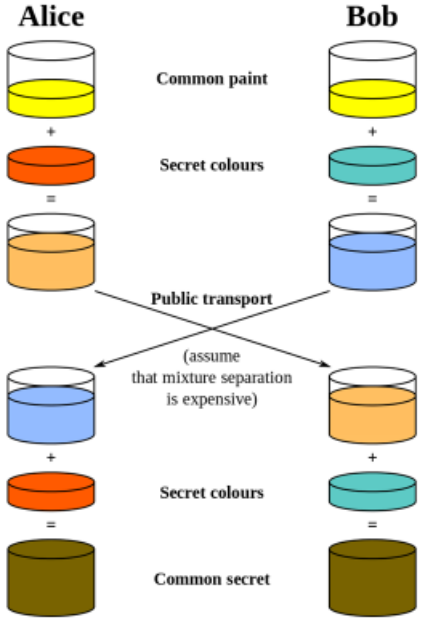


- Complexity example
  - p is a prime of around 300 digits
  - a and b at least 100 digits each.
- Discovering the shared secret given g, p, g^a mod p and g^b mod p would take longer than the lifetime of the universe, using the best known algorithm.
  - This is called the discrete logarithm problem

10

Key exchange: Diffie-Hellman

- Discrete logarithm problem: illustration



11

Issues with basic DH schemes

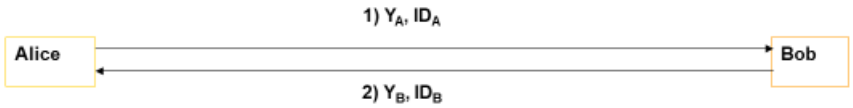
- **Denial-of-Service attacks**
  - **When basic scheme is used**
    - **After receiving attacker's public key the victim will…**
      - ✓ …compute the public key and send it to the attacker
        - » Who may have given a false address to that purpose
      - ✓ …compute the session key (in each configuration)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

12

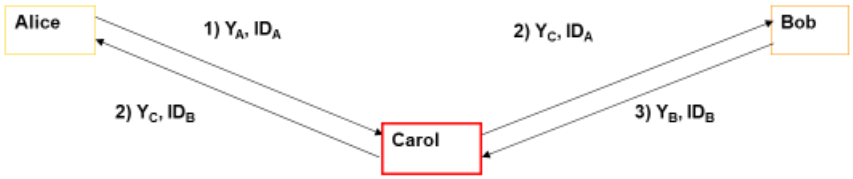
This type of attack is less severe against targets that use *fixed Diffie Helman* (see later), a scheme in which no new keys need to be computed for each session. To prevent this attack from happening, *ephemeral Diffie Helman (see later)* includes a separate prior authentication mechanism that is less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

Issues with basic DH schemes

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



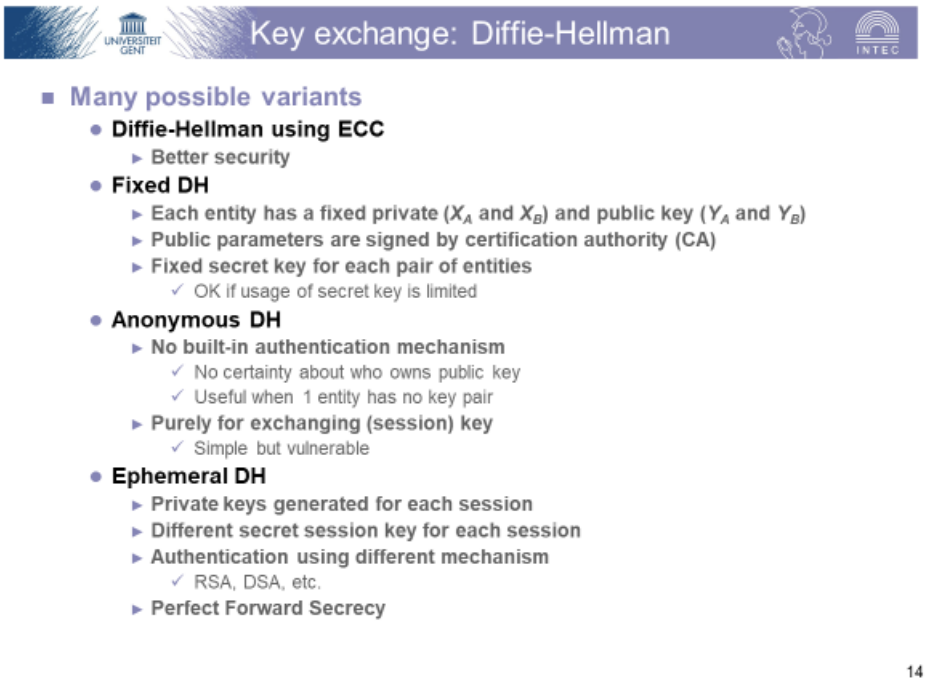
  - **Attack scheme**



13

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.



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?

15

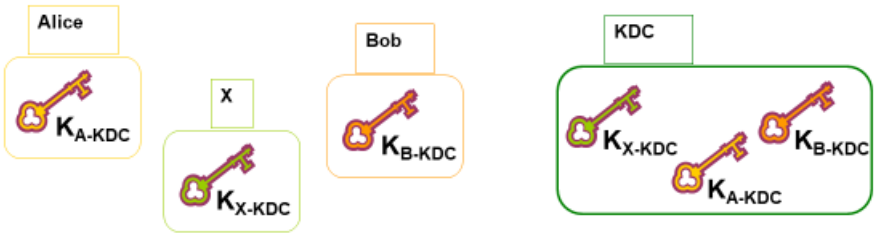
Overview

- Network model
- Secure configuration of devices
- Exchanging keys
  - Out of band
  - Diffie-Hellman
  - Key Distribution Centre (KDC)
  - Asymmetric encryption
    - Public Key Infrastructure (PKI)
    - X.509 authentication
- Secure networking protocols
- Firewalls

16

Key exchange: KDC

- Alternative using symmetrical encryption and a trusted server
  - Each user owns secret key allowing him to communicate with key distribution centre (KDC)



17

Key Distribution Centre (KDC). This approach improves security by tightening control over distribution of the keys. The KDC has properties of a directory and requires users to know the shared key to access the directory. Users interact with the directory to obtain any desired key securely. However, this approach does require real-time access to directory when keys are needed.

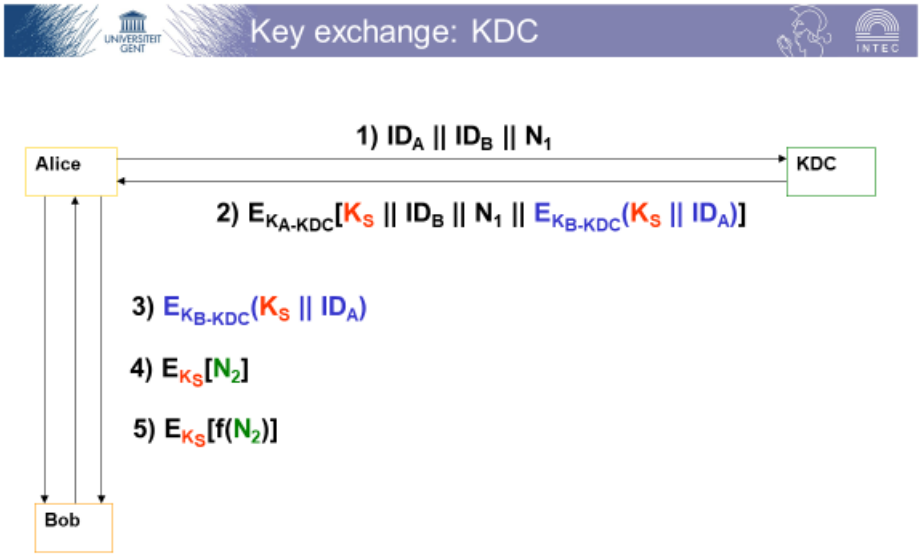
Key exchange: KDC

- Typically a hierarchy of keys
  - session key
    - temporary key
    - used for encryption of data between users
    - for one logical session then discarded
  - master key
    - used to encrypt session keys
    - shared by user & key distribution center

18

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.



