Key Distribution

Tecnologie Internet

a.a. 2022/2023

Summary

- Symmetric Key Distribution Using Symmetric Encryption
- Symmetric Key Distribution Using Asymmetric Encryption
- Public Key Distribution
- X.509 Certificates
- Public Key Infrastructure (PKI)

Symmetric Key Distribution Using Symmetric Encryption

For symmetric encryption to work, the two parties must share the **same key**, and that key must be protected from access by others.

For two parties A and B, key distribution can be achieved in a number of ways, as follows:

1. A can select a key and *physically* deliver it to B.

2. Another party can select the key and *physically* deliver it to A and B.

3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.

4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.

Symmetric Key Distribution Using Symmetric Encryption

Options 1 and 2 call for manual delivery of a key. This is not always feasible.

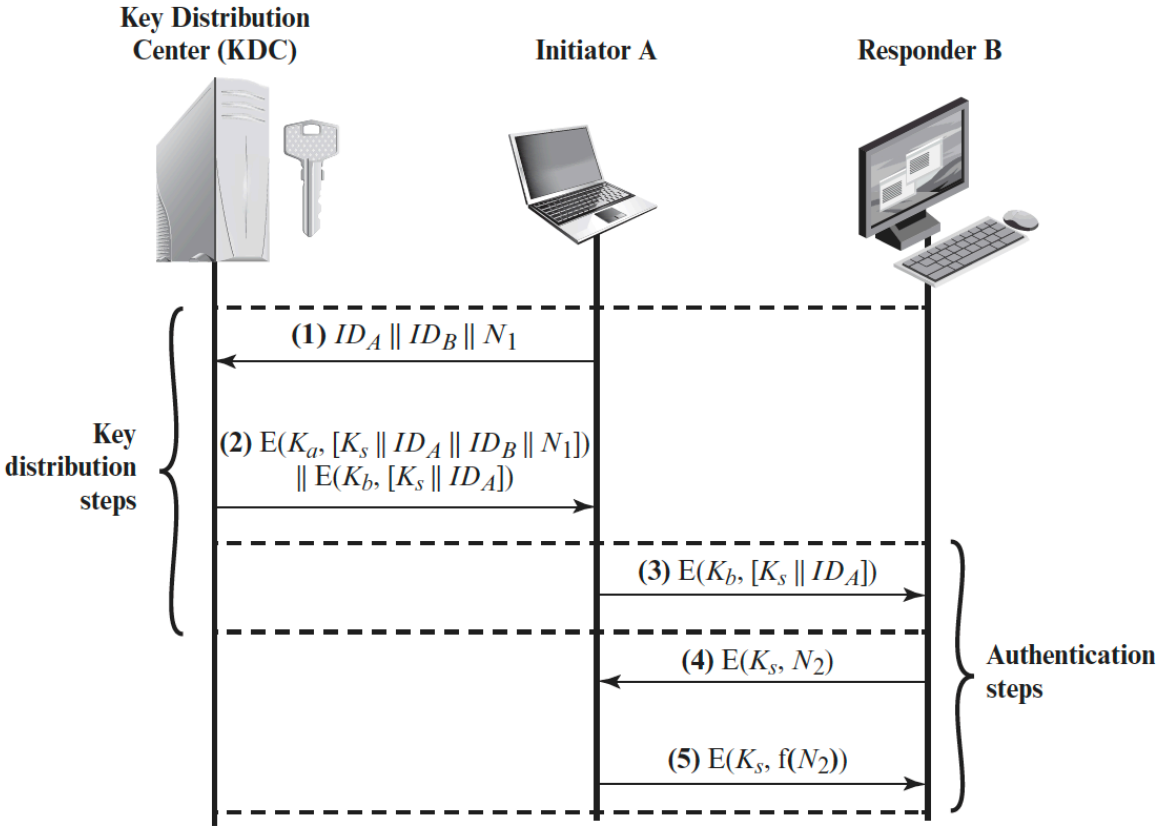
Option 3 is a possibility, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys will be revealed.

For the end-to-end encryption, some variation on option 4 has been widely adopted:

- **Key Distribution Center (KDC)**
- **hierarchy of keys**, at least two levels:
  - persistent **master key** shared by the KDC and the users
  - temporary **session key** for each pair of users

Key Distribution Scenario

Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection.



Key Distribution Scenario

1. A issues a request to the KDC, including: identity of A and B, and a unique identifier N_1 (**nonce**).

2. The KDC responds with a message encrypted using K_a, i.e., the master key shared with A. The message includes the session key K_s and the original request message (to allow A for matching the response with the appropriate request). Moreover, KDC transmits a message encrypted using K_b, i.e., the master key shared with B. Such a message includes the session key K_s and the identifier of A.

3. A forwards to B the information originated at the KDC for B. Being encrypted with K_b, this information is protected from eavesdropping.

Now B knows the session key K_s and the identity of the other party.

Key Distribution Scenario

After step 3, A and B may start communicating privately, using the session key. However, two additional steps are desirable:

4. Using the session key K_s, B sends a nonce N_2 to A.

5. Using the session key K_s, A responds with f(N_2), where f() is a function that performs some transformation on N_2 that is known to B (e.g., adding one).

These last steps assure B that the original message it received (step 3) was not a replay sent by an attacker that is not A and does not own K_s.

Hierarchical Key Control

For very large networks, it may not be practical to limit the key distribution function to a single KDC.

A hierarchy of KDCs can be established:

- **local KDC** responsible for a small domain (single LAN, or single building)
- **global KDC** used by local KDCs when a pair of users desire a shared session key

The hierarchical concept can be extended to three or even more layers, depending on the size of the user population and the geographic scope of the internetwork.

