Aalto University

Public Key Infrastructure & TLS

Tuomas Aura
CS-C3130 Information security

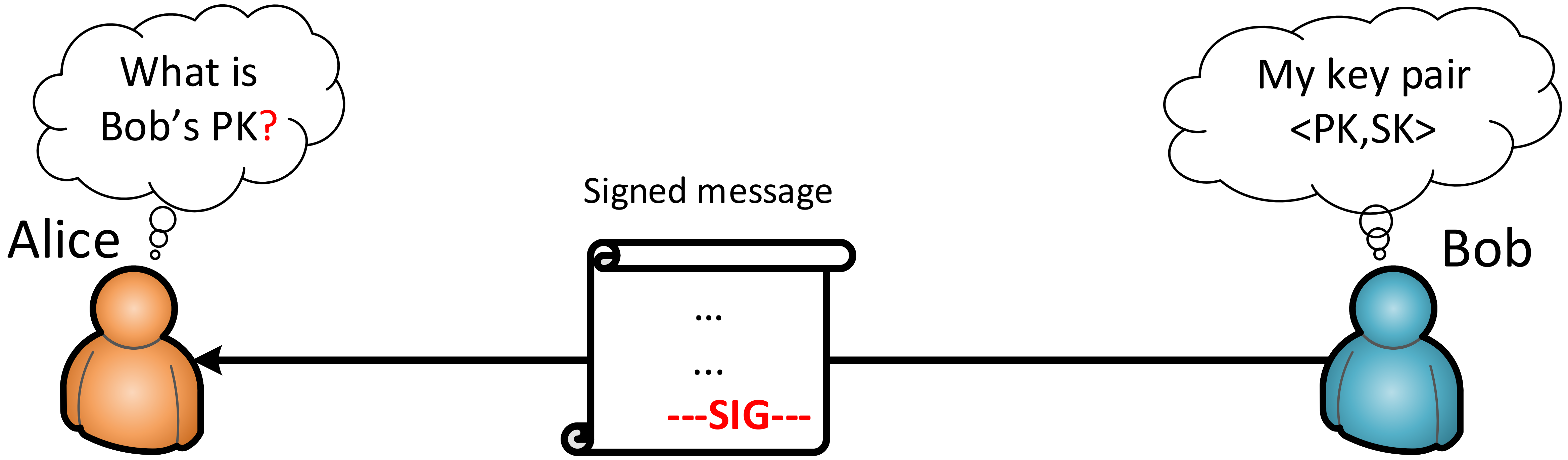
Aalto University 2023

Outline

- X.509 certificates and PKI
- Web PKI
- Transport Layer Security (TLS)

X.509 CERTIFICATES

Key distribution problem



Key distribution problem

- How to find out someone's *authentic* public key?
- Solution: an authority issues identity certificates that bind public keys to names
- Certificate is a message signed by the issuer, containing the subject's name (or identifier) and the subject's public key

$$\text{Certificate} = \text{Sign}_{\text{issuer}}(\text{Name}, PK_{\text{subject}}, \text{validity\_period})$$

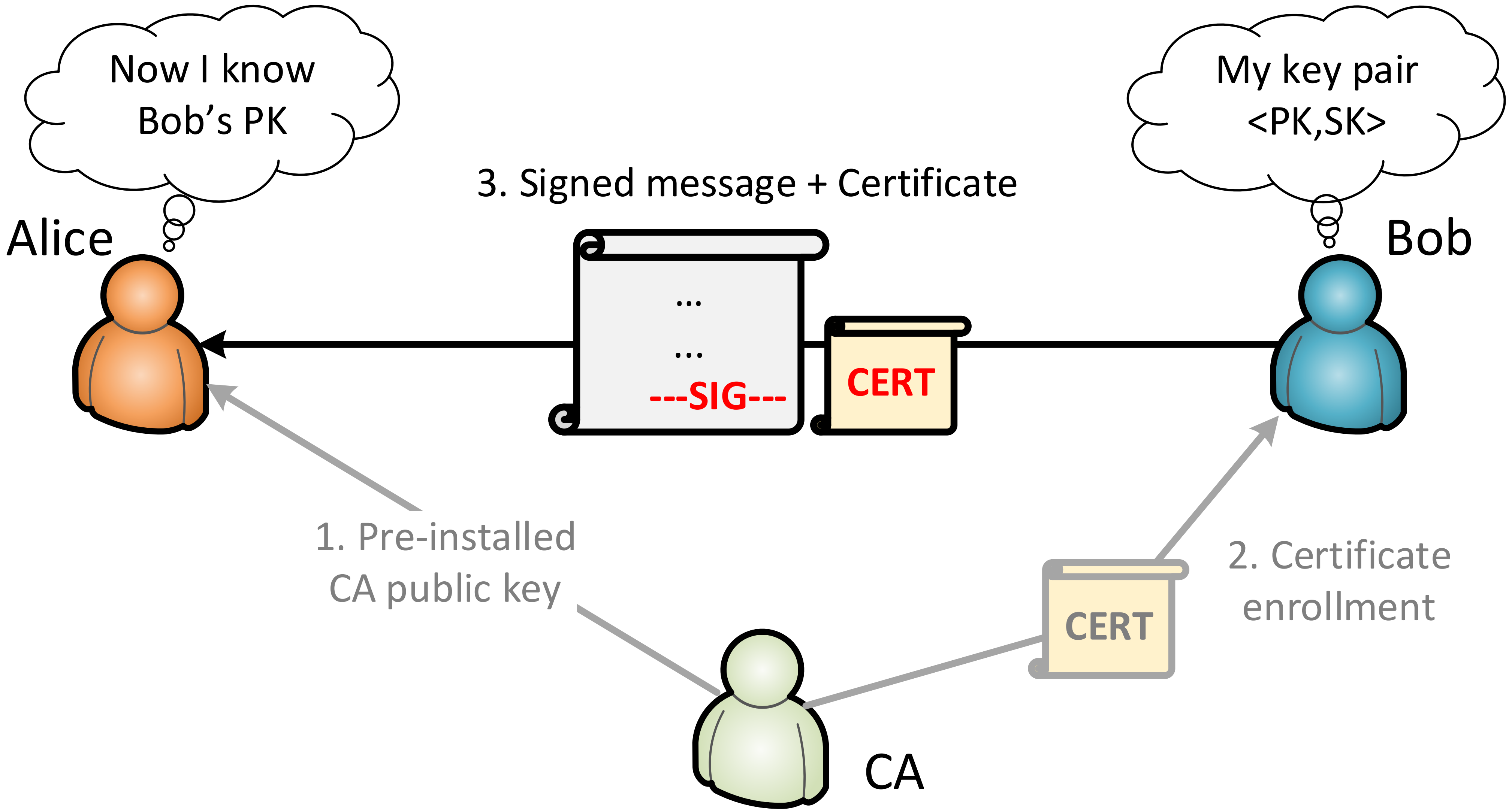
!