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

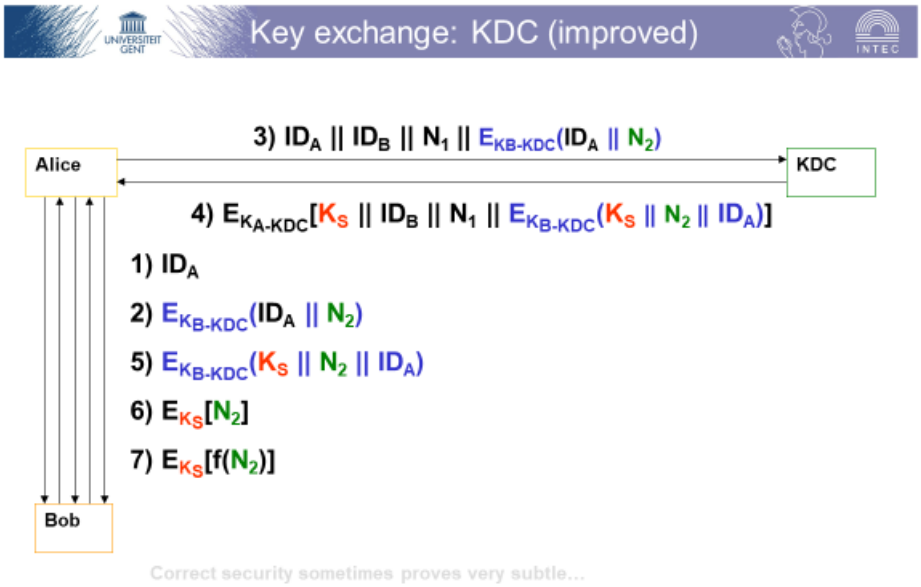
- Needham-Schroeder protocol
  - Security depends on:
    - Secret key $K_{A\text{-}KDC}$
    - Secret key $K_{B\text{-}KDC}$
    - Secret session key K_S
      - ✓ Even after session has expired

20

Key exchange: KDC

- 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

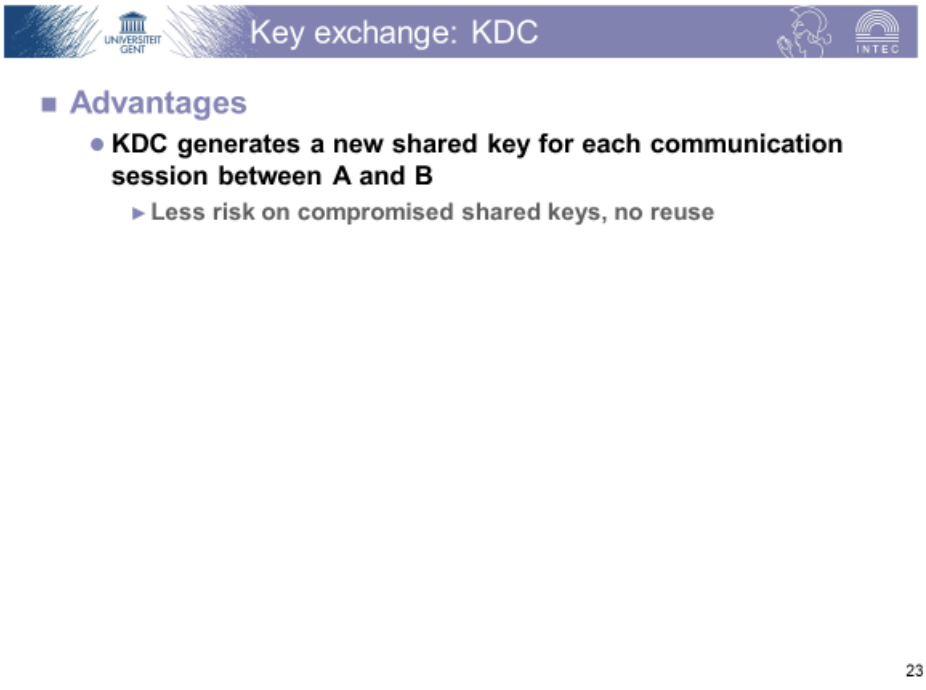
21



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

24

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 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.

25

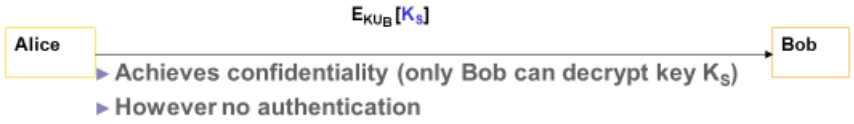
Overview

- Network model
- Secure configuration of devices
- Exchanging keys
  - Out of band
  - Diffie-Hellman
  - Key Distribution Centre (KDC)
  - **Asymmetric encryption**
    - Public Key Infrastructure (PKI)
    - X.509 authentication
- Secure networking protocols
- Firewalls

26

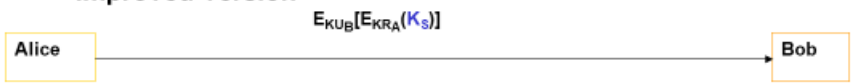
Key exchange: asymmetrical approach

- **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?)