Session Key Lifetime

Competing considerations:

- the more frequently the session keys are changed, the more secure they are
- session key distribution delays the start of the communication

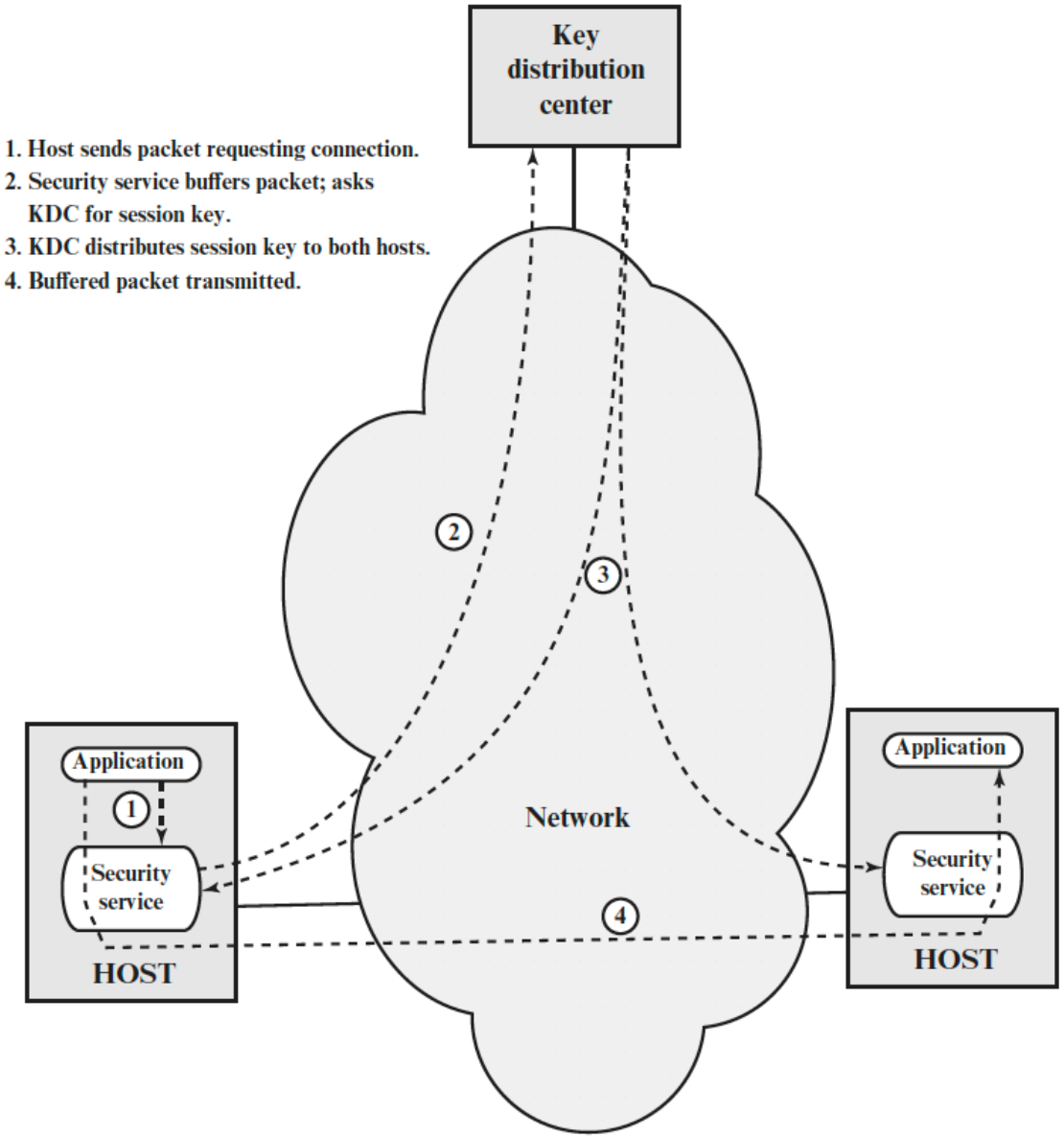
For **connection-oriented protocols**: use the same session key for the length of time the connection is open. If the connection has a very long lifetime, it would be prudent to change the session key periodically.

For a **connectionless protocol**, such as a transaction-oriented protocol, there is no explicit connection initiation or termination. Thus, it is not obvious how often one needs to change the session key. Best strategy: use a given session key for a certain fixed period only or for a certain number of transactions.

Transparent Key Control

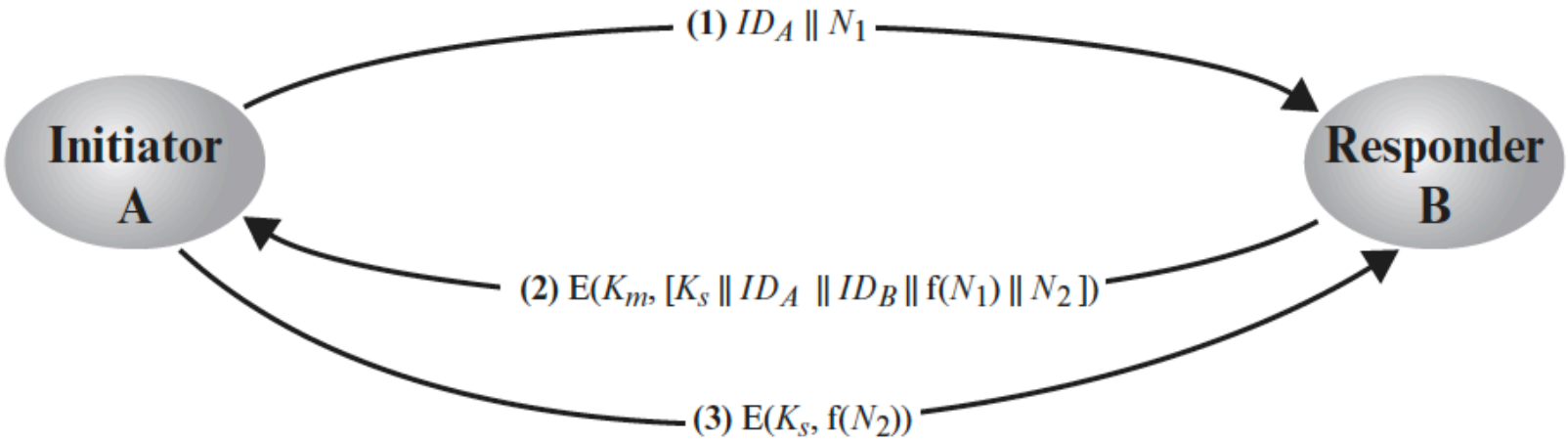
Assumption: connection-oriented end-to-end protocol, such as TCP.

The **security service** module behaves transparently with respect to the application.
It may consist of functionality of the transport or network layer.