Certification authority (CA)

- Who would the issue the certificates?
  The issuer is typically called certification authority (CA)

- Is it an authority or a trusted third party (TTP)?
- Will everyone trust/obey the same authority?
- CA public key is needed for verifying the certificates.
  How does everyone know it?

Key distribution problem solved with certificate



X.509 PKI

- Public-key infrastructure (PKI):
  - ITU-T/ISO X.509 standard, IETF RFC 5280
- X.509 certificates are identity certificates: bind they bind a principal name to a public key
- Certification authority (CA) issues certificates
- Users, computers and services are end entities
- CAs and end entities are both principals
  - Each principal has a key pair (public and private signature key)
- CA can delegate its authority by issuing a CA certificate → CA hierarchy

X.509 certificate example

Save certificate into a file and pretty print:

```
% openssl x509 -in cert.pem -noout -text
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            35:26:d0:83:be:8f:16:bc:00:00:00:00:50:dc:67:68
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = "Entrust, Inc.", OU = See www.entrust.net/legal-terms,
             OU = "(c) 2012 Entrust, Inc. - for authorized use only", CN = Entrust Certification Authority - L1K
        Validity
            Not Before: May  2 09:48:18 2017 GMT
            Not After : May  2 10:18:16 2020 GMT
        Subject: C = FI, L = Helsinki, O = Eduskunta, CN = www.eduskunta.fi
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:b0:e2:99:41:56:8f:d2:fc:af:ae:8f:f7:e6:1f:  35:71:a1:f3:ea:bf:c7:e0:a3:14:96:f7:76:bb:90:
                    00:71:a5:3d:5b:61:34:fc:12:80:df:4f:2b:d3:31:  c4:83:73:f6:87:6b:9d:45:f5:f5:35:3d:0c:f9:f3:
                    8b:74:e5:17:8f:09:4d:e8:8d:40:f5:83:52:3b:a6:  47:a6:b7:c1:7e:a9:70:3b:4e:a8:32:5e:b9:6e:7f:
                    e3:53:0a:71:60:c5:1e:db:7d:b1:42:a4:fc:24:f7:  c9:25:6f:04:16:ec:b1:c5:04:c0:d9:93:01:58:61:
                    3c:fb:30:e3:ee:58:3c:89:9c:f7:5f:ee:0a:5b:e7:  31:ae:ee:35:7f:93:f6:57:95:20:38:23:81:0c:b6:
                    86:02:90:06:2c:0b:59:18:94:89:d3:1d:df:bd:9b:  68:d3:0a:ed:3f:fc:1f:96:ad:11:b2:d7:f7:fe:86:
                    c6:ef:80:c1:00:57:6a:97:bf:b7:75:2e:ed:08:ab:  28:c1:09:21:39:14:39:da:dd:be:ab:c7:d5:1b:bd:
                    76:a8:66:75:78:59:fe:37:08:c5:40:36:93:03:09:  2b:3a:02:08:71:01:78:db:05:46:d7:b9:9f:dd:ef:
                    98:af:cd:70:19:9c:a0:72:77:3f:1b:4e:f0:56:de:  e0:6f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://crl.entrust.net/level1k.crl
            X509v3 Certificate Policies:
                Policy: 2.16.840.1.114028.10.1.5
                  CPS: http://www.entrust.net/rpa
                Policy: 2.23.140.1.2.2

            Authority Information Access:
                OCSP - URI:http://ocsp.entrust.net
```

Issuer info

Validity dates

Subject name

Subject public key

Key usage

Revocation list URL

X.509 certificate example

Save certificate into a file and pretty print:

```
% openssl x509 -in cert.pem -noout -text
```

```
                3c:fb:30:e3:ee:58:3c:89:9c:f7:5f:ee:0a:5b:e7:  31:ae:ee:35:7f:93:f6:57:95:20:38:23:81:0c:b6:
                86:02:90:06:2c:0b:59:18:94:89:d3:1d:df:bd:9b:  68:d3:0a:ed:3f:fc:1f:96:ad:11:b2:d7:f7:fe:86:
                c6:ef:80:c1:00:57:6a:97:bf:b7:75:2e:ed:08:ab:  28:c1:09:21:39:14:39:da:dd:be:ab:c7:d5:1b:bd:
                76:a8:66:75:78:59:fe:37:08:c5:40:36:93:03:09:  2b:3a:02:08:71:01:78:db:05:46:d7:b9:9f:dd:ef:
                98:af:cd:70:19:9c:a0:72:77:3f:1b:4e:f0:56:de:  e0:6f
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
```

Key usage

```
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://crl.entrust.net/level1k.crl
```

Revocation list URL

```
            X509v3 Certificate Policies:
                Policy: 2.16.840.1.114028.10.1.5
                  CPS: http://www.entrust.net/rpa
                Policy: 2.23.140.1.2.2

            Authority Information Access:
                OCSP - URI:http://ocsp.entrust.net
                CA Issuers - URI:http://aia.entrust.net/l1k-chain256.cer

            X509v3 Subject Alternative Name:
                DNS:www.eduskunta.fi, DNS:eduskunta.fi, DNS:www.riksdagen.fi, DNS:www.parliament.fi, DNS:riksdagen.fi
```

Subject alternative names

```
            X509v3 Authority Key Identifier:
                keyid:82:A2:70:74:DD:BC:53:3F:CF:7B:D4:F7:CD:7F:A7:60:C6:0A:4C:BF

            X509v3 Subject Key Identifier:
                D6:2B:E6:54:52:A1:CE:DC:AE:01:13:FC:1D:BE:14:62:F6:F8:68:3C
            X509v3 Basic Constraints:
                CA:FALSE
```

CA or end-entity?

```
    Signature Algorithm: sha256WithRSAEncryption
         d8:36:b8:b5:8a:3f:f0:cd:fe:f3:b1:d2:86:a4:8c:d8:34:53:  a5:6d:38:9e:67:e5:ba:9d:b6:61:c2:aa:79:b8:56:5b:67:eb:
         32:75:00:e3:7b:a4:ee:c6:ce:9a:db:5c:ce:59:aa:45:cd:5a:  73:86:f9:cd:33:f8:f4:1a:9e:8a:ef:25:ff:45:71:40:2d:d7:
         d6:9e:97:48:9d:70:91:2e:3c:0b:df:d3:b6:0e:ba:66:87:e0:  f8:97:1a:3d:2a:38:1b:6c:fb:be:ca:e6:98:d2:e3:02:ba:29:
         04:e5:13:aa:c7:42:35:3f:a7:ca:17:15:fa:05:ad:62:11:45:  4d:3e:c9:c2:2a:c2:67:31:64:95:88:e3:d3:d8:c8:9f:76:77:
         8e:f7:91:c8:53:bf:c5:9d:b2:7f:4c:37:74:7e:4e:a5:96:74:  e2:3f:94:58:01:8a:91:ac:84:c9:93:f5:b1:25:aa:9f:1a:34:
         07:23:03:31:4c:26:01:ab:fa:a7:f8:ff:6e:83:ff:a1:69:7c:  2a:2a:0a:e0:ae:06:69:0e:de:52:db:95:79:0d:6c:f3:d6:d5:
         60:aa:26:83:3f:47:09:d8:9e:f6:03:f1:29:bd:b6:33:8e:7c:  d1:e6:0f:82:cd:18:59:c6:4f:fb:8f:ba:45:a7:ab:5b:6a:2b:
         fa:93:46:21
```