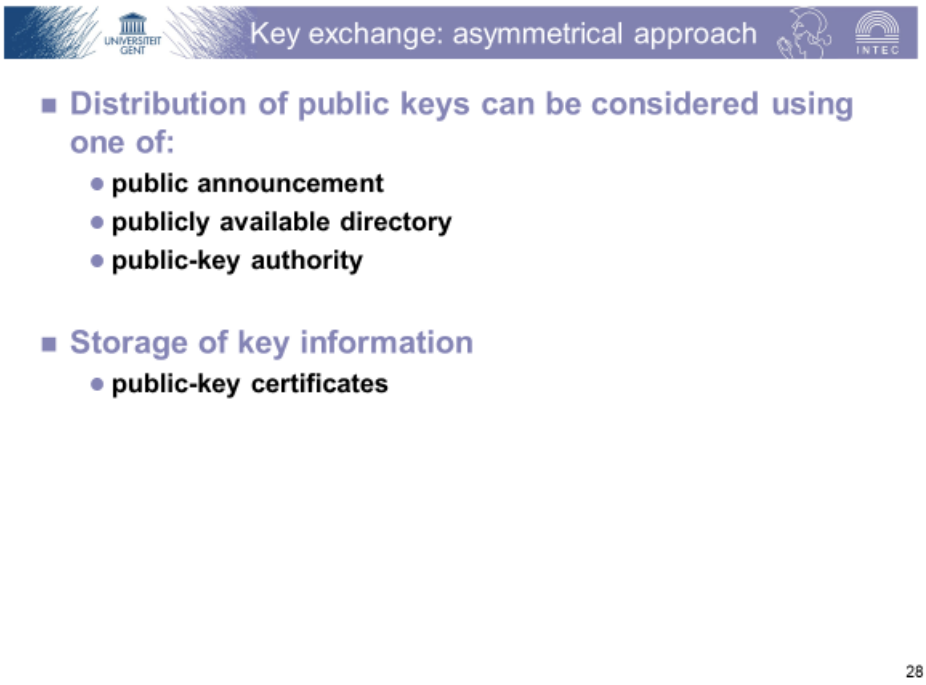
27

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

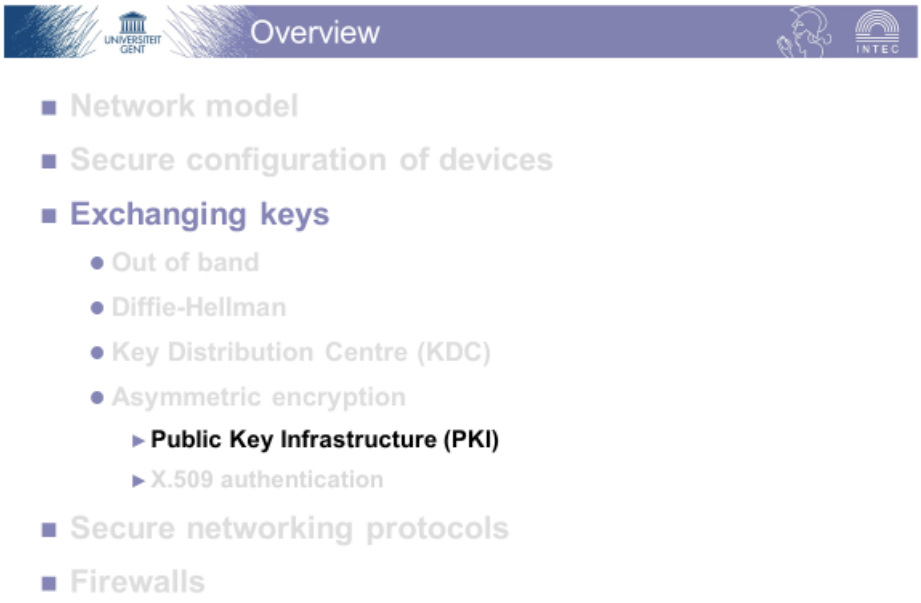
However, this approach is insecure against an adversary who can intercept messages and then either relay the intercepted message or substitute another message. The opponent

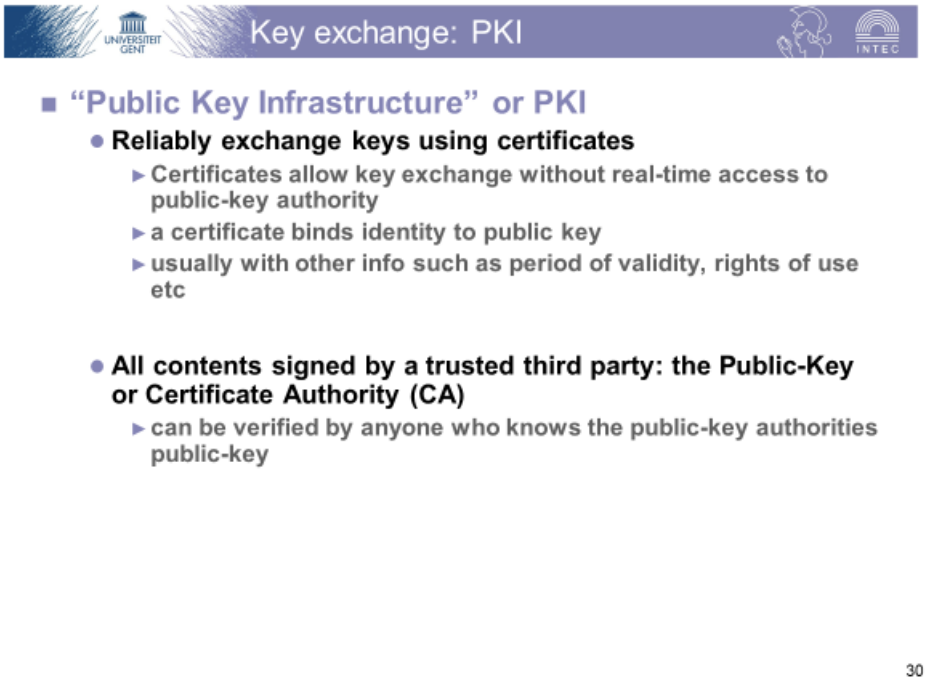
can thereby impersonate both halves of protocol (i.e. a man-in-the-middle attack). As such, this approach is seldomly used in practice.



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.

- **Public Announcement.** In this approach, users distribute public keys to recipients or broadcast their public keys to the community at large. This can be done be e.g. appending PGP keys to email messages or by posting them to news groups or email list. The major disadvantage of this approach is the possibility of forgery. Anyone can create a key claiming to be someone else and broadcast it. Until the forgery is discovered the adversary can masquerade as the claimed user.
- **Publicly Available Directory**. A greater degree of security can be achieved by maintaining a publicly available managed directory of public keys. This directory contains {name,public-key} entries of multiple parties. Participants register securely with the directory and can replace their key at any time. The content of the directory is periodically published and can be accessed electronically. Typically, uses have to prove their identity somehow before they can create an account. 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.
- **Public Key Authority**. In this case, a trusted third party manages the public keys of users. It is similar in function to a Key Distribution Center and uses similar information exchanges.





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. Certificates can be downloaded using any of the previously discussed distribution methods.

Key exchange: PKI

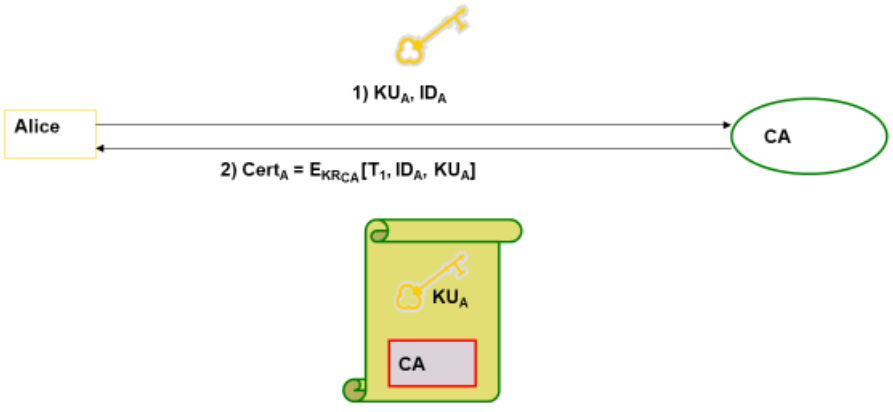
- Using certification authority (CA)
  - Guarantees identity of the user of the public key
    - Entity registers key at CA, where entity proves its identity
    - CA verifies identity and delivers certificate binding this identity to the public key
    - Entity can use certificate (with CA's guarantee)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

31

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.

