Public keys distribution and Public Key Infrastructure

Luca Ferretti

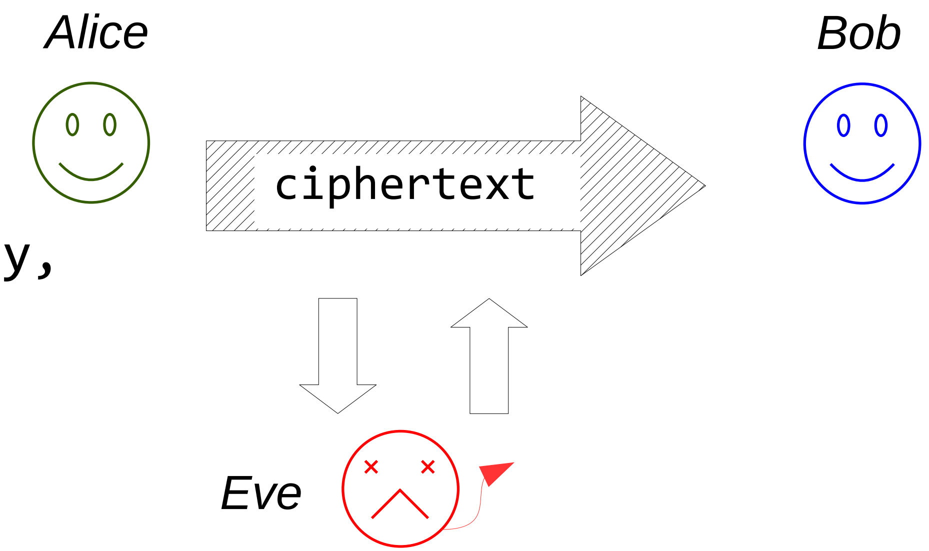
Protocolli e Architetture di Rete

Università degli Studi di Modena e Reggio Emilia

Laurea Informatica A.A. 2024/2025

Key distribution

```
Encrypt(
    encryption_key,
    plaintext
) → ciphertext
```



Alice

Bob

ciphertext

Eve

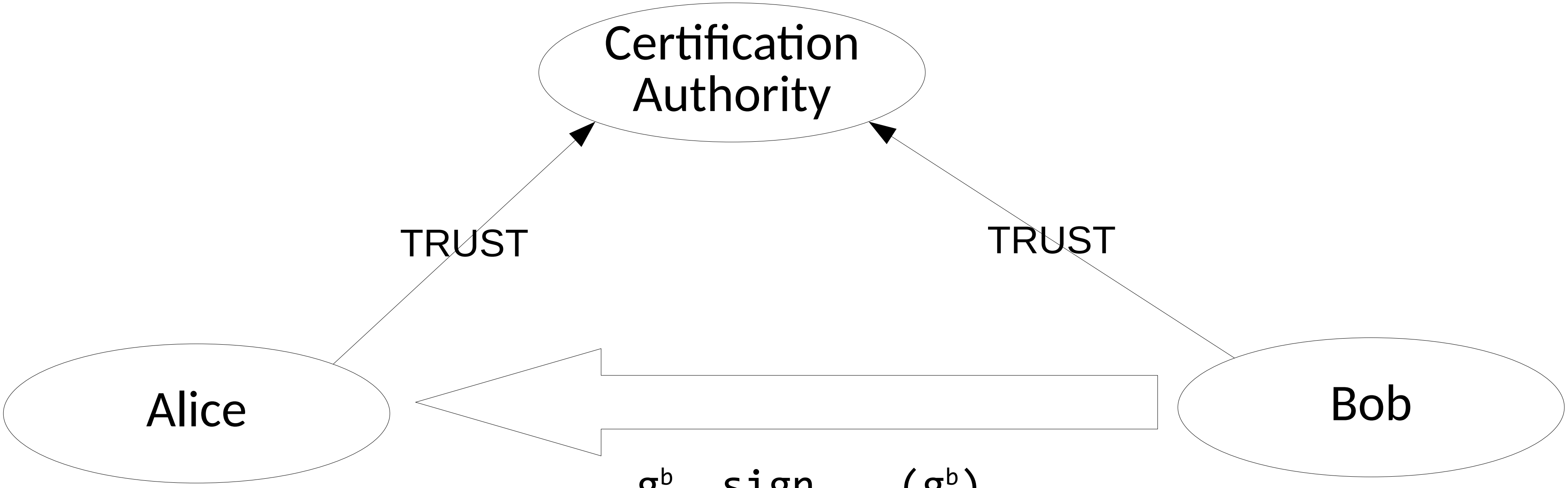
```
Decrypt(
    decryption_key,
    ciphertext
) → plaintext
```