CA signature

X.509 certificate example

Save certificate into a file and pretty print:

```
% openssl x509 -in cert.pem -noout -text
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            35:26:d0:83:be:8f:16:bc:00:00:00:00:50:dc:67:68
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = "Entrust, Inc.", OU = See www.entrust.net/legal-terms,
             OU = "(c) 2012 Entrust, Inc. - for authorized use only", CN = Entrust Certification Authority - L1K
        Validity
            Not Before: May  2 09:48:18 2017 GMT
            Not After : May  2 10:18:16 2020 GMT
        Subject: C = FI, L = Helsinki, O = Eduskunta, CN = www.eduskunta.fi
```

Issuer info

Validity dates

Subject name

```
Subject: C = FI, L = Helsinki, O = Eduskunta, CN = www.eduskunta.fi
```

```
                Modulus:
                    00:b0:e2:99:41:56:8f:d2:fc:af:ae:8f:f7:e6:1f:  35:71:a1:f3:ea:bf:c7:e0:a3:14:96:f7:76:bb:90:
                    00:71:a5:3d:5b:61:34:fc:12:80:df:4f:2b:d3:31:  c4:83:73:f6:87:6b:9d:45:f5:f5:35:3d:0c:f9:f3:
```

Subject public key

```
X509v3 Key Usage: critical
      Digital Signature, Key Encipherment
X509v3 Extended Key Usage:
      TLS Web Server Authentication, TLS Web Client Authentication
```

```
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://crl.entrust.net/level1k.crl
```

Key usage

Revocation list URL

```
X509v3 CRL Distribution Points:
      Full Name:
            URI:http://crl.entrust.net/level1k.crl
```

```
            Authority Information Access:
                OCSP - URI:http://ocsp.entrust.net
```

X.509 certificate example

Save certificate into a file and pretty print:

```
% openssl x509 -in cert.pem -noout -text
```

```
                3c:fb:30:e3:ee:58:3c:89:9c:f7:5f:ee:0a:5b:e7:  31:ae:ee:35:7f:93:f6:57:95:20:38:23:81:0c:b6:
                86:02:90:06:2c:0b:59:18:94:89:d3:1d:df:bd:9b:  68:d3:0a:ed:3f:fc:1f:96:ad:11:b2:d7:f7:fe:86:
                c6:ef:80:c1:00:57:6a:97:bf:b7:75:2e:ed:08:ab:  28:c1:09:21:39:14:39:da:dd:be:ab:c7:d5:1b:bd:
                76:a8:66:75:78:59:fe:37:08:c5:40:36:93:03:09:  2b:3a:02:08:71:01:78:db:05:46:d7:b9:9f:dd:ef:
                98:af:cd:70:19:9c:a0:72:77:3f:1b:4e:f0:56:de:  e0:6f
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Key Usage: critical
            Digital Signature, Key Encipherment
        X509v3 Extended Key Usage:
            TLS Web Server Authentication, TLS Web Client Authentication
        X509v3 CRL Distribution Points:
            Full Name:
              URI:http://crl.entrust.net/level1k.crl
        X509v3 Certificate Policies:
            OCSP - URI:http://ocsp.entrust.net
            CA Issuers - URI:http://aia.entrust.net/l1k-chain256.cer

        X509v3 Subject Alternative Name:
            DNS:www.eduskunta.fi, DNS:eduskunta.fi, DNS:www.riksdagen.fi, DNS:www.parliament.fi, DNS:riksdagen.fi
        X509v3 Authority Key Identifier:
        X509v3 Basic Constraints:
            CA:FALSE
Signature Algorithm: sha256WithRSAEncryption
     d8:36:b8:b5:8a:3f:f0:cd:fe:f3:b1:d2:86:a4:8c:d8:34:53:  a5:6d:38:9e:67:e5:ba:9d:b6:61:c2:aa:79:b8:56:5b:67:eb:
     32:75:00:e3:7b:a4:ee:c6:ce:9a:db:5c:ce:59:aa:45:cd:5a:  73:86:f9:cd:33:f8:f4:1a:9e:8a:ef:25:ff:45:71:40:2d:d7:
     d6:9e:97:48:9d:70:91:2e:3c:0b:df:d3:b6:0e:ba:66:87:e0:  f8:97:1a:3d:2a:38:1b:6c:fb:be:ca:e6:98:d2:e3:02:ba:29:
     04:e5:13:aa:c7:42:35:3f:a7:ca:17:15:fa:05:ad:62:11:45:  4d:3e:c9:c2:2a:c2:67:31:64:95:88:e3:d3:d8:c8:9f:76:77:
     8e:f7:91:c8:53:bf:c5:9d:b2:7f:4c:37:74:7e:4e:a5:96:74:  e2:3f:94:58:01:8a:91:ac:84:c9:93:f5:b1:25:aa:9f:1a:34:
     07:23:03:31:4c:26:01:ab:fa:a7:f8:ff:6e:83:ff:a1:69:7c:  2a:2a:0a:e0:ae:06:69:0e:de:52:db:95:79:0d:6c:f3:d6:d5:
     60:aa:26:83:3f:47:09:d8:9e:f6:03:f1:29:bd:b6:33:8e:7c:  d1:e6:0f:82:cd:18:59:c6:4f:fb:8f:ba:45:a7:ab:5b:6a:2b:
     fa:93:46:21
```

Key usage

Revocation list URL

```
X509v3 Subject Alternative Name:
       DNS:www.eduskunta.fi, DNS:eduskunta.fi, DNS:www.riksdagen.fi,
       DNS:www.parliament.fi, DNS:riksdagen.fi
```

Subject alternative names

```
 X509v3 Basic Constraints:
                      CA:FALSE
```

CA or end-entity?

CA signature

Viewing certificates with OpenSSL

```
# Download the certificate chain
S=idp.aalto.fi && echo | openssl s_client -connect $S:443 -servername
$S -CApath /etc/ssl/certs/ -showcerts > chain.tmp
# Parse the certificates into separate files
cat chain.tmp | awk '/-BEGIN CERTIFICATE-/ {b=1} {if (b) print >
"cert"(i+1)".pem"} /-END CERTIFICATE-/ {i++; b=0}'
# Prettyprint the host certificate
openssl x509 -text -noout -in cert1.pem | less
# Prettyprint the certificate chain
ls cert?.pem | xargs -n1 openssl x509 -text -noout -in > pretty.txt

# Try some invalid certificates: https://badssl.com/
```

X.509 certificate fields (1)

Mandatory fields:

- Version
- Serial number — together with Issuer, uniquely identifiers the certificate
- Signature algorithm — for the signature on this certificate; usually *sha1RSA*; includes any parameters
- Issuer — name (e.g. CN = Microsoft Corp Enterprise CA 2)
- Valid from — usually the time when issued
- Valid to — expiry time
- Subject — distinguished name of the subject
- Public key — public key of the subject