Key exchange: PKI

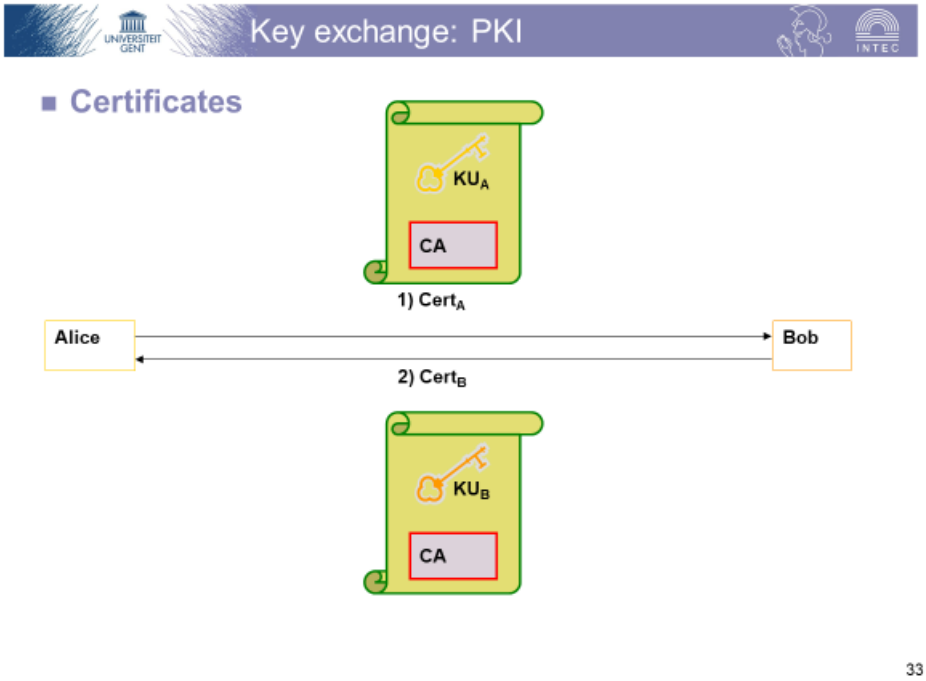
- Certificates



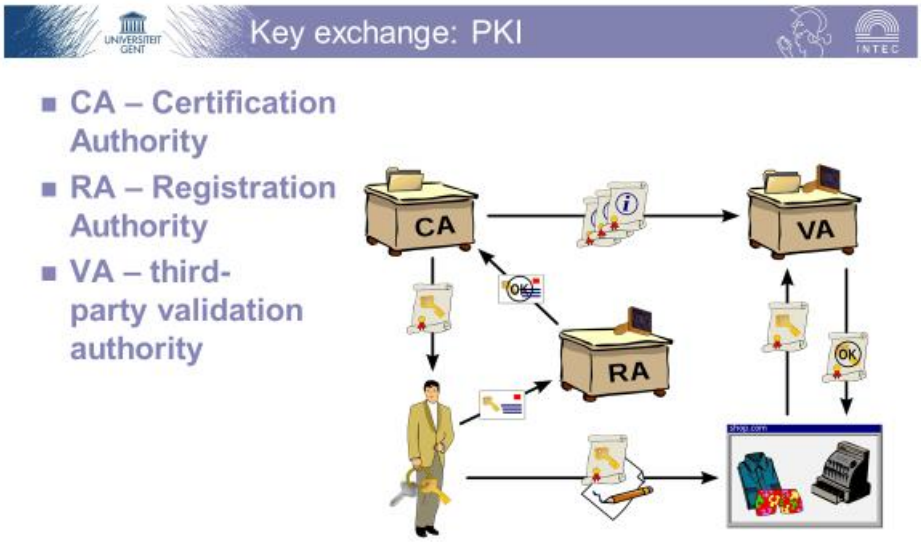
32

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}).



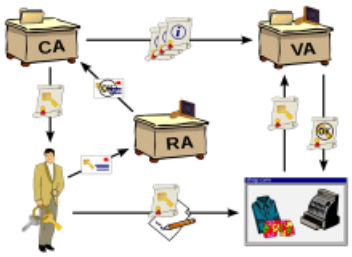
1) Alice sends certificate she has obtained from CA to Bob, allowing Bob (using CA's public key) to verify the public key Alice uses really belongs to Alice
2) Bob sends his certificate back to Alice for verification



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.

Key exchange: PKI

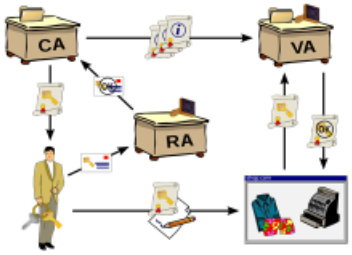
- **CA – Certification Authority**
  - **Issuer/Signer of the certificate**
    - **Binds public key with identity+attributes**
  - **Provider**
    - **Enterprise CA**
    - **Individual as CA (PGP)**
      - ✓ Web of trust
    - **"Global" or "Universal" CAs**
      - ✓ VeriSign, Equifax, Entrust,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



35

Key exchange: PKI

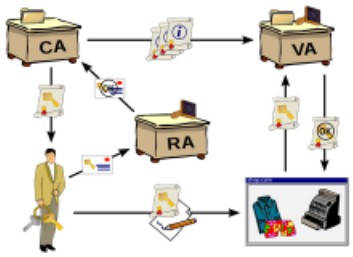
- **RA – Registration Authority**
  - **Also called LRA – Local RA**
  - **Goal: off-load some work of CA to LRAs**
  - **Support all or some of:**
    - **Identification**
    - **User key generation/distribution**
      - ✓ passwords/shared secrets and/or public/private keys
    - **Interface to CA**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



36

Key exchange: PKI

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



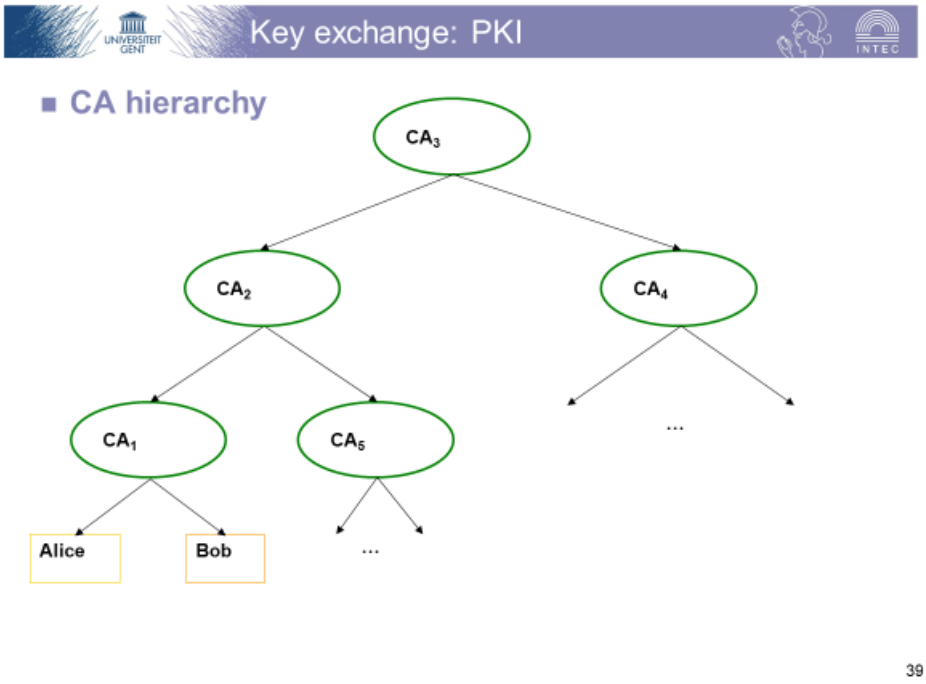
37

Key exchange: PKI

- Remaining issue: who to trust?
  - Which certificates can be trusted?
  - CA's public key
    - 1 order of magnitude smaller, as there are far fewer CAs than user

- Commonly used methods to limit/control trust in a given environment
  - CA Hierarchy
  - Distributed
  - Web
  - User-centric
  - Cross-certification

38

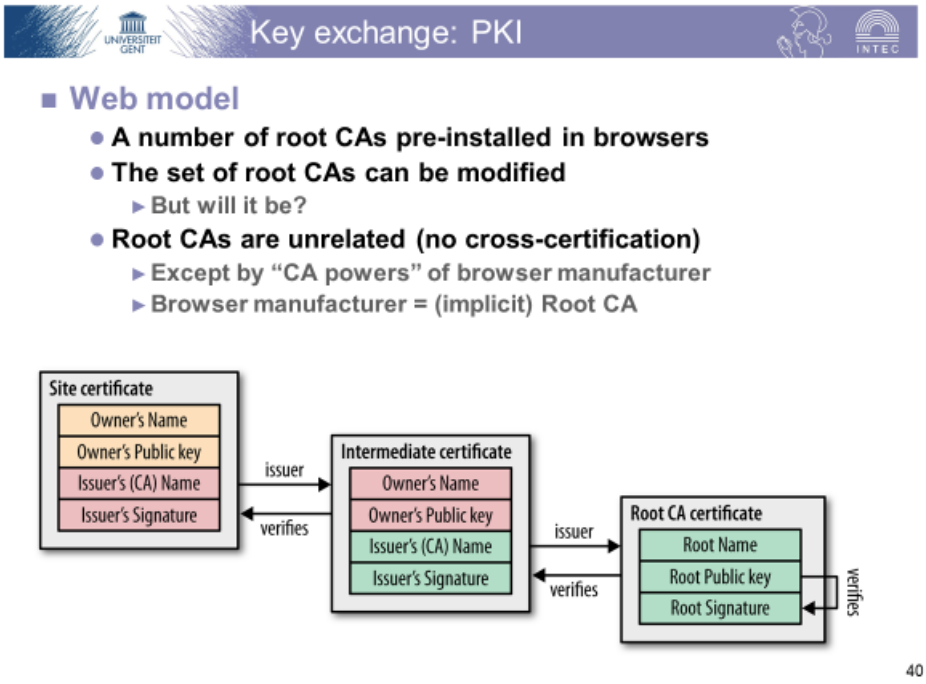


Have CA's public key signed by more important CA, thereby building a CA hierarchy (see also later X.509)

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.)