Decentralized Key Control

Although full decentralization is not practical for larger networks using symmetric encryption only, it may be useful within a local context.



1. A issues a request for a session key, including a nonce N_1.
2. B responds with a message that is encrypted with the master key. The response includes the session key K_s, $f(N_1)$, and N_2.
3. Using the new session key K_s, A returns $f(N_2)$ to B.

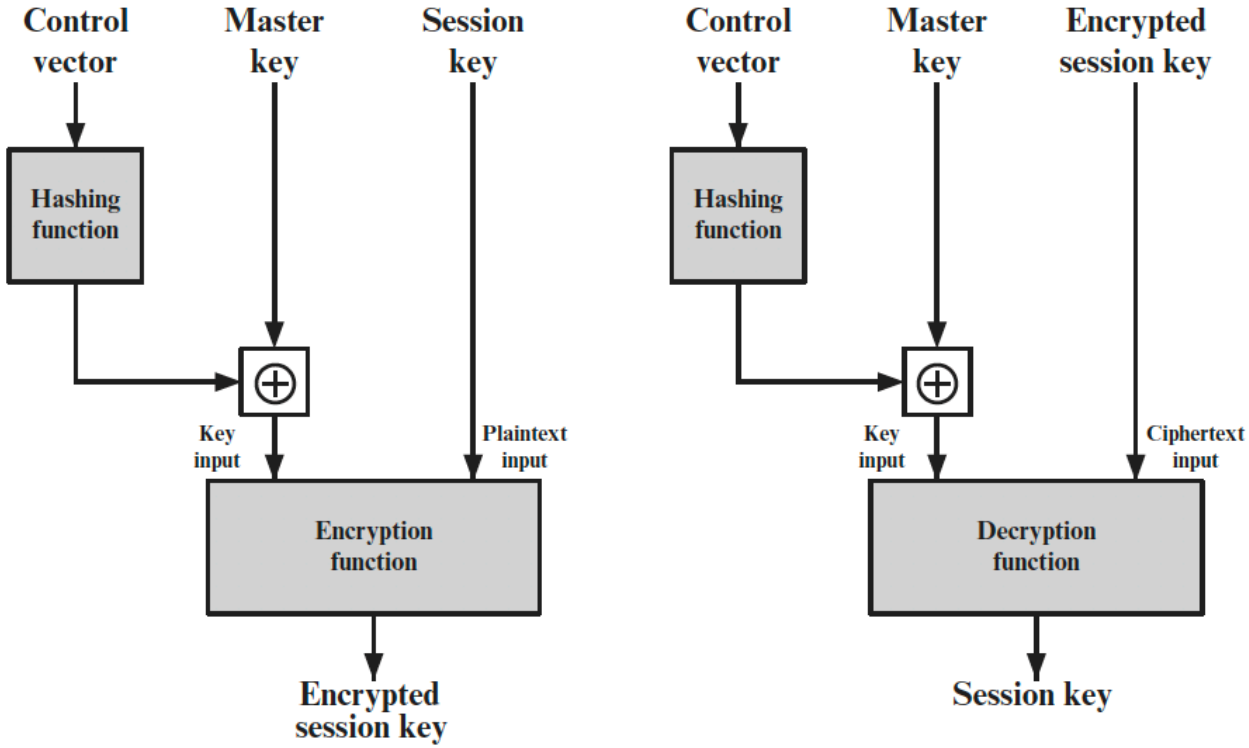
Session Keys with Control Vector

In addition to separating master keys from session keys, we may wish to **define different types of session keys** on the basis of use, such as

- Data-encrypting key, for general communication across a network
- PIN-encrypting key, for personal identification numbers (PINs) used in electronic funds transfer and point-of-sale applications
- File-encrypting key, for encrypting files stored in publicly accessible locations

Session Keys with Control Vector

In the control vector scheme, each session key has an associated control vector consisting of a number of fields that specify the uses and restrictions for that session key. The length of the control vector may vary.



Session Keys with Control Vector

Encryption:

- Hash value = h = H(CV)
- Key input = $K_m \oplus h$
- Ciphertext = E([$K_m \oplus h$], K_s)

When the encrypted session key is delivered to a user from the KDC, it is accompanied by the CV in clear form. In this way, the recipient can compute h and proceed with the decryption of the session key.

Decryption: D([$K_m \oplus h$], E([$K_m \oplus h$], K_s))

Advantages over use of simple tags:

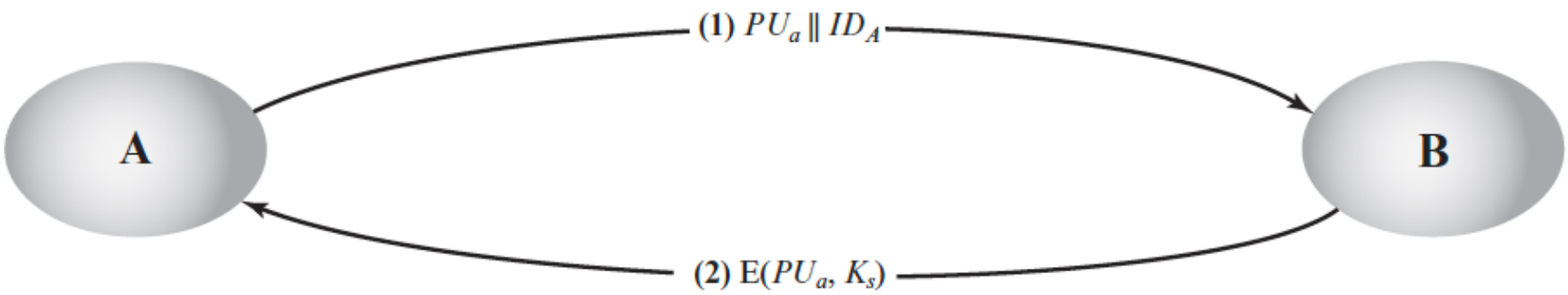
- no restriction on length of the control vector
- the control vector is always available in clear form

Symmetric Key Distribution Using Asymmetric Encryption

Because of the inefficiency of public-key cryptosystems, they are almost never used for the direct encryption of sizable blocks of data, but are limited to relatively small blocks.

One of the most important uses of a public-key cryptosystem is to **encrypt secret keys for distribution**.