X.509 certificate fields (2)

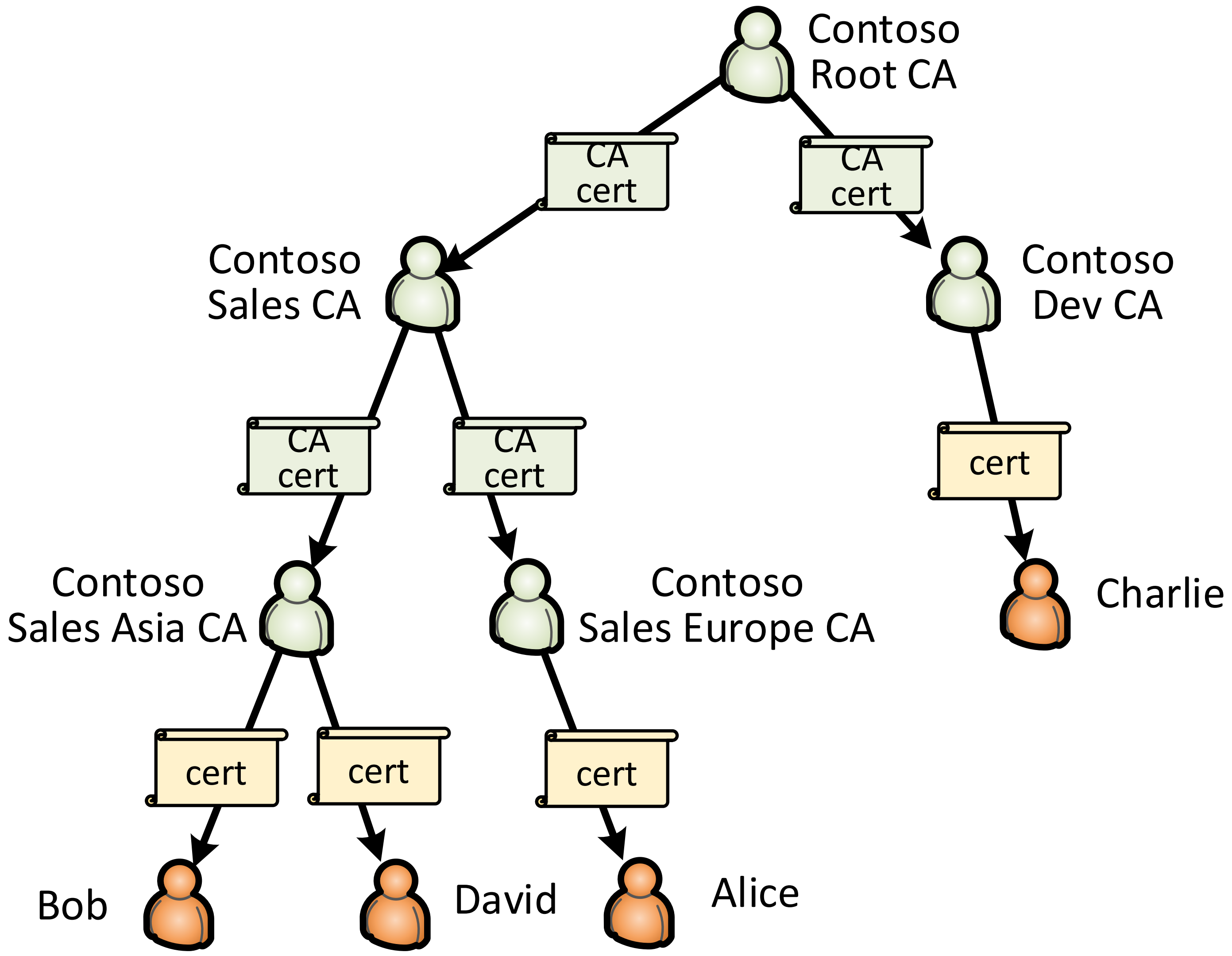
Common extension fields:

- Key usage — bit field indicating usages for the subject key (*digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, decipherOnly*)
- Subject alternative name — email address, DNS name, IP address, etc.
- Issuer alternative name
- Basic constraints — (1) is the subject a CA or an end entity, (2) maximum length of delegation to sub-CAs after the subject
- Name constraints — limit the authority of the CA
- Certificate policies — list of OIDs to indicate policies for the certificate
- Policy constraints — certificate policies
- Extended key usage — list of OIDs for new usages, e.g. server authentication, client authentication, code signing, email protection, EFS key, etc.
- CRL distribution point — where to get the CRL for this certificate, and who issues CRLs
- Authority info access — where to find information about the CA and its policies

PUBLIC-KEY INFRASTRUCTURE (PKI)