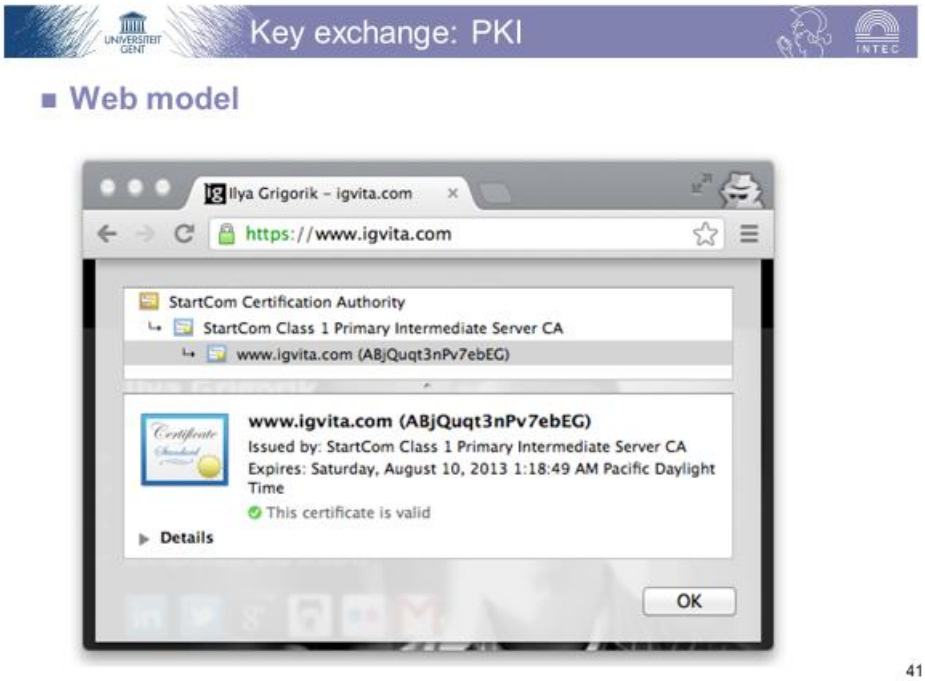
4) CA_2 can also sign public keys of other CAs (such as CA_5), which in turn can sign the keys of other users/CAs
5) CA_3 at a higher level in the hierarchy can also in turn sign the public keys of other CAs (such as CA_4), etc.



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.



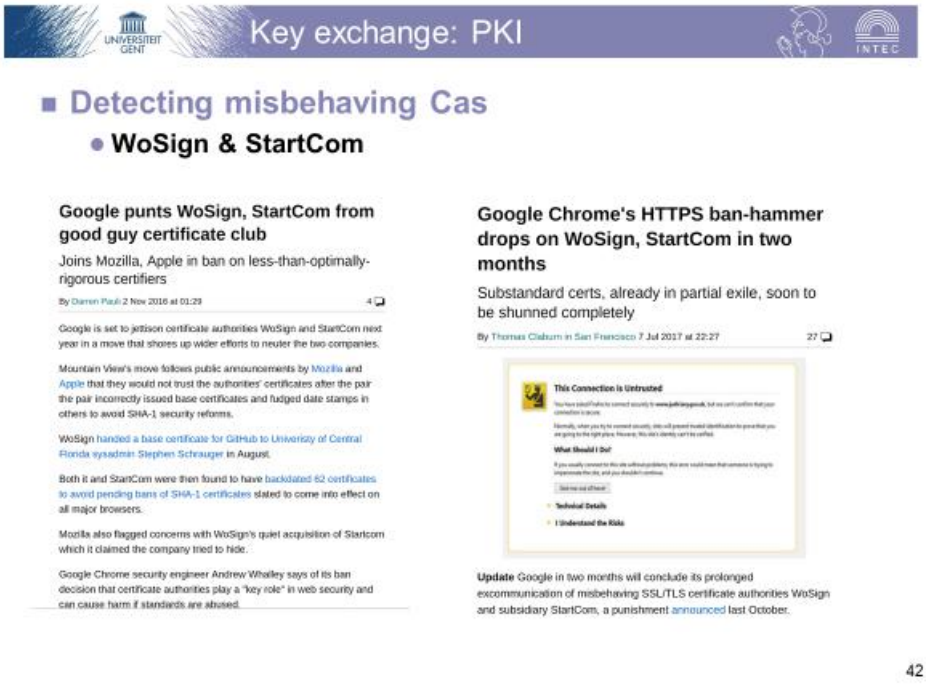
Every browser allows you to inspect the chain of trust of your secure connection, usually accessible by clicking on the lock icon beside the URL.

- 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.

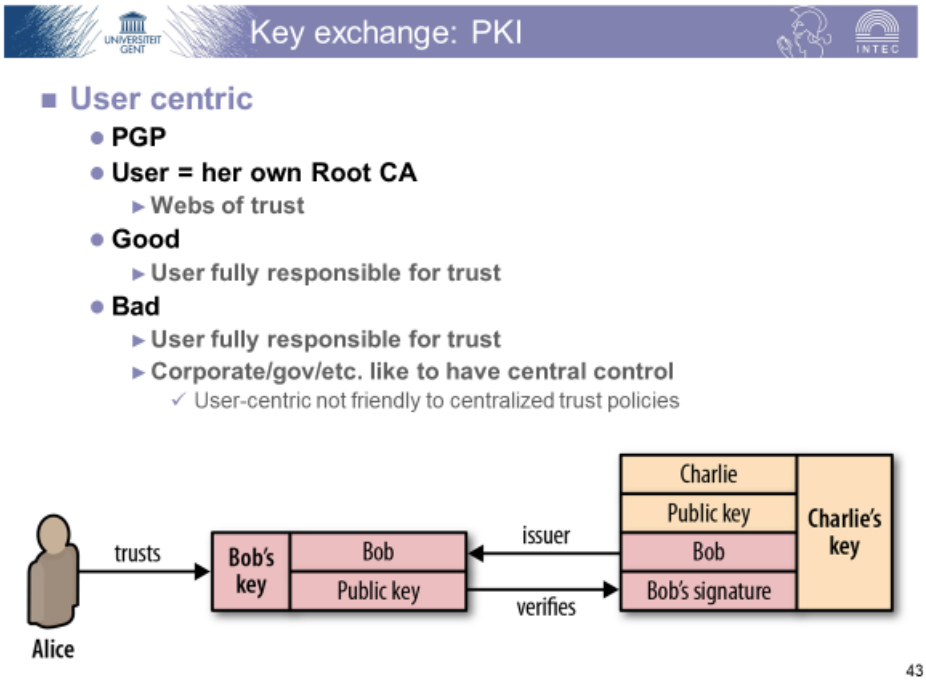
In practice, there are hundreds of well-known and trusted certificate authorities, which is also a common complaint against the system. The large number of CAs creates a potentially large attack surface area against the chain of trust in your browser.



hthttps://www.theregister.co.uk/2017/07/07/google_ban_hammer_drops_on_wosign_startcom_in_two_months/

https://techcrunch.com/2018/10/08/chrome-hundreds-of-sites-to-break/

Chrome 70 is expected to be released on or around October 16, when the browser will start blocking sites that run older Symantec certificates issued before June 2016, including legacy branded Thawte, VeriSign, Equifax, GeoTrust and RapidSSL certificates. Yet despite more than a year to prepare, many popular sites are not ready. Security researcher Scott Helme found 1,139 sites in the top one million sites ranked by Alexa, including Citrus, SSRN, the Federal Bank of India, Pantone, the Tel-Aviv city government, Squatty Potty and Penn State Federal to name just a few.



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.