Simple scheme proposed by Merkle:



Most used scheme: **Diffie-Hellman**.

Public Key Distribution

Both Merkle's and Diffie-Hellman schemes are insecure against man-in-the-middle attacks, unless a mechanism for trusting public keys is introduced.

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

- **Public announcement**
- **Publicly available directory**
- **Public-key authority**
- **Public-key certificates**

Public Announcement of Public Keys

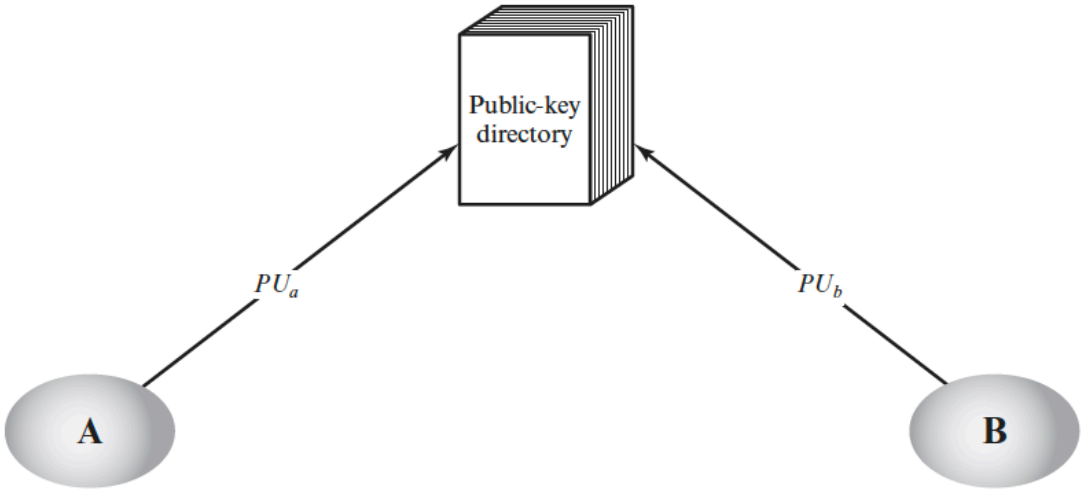
Any participant can send his or her public key to any other participant or broadcast the key to the community at large.

Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key.

Publicly Available Directory

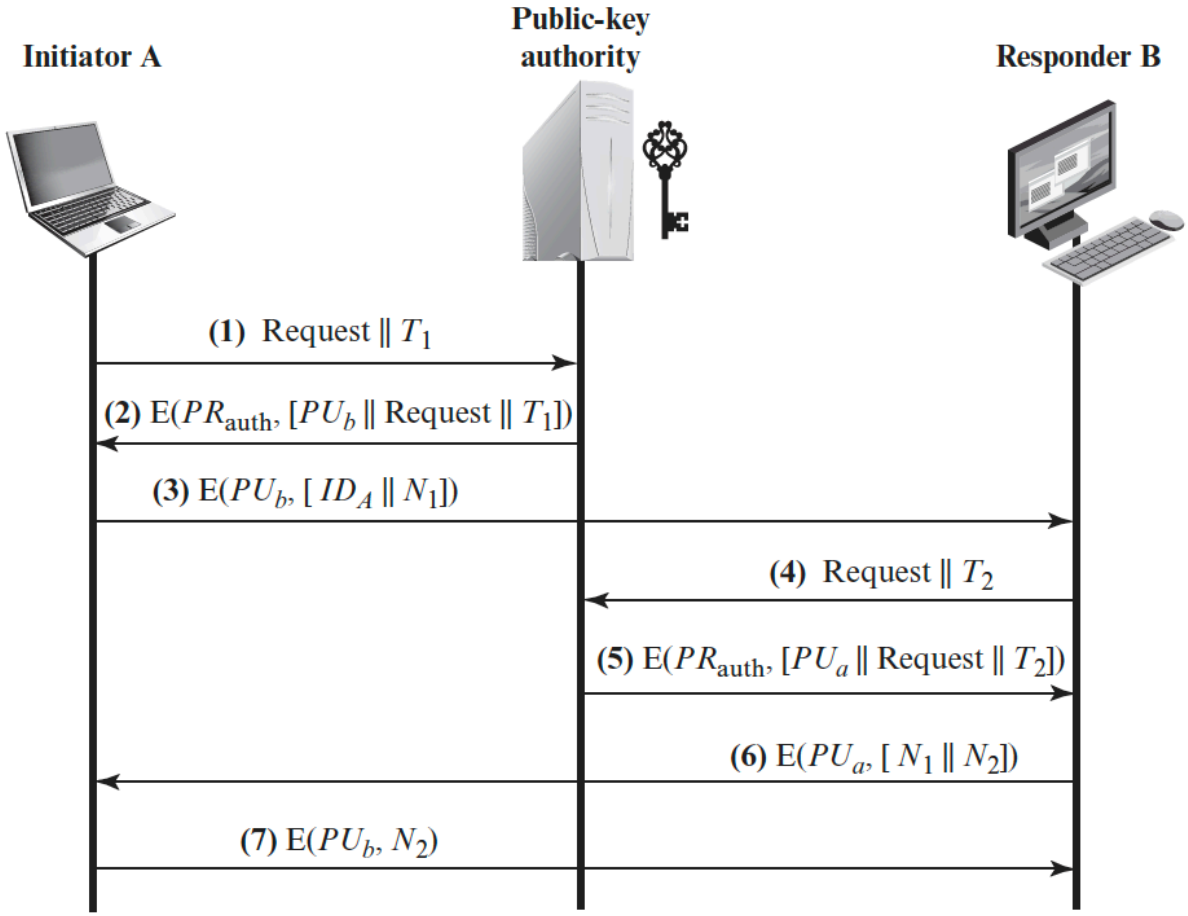
A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some **trusted entity or organization**.