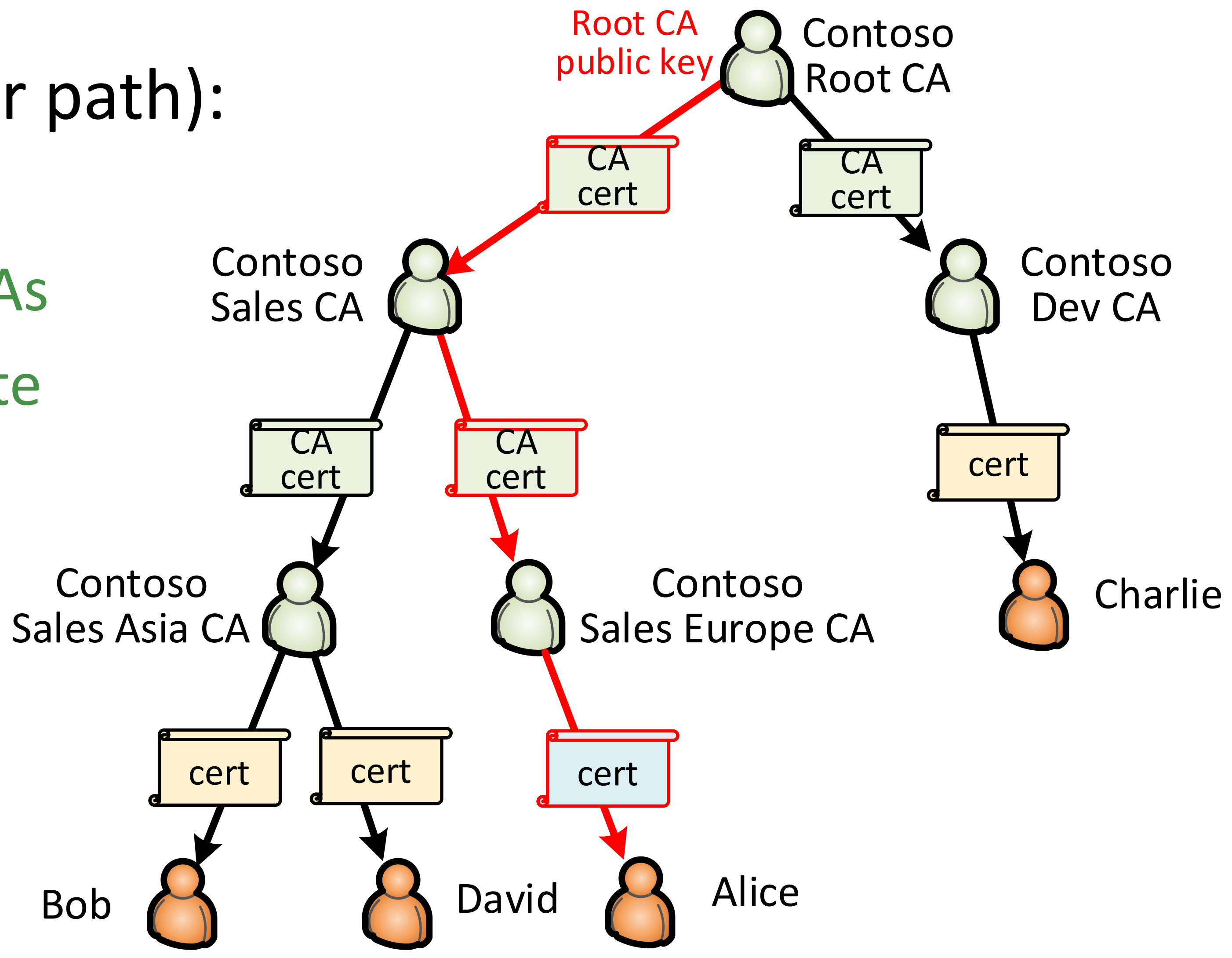
X.509 CA hierarchy

- One root CA
- Each CA can delegate its authority to sub-CAs
- All end-entities trust all CAs to be honest and competent



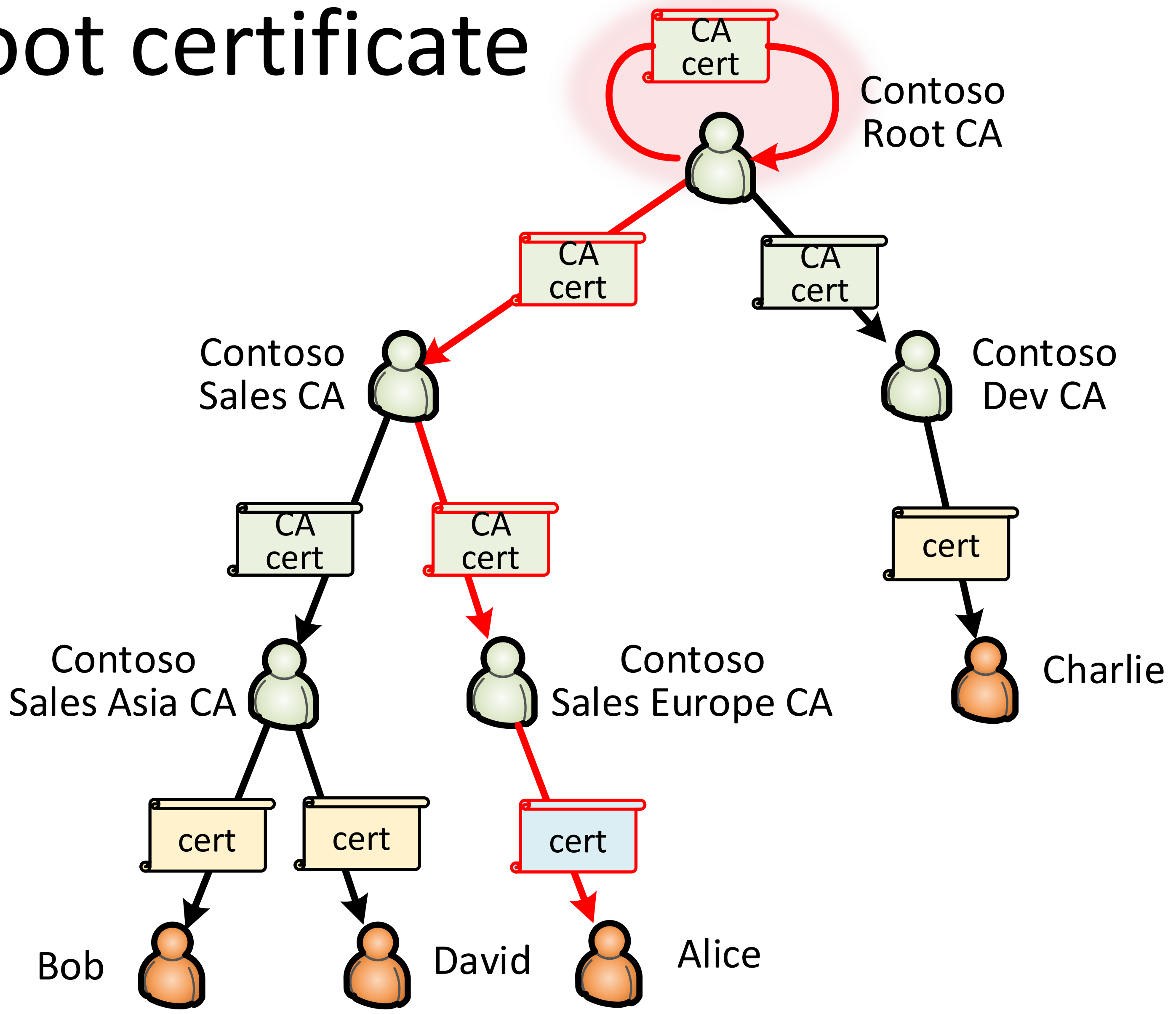
Certificate chain

- Alice's certificate chain (or path):
  - Root CA public key
  - 2 CA certificates for sub-CAs
  - Alice's end-entity certificate
- Root of trust:
  - everyone knows and trusts the root CA's public key



Self-signed root certificate

- Self-signed certificate
  - Issuer and subject keys are the same
  - Often included in the certificate chain
  - Not really a certificate; just a way to store and communicate the root CA public key



Real-world PKIs

Original X.500 idea: one global CA hierarchy to certify all countries, organizations, users, computer and services

Reality: many application and organization specific PKIs

- Web PKI for certifying web servers
  - Many commercial and free root CAs, e.g. Verisign, Telia, Let's Encrypt
- S/MIME for signed (and encrypted) email
  - Commercial CAs certify organizational CAs for cross-organization email
- Smart-card PKIs
  - Bank cards, national identity cards
- Organizational PKIs
  - Windows domain users, computers and services; user login with smartcard; internal web pages; VPN and Wi-Fi access; S/MIME email; Adobe document signing PDF documents

Name and identity

- With the help of certificates, it is possible to authenticate the name or identifier of an entity (e.g. person, web server)
- But what is the right name for an entity?
  - wwwlogin.tkk.fi, idp.aalto.fi, leakybox.cse.tkk.fi
  - George Bush, George W. Bush, George H. W. Bush
  - tuomas.aura@aalto.fi, aaura@hut.fi, taura@cse.tkk.fi, aura@cse.tkk.fi
- Who decides who owns the name?
  - @aalto.fi email addresses, DNS names, Facebook username
- Does knowing the name imply trust?
  - Is it safe to buy a used bicycle from trustedbikes.fi?
  - Should they verify the customer’s name before shipping?