Key exchange: PKI

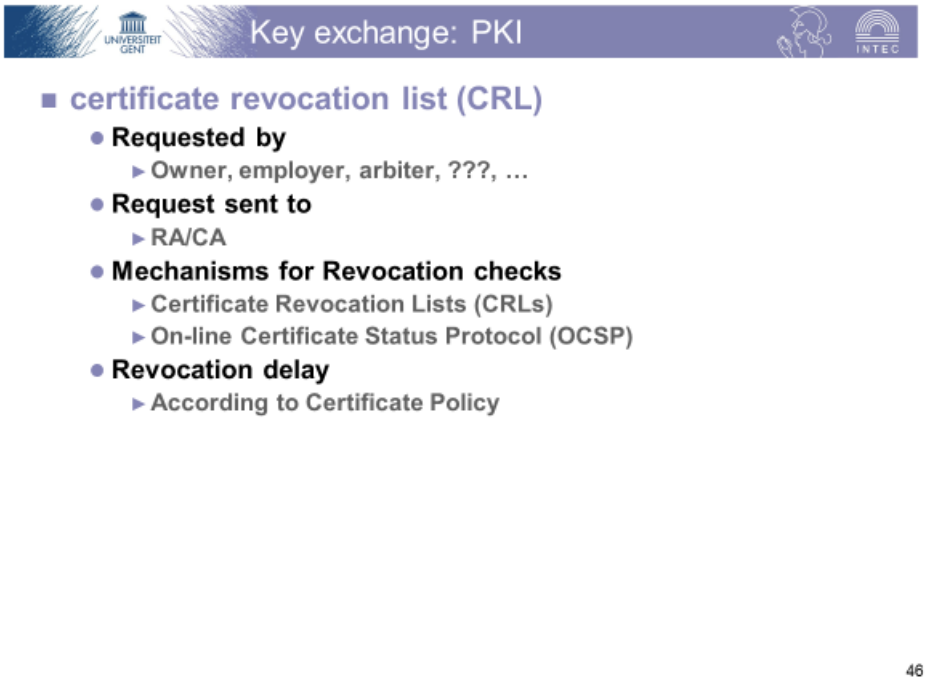
- **Cross-Certification**
  - **Mechanism:**
    - Certificates for CAs (not end-entities)
  - **Intra- vs. Inter- domain**
  - **One or two directions**
    - CA1 certifies CA2 and/or CA2 certifies CA1
  - **Control**
    - Cross-certificate limits trust
      - ✓ Name, policy, path length, etc. constraints

44

Key exchange: PKI

- **Certificate renewal**
  - **Same keys, same certificate, but new dates**
  - **Preferably automatic**
  - **but watch for attributes change!**
- **Certificate history**
  - **Key history**
    - For owner: e.g. to read old encrypted messages
  - **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

45



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.

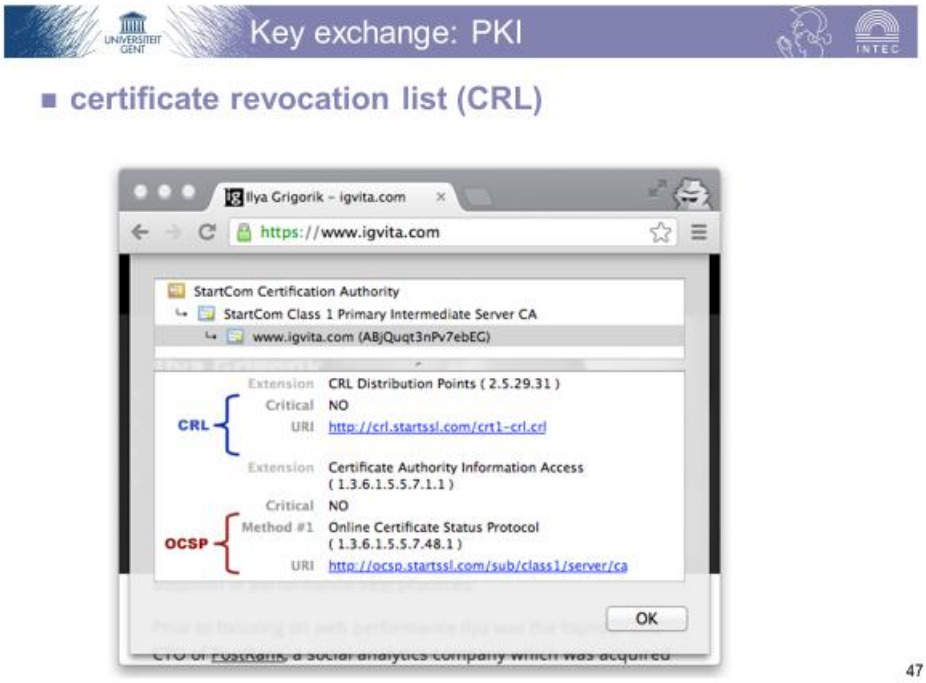
Online Certificate Status Protocol (OCSP)

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!



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
  - Key Distribution Centre (KDC)
  - Asymmetric encryption
    - Public Key Infrastructure (PKI)
    - **X.509 authentication**
- Secure networking protocols
- Firewalls

48

X.509 authentication

- X.509
  - **Part of X.500 ITU-T series of recommendations for "directory services"**
    - "directory" = set of servers maintaining a database with information about users
  - **Framework for providing authentication services by directory to users**
    - As a repository for public key certificates
    - Defines authentication protocols based on certificates

49

X.509 authentication

- X.509 certificates used in
  - TLS/SSL
  - S/MIME (Secure Multipurpose Internet Mail Extensions)
  - PGP
  - IPsec
  - SSH
  - Smart card
  - HTTPS
  - LDAP
  - XMPP
  - Microsoft Authenticode
  - ...

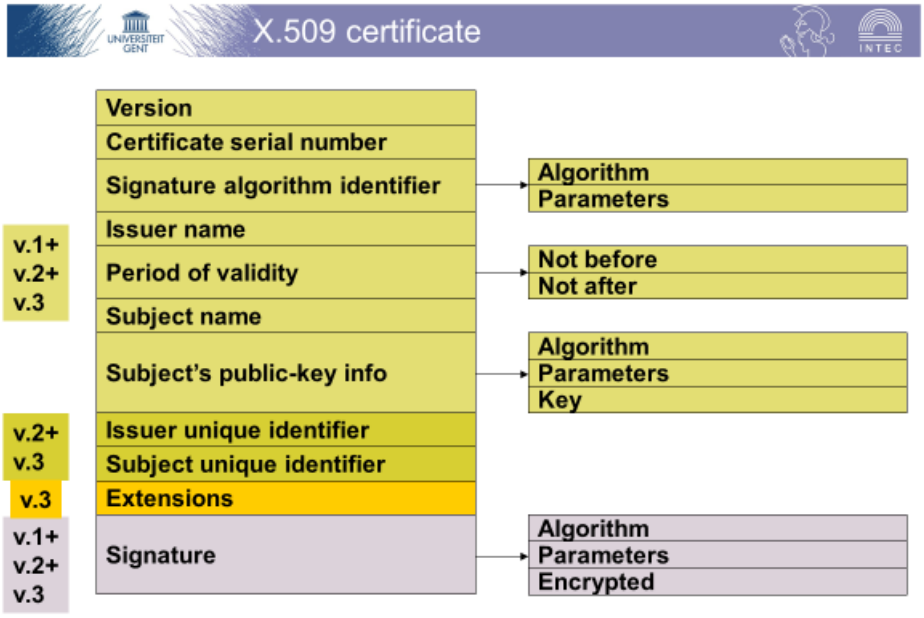
50

X.509 authentication

- X.509
  - Origin: 1988
  - By now version 3 from 1995 (revised in 2000)
  - Relies on:
    - Asymmetric cryptography (RSA recommended)
    - Digital signature (using hash function)
- Obtaining a certificate
  - Purchase a certificate from a certificate authority, such as VeriSign, Inc
  - Set up our own certificate service and have a certificate authority sign the certificates
  - Set up our own certificate service and do not have the certificates signed

51

X.509 was initially issued on July 3, 1988 and was begun in association with the X.500 standard. It assumes a strict hierarchical system of certificate authorities (CAs) for issuing the certificates. This contrasts with web of trust models, like PGP, where anyone (not just special CAs) may sign and thus attest to the validity of others' key certificates. Although most generally recognized certificates have to be paid for, several free options exist. For example, Let's Encrypt, a provider of free HTTPS certificates, gained trust in July 2018 from all the major browser makers — including Apple, Google, Microsoft and Mozilla. To date, the non-profit has issued more than 380 million certificates.