1. Directory with a [name,public key] entry for each participant
2. Authenticated communication **from user to directory**
3. Possibility to replace an existing key with a new one at any time



Public-Key Authority

An even stronger solution, where **also the directory owner is authenticated**:



Public-Key Authority

1. A sends a timestamped message to the public-key authority, containing a request for the current public key of B
2. The authority responds with a message encrypted with PR_{auth}, so A is assured that the message originated with the authority. The message includes PU_B and the original request with its timestamp
3. A stores PU_B and uses it to encrypt a message to B containing ID_A and a nonce N_1, used to identify this transaction uniquely
4. B sends a timestamped message to the public-key authority, containing a request for the current public key of A
5. The authority sends PU_A to B, in a message encrypted with PR_{auth}
6. B sends a message to A encrypted with PU_A, containing N_1 and another nonce N_2; the presence of N_1 assures A that the correspondent is B
7. A returns N_2 encrypted with PU_B, to assure B that its correspondent is A

Public-Key Certificates

The public-key authority could be a bottleneck in the system.

An alternative, more efficient, approach is to use **certificates**. In essence, a certificate consists of a public key, an identifier of the key owner, and the whole block signed by a **trusted third party**.

Typically, the third party is a **certificate authority**, such as a government agency or a financial institution, that is trusted by the user community.

A user can present his or her public key to the authority in a secure manner and obtain a certificate.

Then, the user can convey its key information to another by transmitting its certificate.

Public-Key Certificates

A certificate scheme has the following requirements:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.

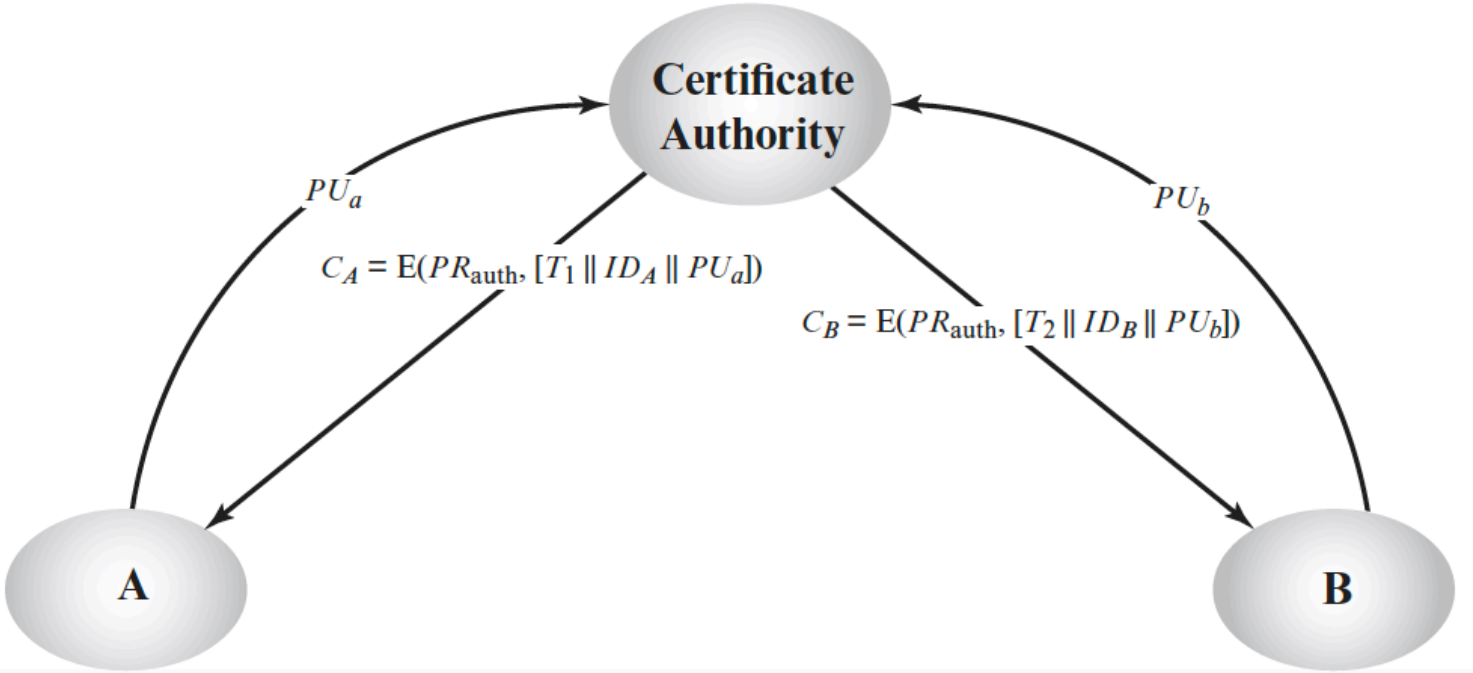
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.

3. Only the certificate authority can create and update certificates.

4. Any participant can verify the **time validity** of the certificate.

Public-Key Certificates

Certificate scheme:



D(PU_{auth}, C_{A}) = (T_1 || ID_{A} || PU_{A}), where T_1 is the timestamp

Public-Key Certificates

Certificates are usually deployed in:

- Web transactions
  - HTTPS uses SSL/TLS
- Virtual Private Networks
  - IPsec uses IKE
- Secure messaging
  - S/MIME and PGP
- Anywhere strong authentication and/or encryption is required

Public-Key Certificates

The owner of the public key (and of the certificate) could be:

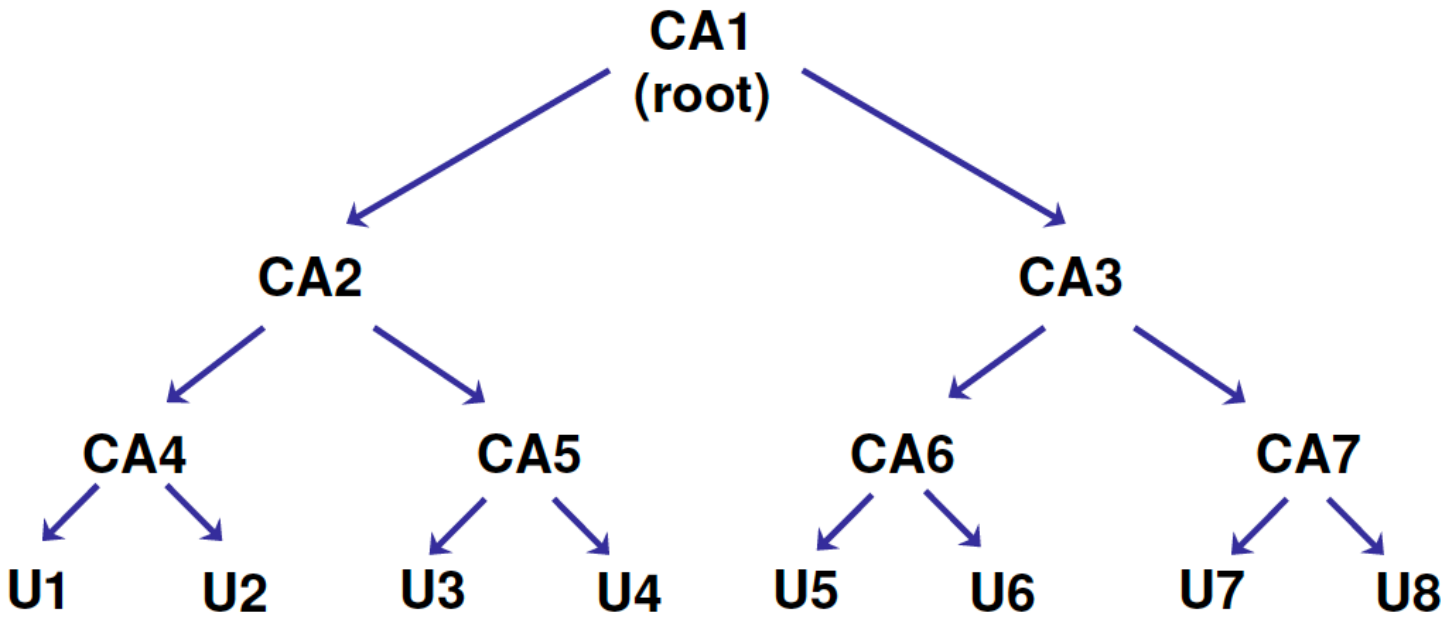
- a person
- an alias of a person
- an organization
- a role within an organization
- a hardware system
- a software system

Some certificates are specific (and valid only) for a subset of these entities.

Public-Key Certificates

Certificate Authorities (CAs) can be certified by other upper level CAs.

CAs may be organized in a hierarchical (trust) structure.



Public-Key Certificates

Now suppose that A has obtained a certificate from certification authority CA_1 and B has obtained a certificate from CA_2.

If A does not securely know the public key of CA_2, then B's certificate, issued by CA_2, is useless to A. A can read B's certificate, but A cannot verify the signature. However, if there is a certificate of CA_2 signed by CA_1, the following procedure will enable A to obtain B's public key.

1) A obtains the certificate of CA_2 signed by CA_1. Because A securely knows CA_1's public key, A can obtain CA_2's public key from its certificate and verify it by means of CA_1's signature on the certificate.
2) A then obtains the certificate of B signed by CA_2. Because A now has a trusted copy of CA_2's public key, A can verify the signature and securely obtain B's public key.

X.509 certificates

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a **directory service**.

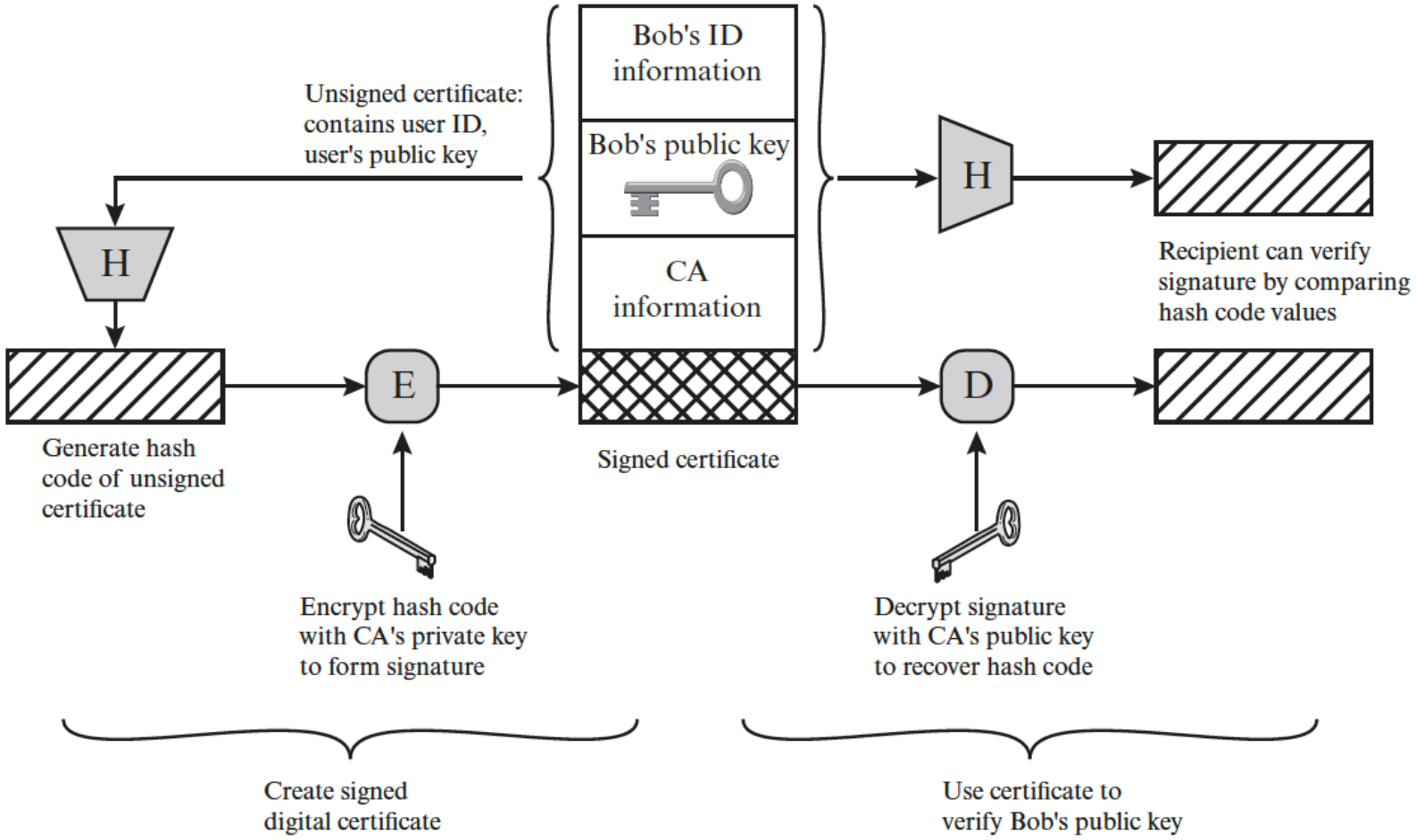
X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates.

Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority.

The X.509 certificate format is used in S/MIME, IPsec, SSL/TLS, and more.

The standard is currently at version 9, issued in 2019.

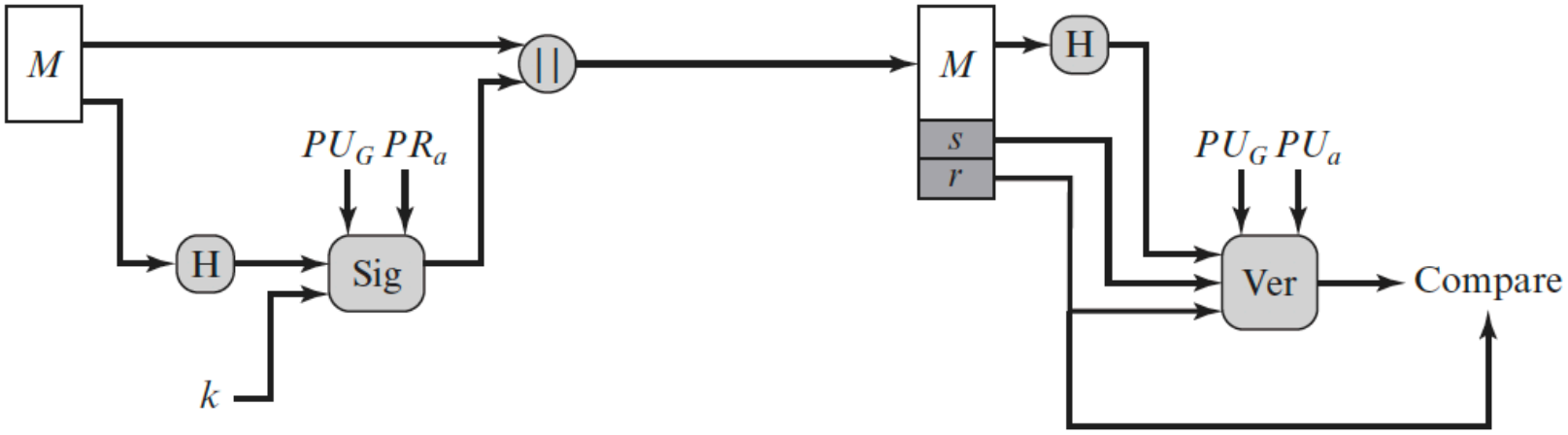
X.509 certificates



X.509 certificates

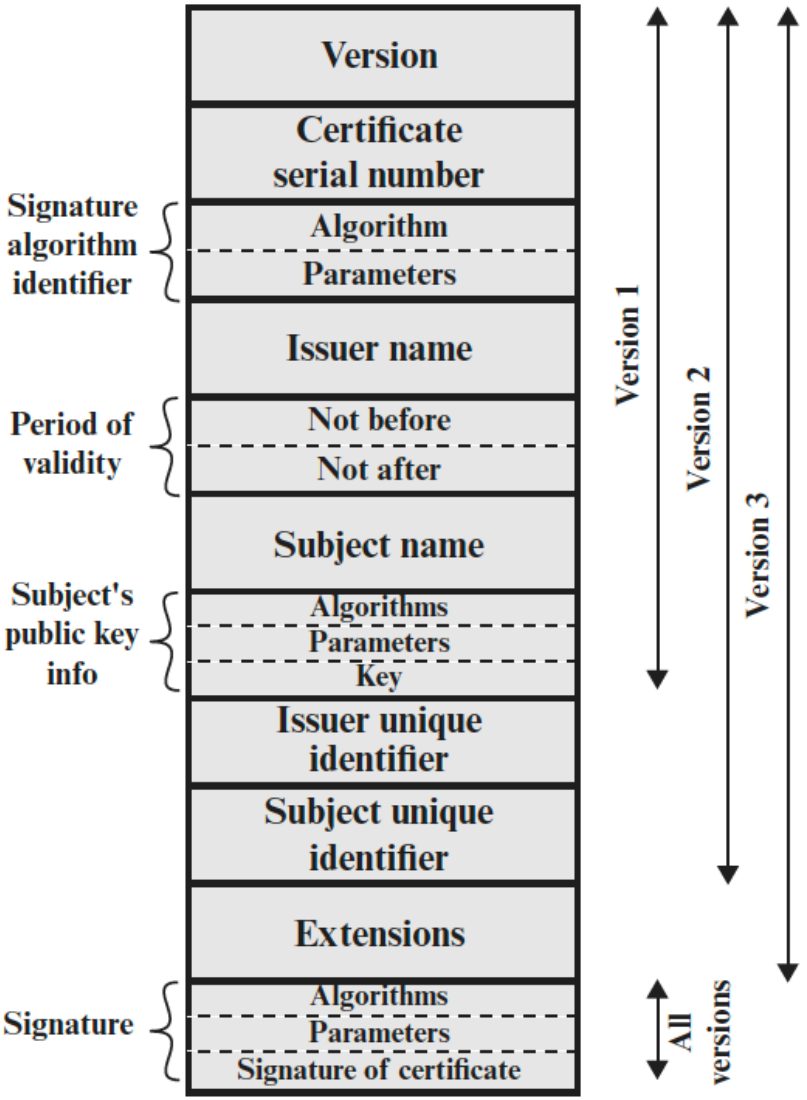
The current version of X.509 does not dictate a specific digital signature algorithm.

If the DSA scheme is used, then the hash value is not encrypted but serves as input to a digital signature generation algorithm.