CERTIFICATE REVOCATION: CRL AND OCSP

Need for certificate revocation

- When might CA need to revoke (i.e. cancel) a certificate?
  - If the conditions for issuing the certificate no longer hold: e.g., employee leaves, student graduates, or computer is decommissioned
  - If the certificate was originally issued in error
  - If the subject private key has been compromised
  - When upgrading cryptographic algorithms

!

- Certificate can be verified offline, but revocation requires online checks

Revocation list

- Certificate revocation list (CRL) = signed list of revoked certificate serial numbers and a timestamp

- Who issues the CRL? How to find it?
  - CRL distribution point and issuer may be specified in each certificate
  - By default, CRL is signed by the same CA that issued the certificate

- Certificate verifier downloads the CRL (or delta) and caches it
  - If CRL server is offline, the certificate verification fails

- Expired certificates can be removed from the CRL

- In X.509, only certificates are revoked, not keys

X.509 CRL fields

- Signature algorithm
- Issuer — name
- This update — time
- Next update — time

For each revoked certificate:

- Serial number
- Revocation date — (how would you use this information?)
- Extensions — reason code etc.

- Signature

OCSP

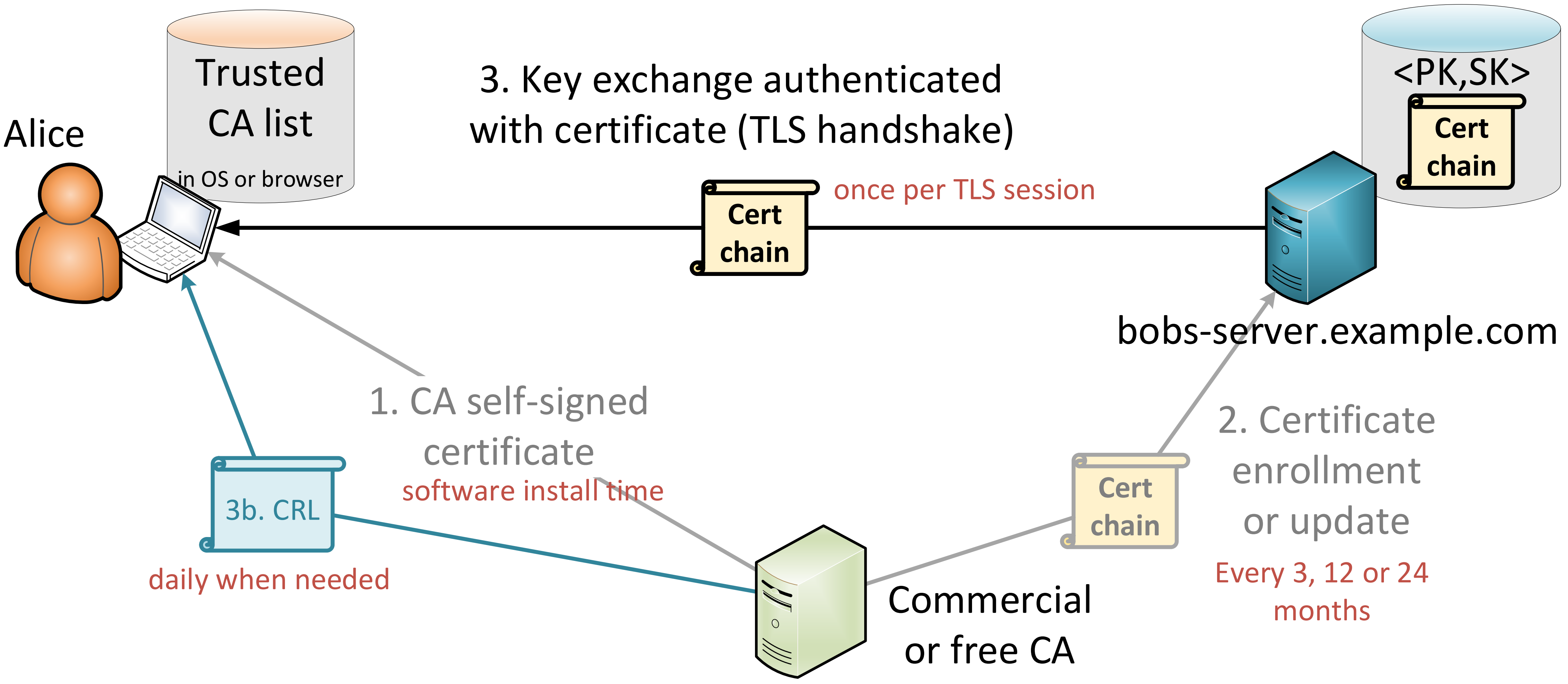
- Online certificate status protocol (OCSP)
  - Request-response protocol for checking certificate status from issuer
  - Timestamp and optional nonce for response freshness

C → CA: certificate-id, N_C
CA → C: Sign_{CA}(certificate-id, certificate status, T, [N_C])

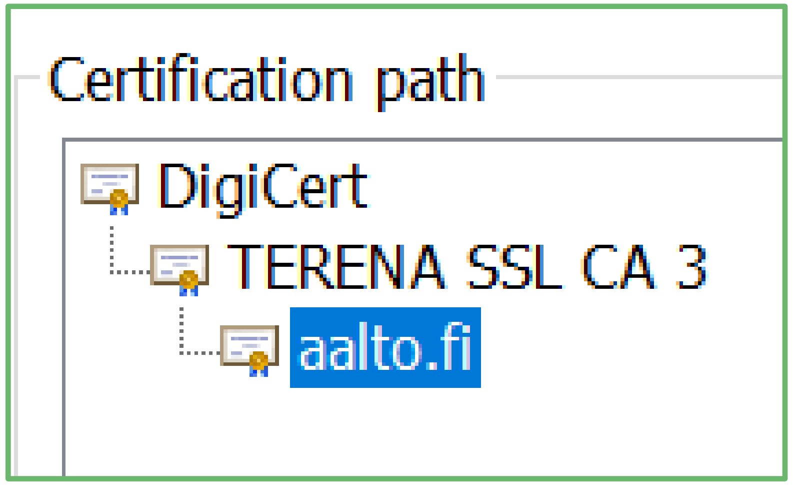
- CRL vs OSCP
  - OCSP implementation simpler and messages shorter
  - OCSP server learns which web pages the client is accessing