The "serial number" is different for each certificate issued by a single CA.

The "signature algorithm identifier" is in fact quite useless, as the algorithm is described again in the signature field.

The "issuer name" is the X.500 name of the CA that has issued the certificate.

The "period of validity" gives the time from which and the time until which the certificate is valid.

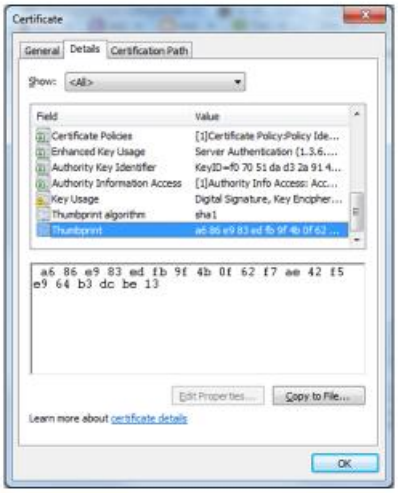
The "subject name" is the X.500 name of the user of the certificate. The certificate binds his name to the public key, which is described in the next field (identification of the algorithm and parameters used and the key itself).

The extensions from version 2 ("Issuer unique identifier" and "subject unique identifier") are rarely used.

The principle of the extensions from version 3 is discussed later on.

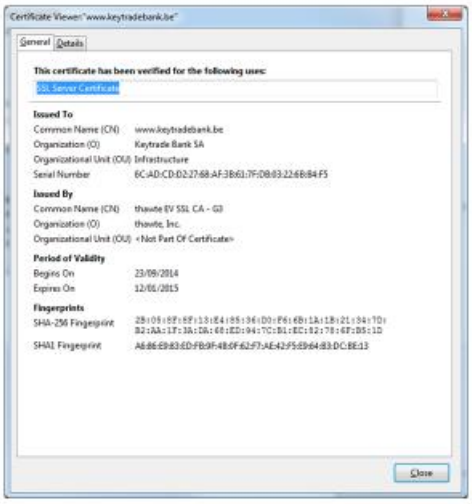
The digital signature covers (the hash value of) all other fields of the certificate using the CA private key. The field for the digital signature also contains the required information about the algorithm used. The signature itself is stored in the subfield "Encrypted".

X.509 certificate: illustration



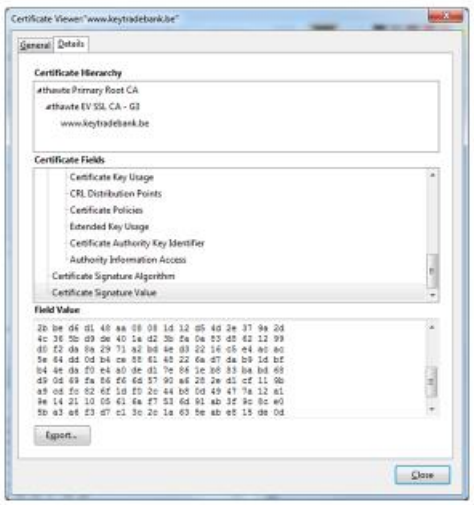
53

X.509 certificate: illustration



54

X.509 certificate: illustration

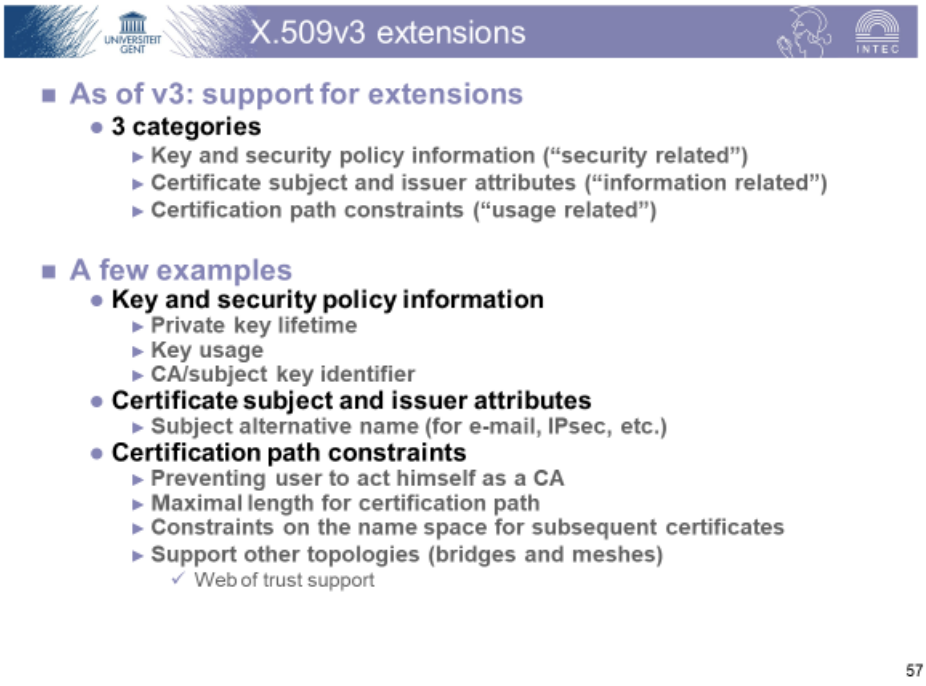


55

Limitations of X.509v1 certificates

- Limitations
  - **"Subject" field may be inadequate for determining user's identity (X.500 names often are rather short)**
  - **"Subject" field inadequate for applications identifying entities by e-mail address, URL, etc.**
  - **Need for information about security policy, e.g. for IPsec**
  - **Need for damage containment when CA is malicious or careless**
  - **Need for identification of different keys from single user for different uses**

56



The lifetime of a private key is typically shorter than the lifetime of the corresponding public key. Indeed, the private key is used to generate a digital signature, while the public key must also be used later on for the verification of this digital signature.

The key usage may be limited to certain applications: digital signature, key encryption, data encryption, etc.

The extension for the CA (or the subject) key identifier allows the CA (or the subject) to use different keys. This may be practical for the update of key pairs, or to use separate keys for encryption and for digital signatures.

The constraints on the name space within which a subject can issue new certificates himself may e.g. mean that he will only be able to issue certificates for units of his own company, but not to issue a certificate for a different company.

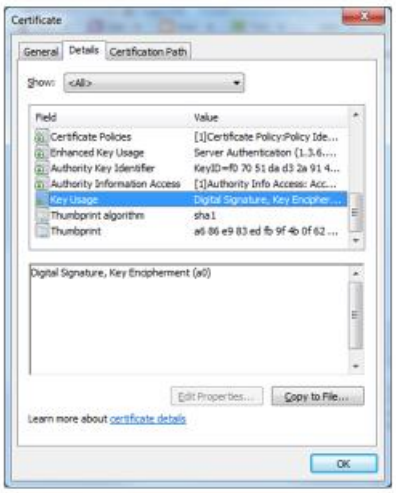
X.509v3 extensions

- **Extension structure**
  - **Any number of extensions can be supported**
    - ▸ **Future proof**
  - **Structure: type-and-value pair + optional critical flag**
    - ▸ **If an implementation doesn't recognise a critical extension, the certificate must be considered invalid**
    - ▸ **Actual definitions of critical flag usage are extremely vague**
      - ✓ X.509: Noncritical extension "is an advisory field and does not imply that usage of the key is restricted to the purpose indicated"
      - ✓ PKIX: "CA's are required to support constraint extensions", but "support" is never defined
      - ✓ S/MIME: Implementations should "correctly handle" certain extensions
      - ✓ MailTrusT: "non-critical extensions are informational only and may be ignored"
      - ✓ Verisign: "all persons shall process the extension... or else ignore the extension

58

Each extension has its own id, expressed as Object identifier, which is a set of values, together with either a critical or non-critical indication. In theory, a certificate-using system MUST reject the certificate if it encounters a critical extension that it does not recognize, or a critical extension that contains information that it cannot process. A non-critical extension MAY be ignored if it is not recognized, but MUST be processed if it is recognized.