- Cryptographic schemes need **keys**
  - Symmetric schemes need to the same key
  - We can distributed symmetric keys by knowing asymmetric schemes
  - Asymmetric schemes need to distribute public keys

→ **We need to study how to distribute public keys**

Trusted third party [abstract]

- Hi Alice, I’m Bob
  - here is my Public key
  - here is the ***certificate*** where our **mutual friend Carl** confirms this information: **the certificate is signed by Carl**
- The trusted third party in PKI is the ***Certification Authority***



g^b, $\mathrm{sign}_{bob\text{-}sk}(g^b)$,

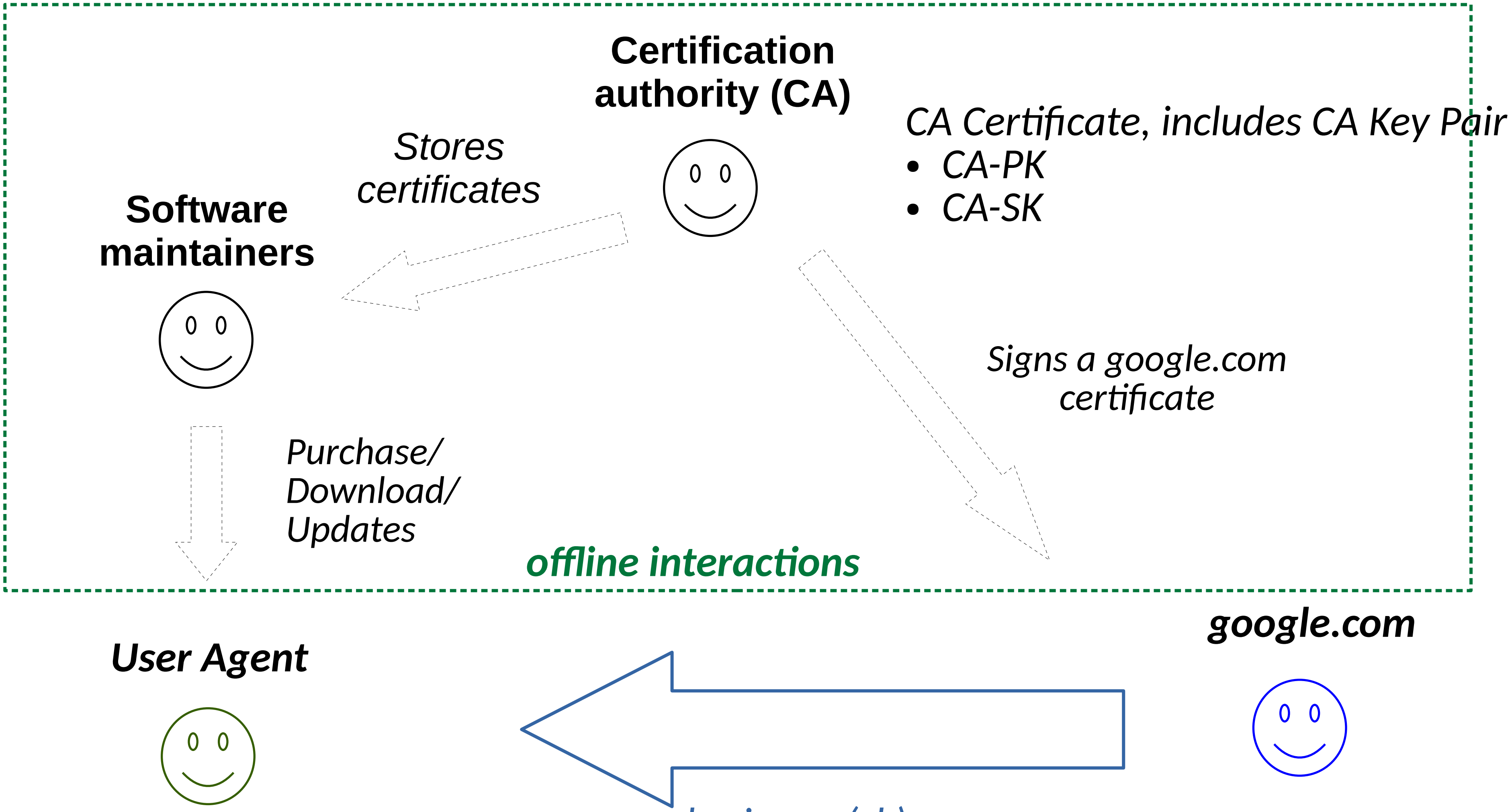
$\boxed{\mathbf{signature}_{CA}\mathbf{(bob\text{-}pk)}}$