WEB PKI

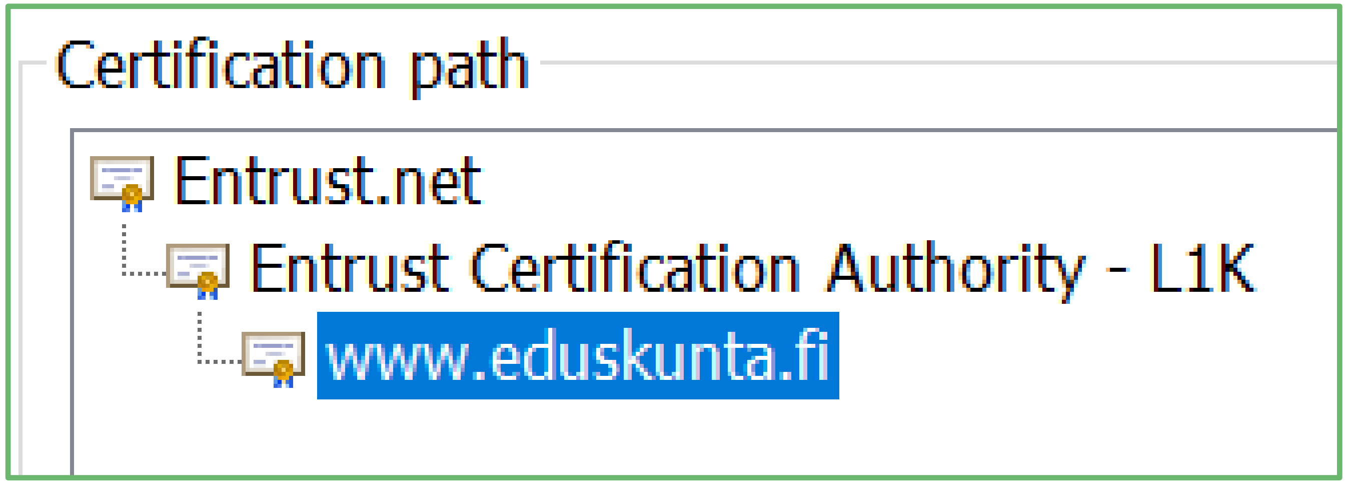
Web PKI


Alice
Trusted CA list
in OS or browser
3. Key exchange authenticated with certificate (TLS handshake)
Cert chain
once per TLS session
<PK,SK>
Cert chain
bobs-server.example.com
1. CA self-signed certificate
software install time
3b. CRL
daily when needed
Cert chain
2. Certificate enrollment or update
Every 3, 12 or 24 months
Commercial or free CA

Web PKI


Certification path

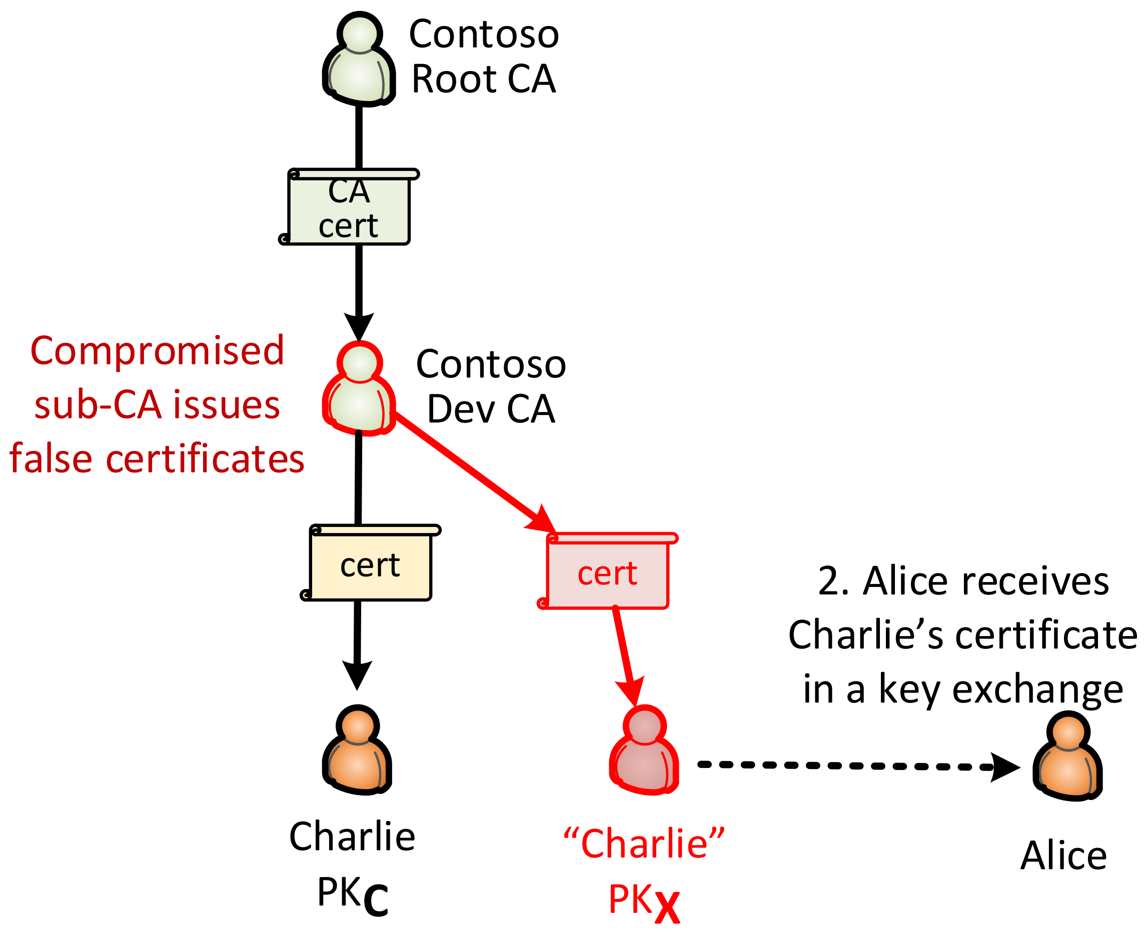
- DigiCert
  - TERENA SSL CA 3
    - aalto.fi