X.509 certificates

X.509 formats:



X.509 certificates

Standard notation to define a certificate:

CA <<A>> = CA {V, SN, AI, CA, UCA, A, UA, Ap, T^A}

where

CA <<A>> = the certificate of user A issued by CA
V = version
SN = serial number
AI = identifier of the algorithm used to sign the certificate
CA = name of the certificate authority
UCA = optional unique identifier of the certificate authority
A = name of the user
UA = optional unique identifier of the user
Ap = public key of the user
T^A = period of validity of the certificate

X.509 certificates

Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to **revoke** a certificate before it expires, for one of the following reasons.

1. The user's private key is assumed to be compromised.

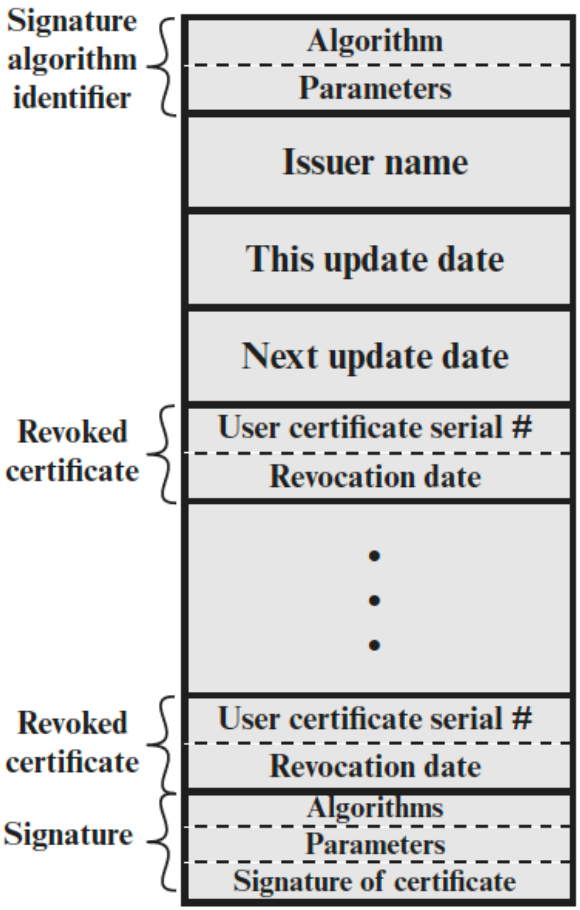
2. The user is no longer certified by this CA. Reasons for this include that the subject's name has changed, the certificate is superseded, or the certificate was not issued in conformance with the CA's policies.

3. The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA. This list should also be posted on the directory.

X.509 certificates

Certificate Revocation List (CRL):



Public Key Infrastructure (PKI)

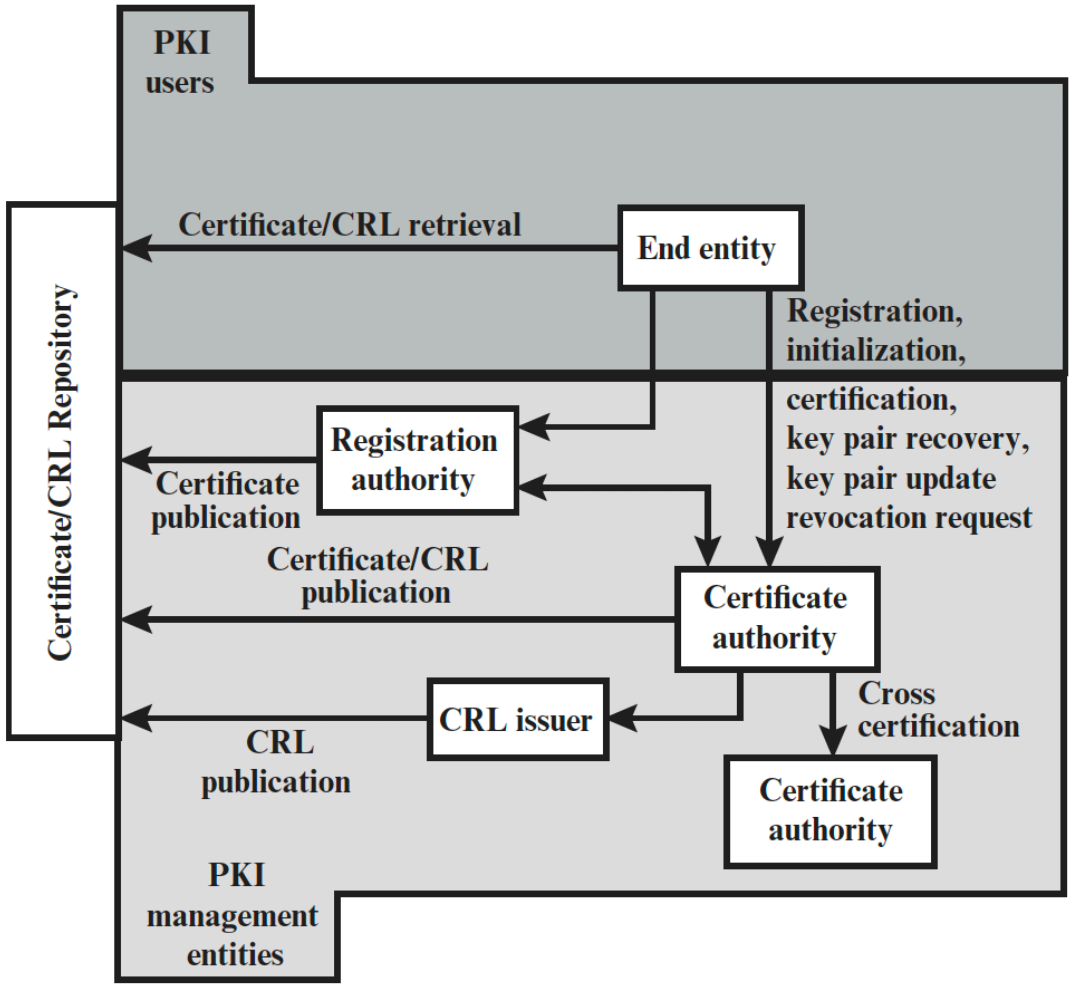
RFC 4949 (Internet Security Glossary):

A public-key infrastructure (PKI) is the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.

The **IETF Public Key Infrastructure X.509 (PKIX)** working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet.

Public Key Infrastructure (PKI)

PKIX architectural model:



References

William Stallings, ***Cryptography and Network Security - Principles and Practice***, 7th edition, Pearson 2017