The signature was provided to Bob by the CA

x509 Certificate

- Standard for binding metadata to cryptographic material in PKI architectures
- Identifies entities by using ***the distinguished name (DN)***
  - *a composite information obtained from multiple fields*
- The certificate binds the ***public key of the entity*** to the ***DN***
- The certificate includes other ***mandatory information***, such as:
  - format version (currently v3)
  - ***issuer*** (the CA that released this certificate)
  - **unique serial number** that identifies the certificate
  - ***validity period***
    - ***start date ("Not before")***
    - ***expire date ("Not after")***
  - certificate **type** (server, client, email)
  - information about crypto protocols (hash, signature)

Certification authorities, messages flow



Certification authority (CA)

CA Certificate, includes CA Key Pair
- *CA-PK*
- *CA-SK*

Stores certificates

Software maintainers

Signs a google.com certificate

Purchase/ Download/ Updates

offline interactions

google.com

User Agent

sk, $sign_{G\text{-}SK}(sk)$

google.com certificate

Look for the CA certificate locally, verifies the signature included in the “google.com” certificate by using CA-PK

google.com key pair
- G-SK, G-PK

google.com certificate
- **signature_{CA}(G-PK)**

Certification authority

- The **Certification Authority (CA)** releases certificates that bind the **public key** to an **entity**
  - persons
  - role
  - organizations
  - devices
- Entities might include ***identification information***
  - common name, country, state, city, …
- The certificates include additional metadata and information