X.509 certificate: illustration extensions



59

Example additional extension fields can be found below.

The **subject alternate name** fits everything that doesn't fit in a DN
• rfc822Name – email address, dave@wetaburgers.com
• dNSName – DNS name for a machine, ftp.wetaburgers.com
• uniformResourceIdentifier – URL, http://www.wetaburgers.com
• iPAddress – 202.197.22.1 (encoded as 0xCAC51601)
• x400Address, ediPartyName – X.400 and EDI information
• directoryName
• otherName – Type-and-value pairs (type=MPEG, value=MPEG-of-cat)
…
The **basic constraints extension** defines, amongst others:

- Whether the certificate is a CA certificate or not. This prevents users from acting as CAs and issuing their own certificates. This information is in fact redundant, since keyUsage (see later) specifies the same thing in a more precise manner. However, it is also included here because various trial-and-error proto-X.509v3 extension ideas have lingered on into current versions . As a result, there is much confusion over its use in non-CA certificates: German ISIS profile mandates its use, whereas the Italian profile forbids its use.
- Name Constraints. Constrain the DN subtree under which a CA can issue certificates. For example, a constraint of C=NZ, O=University of Auckland would enable a CA to issue certificates only for the University of Auckland, so that a CA can buy or license the right to issue certificates in a particular area. Constraints can also be applied to email addresses, DNS names, and URLs.

The **certificate policy** serve three functions:
• It provides a CA-specific mini-profile of X.509
• It defines the CA terms and conditions/indemnifies the CA
• It hides kludges for PKI problem areas
The referred to CA policy may define
• Obligations of the CA
– Check certificate user validity
– Publish certificates/revocations
• Obligations of the user
– Provide valid, accurate information
– Protect the private key
– Notify the CA on private key compromise
• List of applications for which the issued certificates may be used/may not be used
• CA liability (Warranties and disclaimers)
• Financial responsibility
• Certificate publication details (Access mechanism, Frequency of updates, Archiving)
• Compliance auditing (Frequency and type of audit, Scope of audit, ..)
• Security auditing (Which events are logged, Period for which logs are kept, How logs are protected)
• Confidentiality policy – What is/isn't considered confidential – Who has access – What will be disclosed to law enforcement/court

…
It is important to note that this extension defines/constrains what the CA does, not what the user does. X.509 delegates most issues of certificate semantics or trust to the CA's policy. Many policies serve mainly to protect the CA from liability. E.g. "Verisign disclaims any warranties… Verisign makes no representation that any CA or user to which it has issued a digital ID is in fact the person or organisation it claims to be… Verisign makes no assurances of the accuracy, authenticity, integrity, or reliability of information". Effectively these certificates have null semantics (if CAs didn't do this, their potential liability would be enormous).

Extended key usage.

Two interpretations exist as to what extended key usage values mean when set in a CA certificate

• Certificate can be used for the indicated usage – Interpretation used by PKIX, some vendors

• Certificate can issue certificates with the given usage – Interpretation used by Netscape, Microsoft, other vendors

These ambiguities remain even after many years.

Examples of key usage include

- Short-term authentication signature (not allowed for long term signatures such as certificates, e.g. for automatic signing of short term messages)
- Binding long-term signature (performed consciously)
- keyEncipherment (exchange of encrypted session keys, e.g. RSA)
- keyAgreement (DH)
- keyCertSign/cRLSign (signature bits used by CA's)
- …

X.509 certificates: limitations

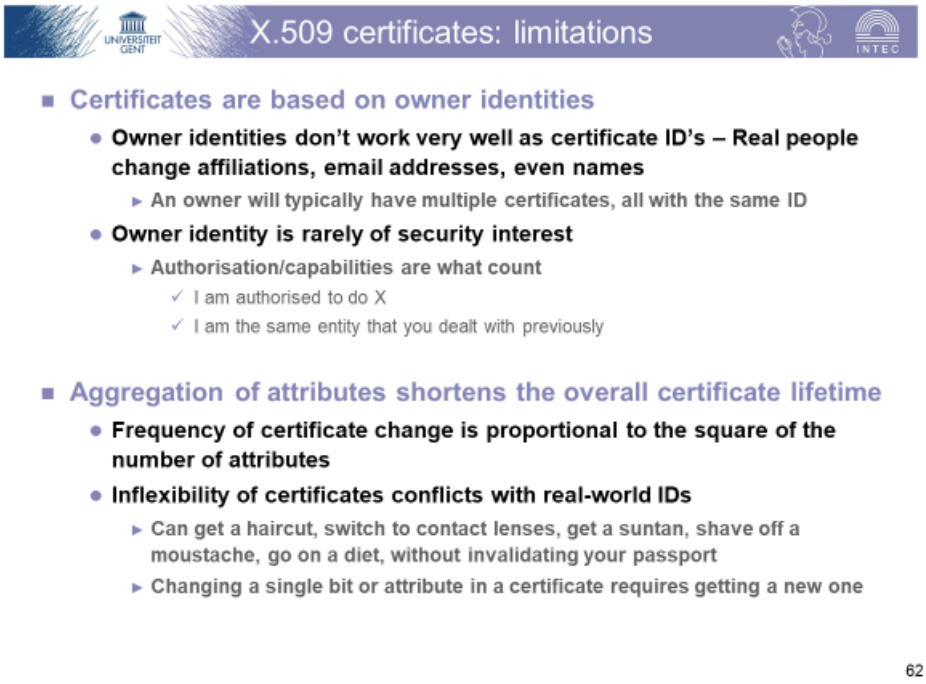
- X.509 is extremely vague and nonspecific in many areas
  - Extensions can be added over time
    - Most of the world has stopped caring about new PKI developments, so most of the new standards are being ignored by implementers
    - Creation of tens of "flavors" of certificate extension combinations
  - Slow progress on new extensions or fixes
- Revocation is not always handled well
  - Large time delays, not always checked (e.g. by browsers), etc.
  - Revocation should revoke the capability, not identities
    - Revoking a key requires revoking the identity of the owner
    - Renewal/replacement of identity certificates is nontrivial

60

X.509 certificates: limitations

- Authentication and confidentiality certificates are treated the same way for certification purposes
  - X.509v1 and v2 couldn't even distinguish between the two
- Certificates rapidly become a dossier as more attributes are added
  - Large overhead for a single mail exchange
- Software incompatibilities
  - Due to bugs, different extensions, different interpretations, different standards, …
  - Different interpretations and coding methods of fields
    - Unicode support? Negative numbers? Etc.
  - Ignoring extensions
  - Invalid extension combinations

61



For more shortcomings of X.509 certificates we refer to http://www.cypherpunks.to/~peter/T2a_X509_Certs.pdf

Also, some providers of security solutions (Microsoft, S/MIME, ..) provide a compatibility checking service

"You can't be a real country unless you have a beer and an airline. It helps if you have some kind of a football team, or some nuclear weapons, but at the very least you need a beer."
— Frank Zappa

"And an X.509 profile."
— Peter Gutmann

64

For the e-ID implementation, Belgium follows the ISO 7816 standard which included definitions on which certificates to use.

- Explain the benefits of ephemeral DH over traditional DH.
- What is the difference between a Key Distribution Centre (KDC) and Public Key Infrastructure (PKI)? What are the advantages / disadvantages of both approaches?
- List 5 reasons why a certificate might have to be revoked. How can this revocation be implemented?
- Give example restrictions that can be part of a certificate. Why are these relevant?

65