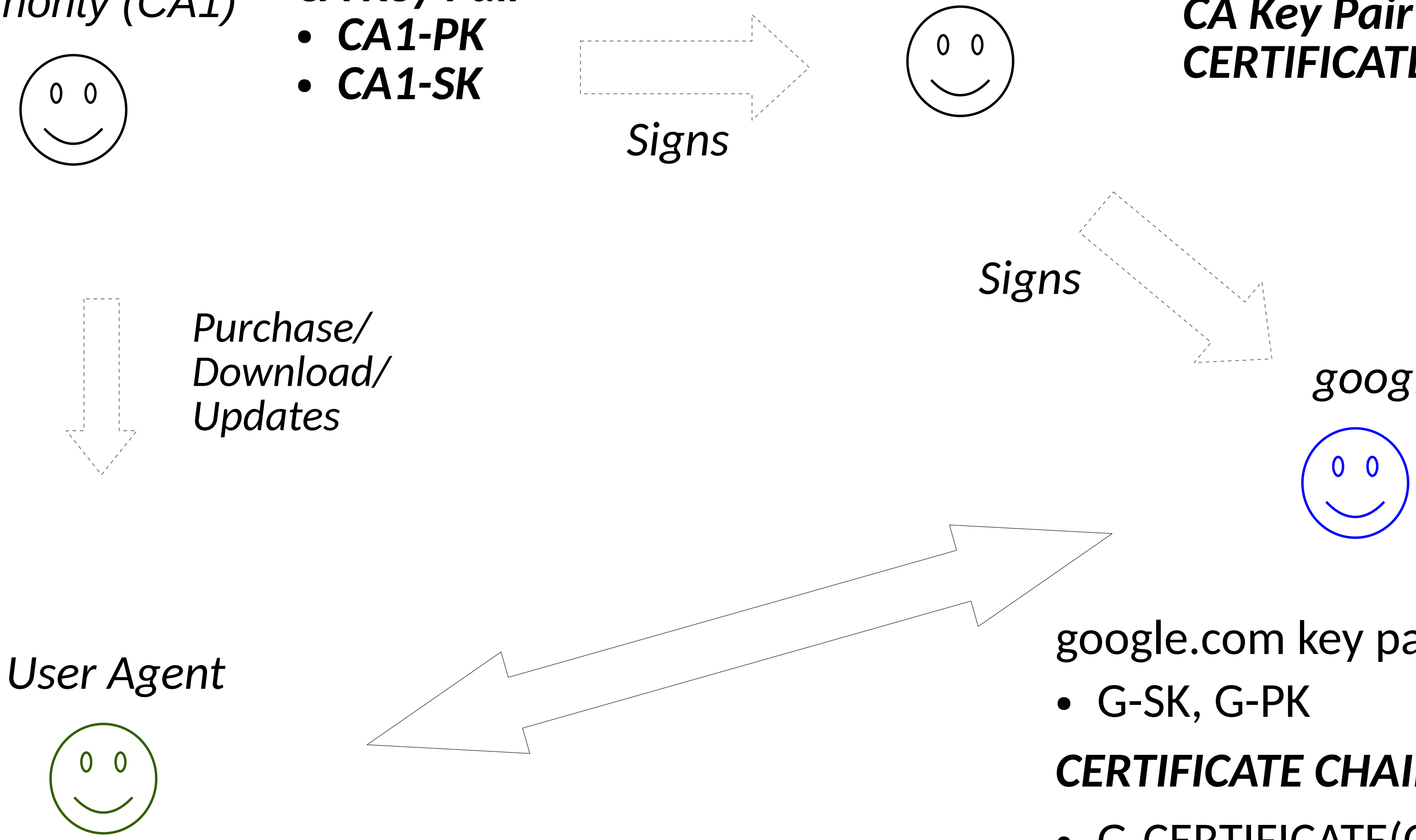
Delegating certificates

- Requiring a few authorities to sign all certificates is not scalable
  - point of failures
  - political and economic conflicts
  - complex configurations

- A **hierarchical** approach is a viable trade-off
  - the root CA certificates a CA that certificates a CA that....
  - PKI trust model

- A few CAs PK included in the sofware allow to verify a **huge amount of certificates**

Real World: Hierarchical Certification Authorities



Root CA

Certification authority (CA1)

CA Key Pair

- ***CA1-PK***
- ***CA1-SK***

Signs

Intermediate CA

Certification authority (CA2)

CA Key Pair → CA1-PK, CA1-SK
CERTIFICATE(CA1-SK, CA2-PK)

Signs

Purchase/ Download/ Updates

google.com

User Agent

User Agent or OS include CA1-PK

google.com key pair

- G-SK, G-PK

CERTIFICATE CHAIN

- G-CERTIFICATE(CA2-SK, G-PK)
- CA2-CERTIFICATE(CA1-SK, CA2-PK)
- CA1-PK

Certificate chain

- The server has a certificate, issued by an ***intermediate CA***
  - The intermediate CA has a certificate, issued by ***another intermediate CA*** or a ***root CA***
  - ***Root CAs*** are known and installed in operating systems and Web browsers with regard to governance policies
    - https://wiki.mozilla.org/CA
- To verify the end-user certificate, a client needs to verify all certificates in the chain, until it finds a known trusted certificate
- A server may return the full certificate chain
  - or assume that the client knows many “famous” intermediate servers and only return the end certificates
    - Web browsers often store more certificates then the OS

Validating the x509 certificate

- Validating the certificate requires to
  - ***verify the signature*** of the issuer
  - ***verify all metadata accordingly to the application***
    - entity names (is it really the certificates for google.com?)
    - validity (when is the expiration date?)
    - type (I'm contacting a server: is this a valid server certificate?)

- Validating all certificates in the certificate chain

- Verifying that the certificate is not included in the CA certificate revocation list

Certificate, examples

- It is possible to read existing Web sites certificates
  - by using the **browser** (https, high-level information)
  - by using **openssl** (any SSL/TLS connections, low level data)

```
openssl s_client -host <host> -port <port>
```

```
openssl s_client -host google.it -port 443

CONNECTED(00000003)
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify return:1
depth=1 C = US, O = Google Inc, CN = Google Internet Authority G2
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O = Google Inc, CN = google.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=google.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
```

Certificate, examples

- ***For Web certificates, the common name (CN) or an alternative name (Subject Alt Names) must match the FQDN***
  - *try to access an https Web site with a different hostname (e.g., by modifying the hosts file)*

- If we want to read all the details of a certificate

```
openssl x509 -noout -text -in <certificate>
```

Self-signed certificates

- **Certification authorities are trusted third parties**

- If we want to configure a private server for internal usage, we could avoid them and use ***self-signed certificates***
  - not signed by any other CA
  - can be used by all tools and applications that support PKI
  - must be explicitly accepted by users or added to the client system

Self-signed certificates – OpenSSL example

- As an example, it is possible to create a self-signed certificate by using the command:

```
openssl req -newkey rsa:2048 -nodes \
            -x509 -days 365 \
            -keyout <SK-FILE> -out <CRT-FILE>
```

- Then, we can test our secret key and our certificate in any Web server
- OpenSSL also provides a simple debug server

```
openssl s_server -cert <CRT-FILE> -key <SK-FILE> \
                 -port 4433


openssl s_client -connect 127.0.0.1:4433 \
                 -Cafile <CRT-FILE> \
                 -verify_hostname <FQDN>
```