Certification path

- Entrust.net
  - Entrust Certification Authority - L1K
    - www.eduskunta.fi

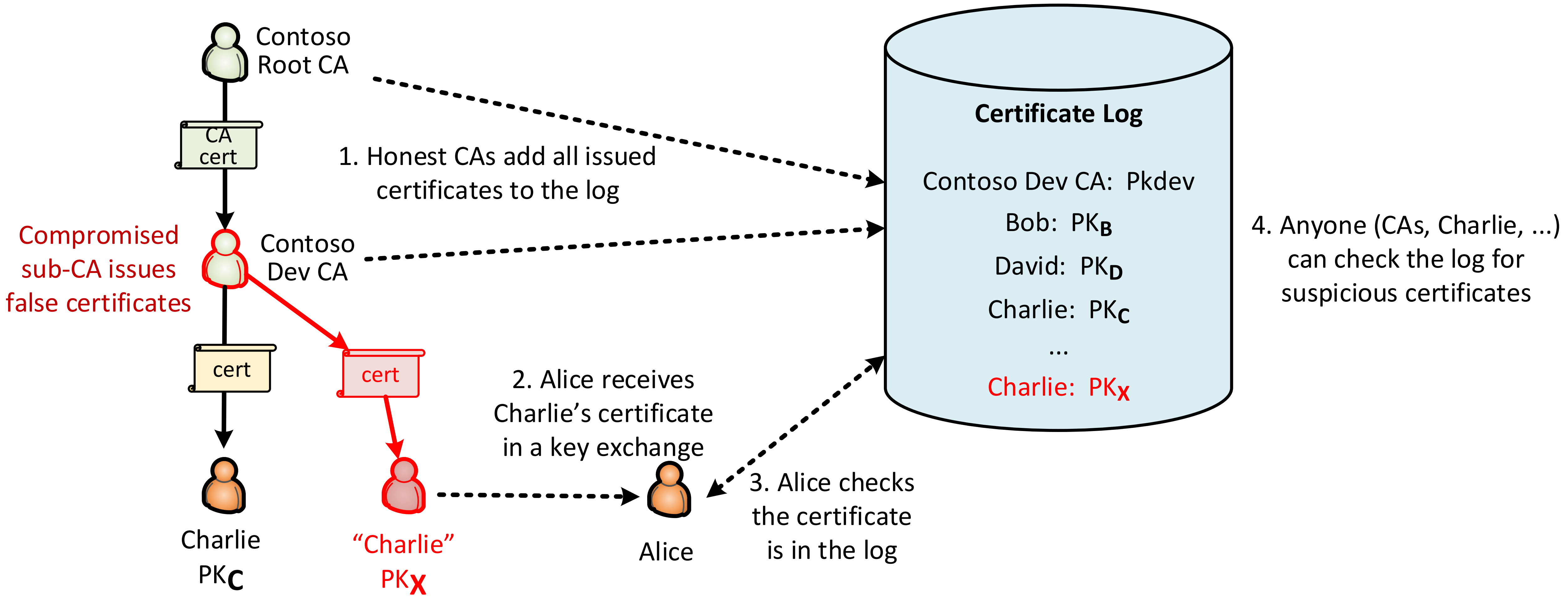
1. Root CA's self-signed certificate
   - Issued by the root CA to itself; essentially just the CA public key
2. Root CA issues a CA certificate to a sub-CA
   - Typically one sub-CAs in the chain (why?)
3. Sub-CA issues end-entity certificate to a web server

Problems with revocation in web PKI


Contoso
Root CA
CA cert
Compromised sub-CA issues false certificates
Contoso
Dev CA
cert
cert
2. Alice receives Charlie's certificate in a key exchange
Charlie
PKC
"Charlie"
PKX
Alice

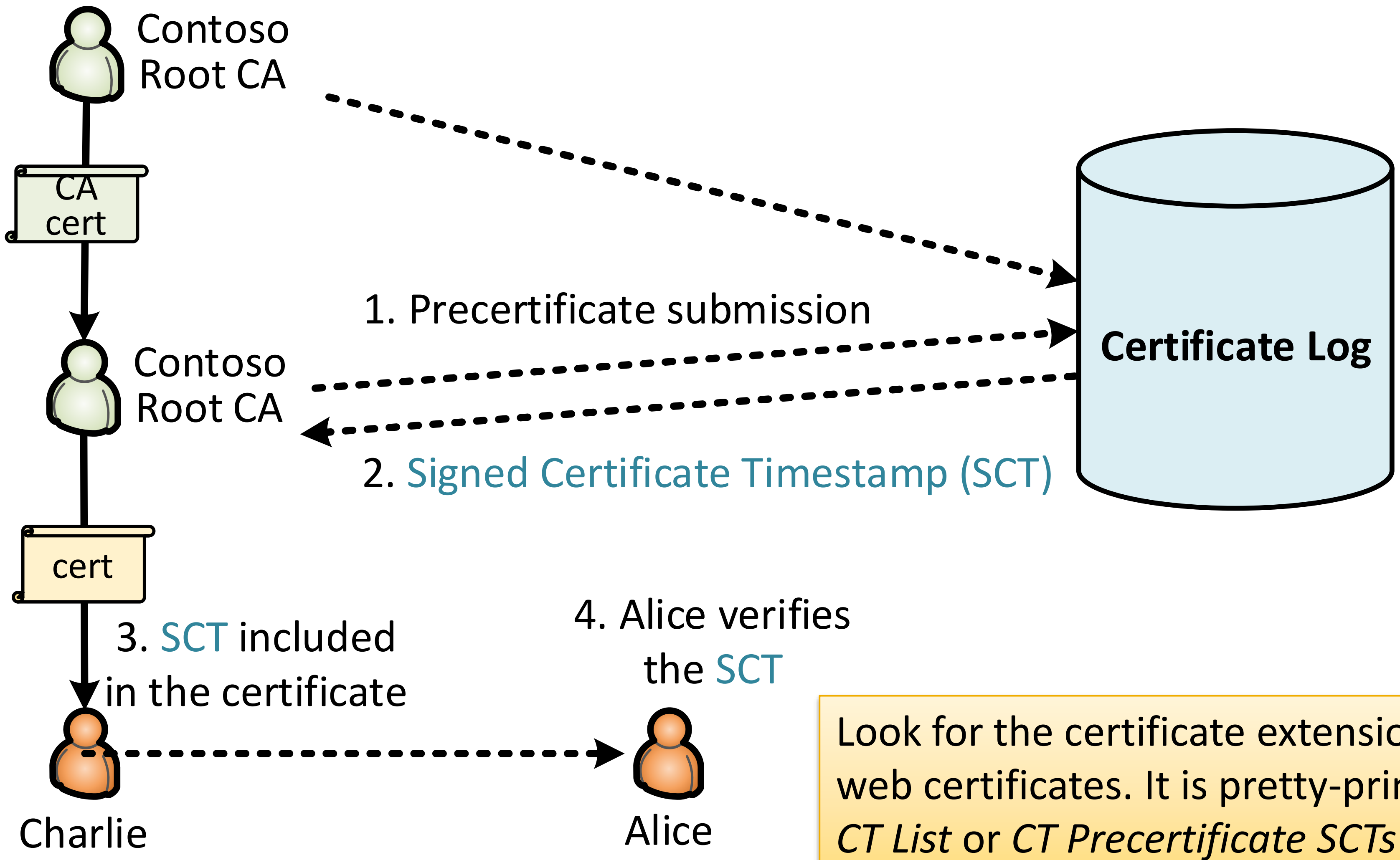
- Web clients typically ignore CRL or OCSP, especially if the server does not respond
- If a sub-CA is compromised, it won't be detected and, thus, won't be revoked

Certificate Transparency (CT)



Contoso Root CA
CA cert
1. Honest CAs add all issued certificates to the log
Compromised sub-CA issues false certificates
Contoso Dev CA
cert
cert
Charlie PK$_C$
"Charlie" PK$_X$
2. Alice receives Charlie's certificate in a key exchange
Alice
3. Alice checks the certificate is in the log
Certificate Log
Contoso Dev CA: Pkdev
Bob: PK$_B$
David: PK$_D$
Charlie: PK$_C$
…
Charlie: PK$_X$
4. Anyone (CAs, Charlie, …) can check the log for suspicious certificates

Certificate Transparency (CT)


Contoso Root CA
CA cert
Contoso Root CA
cert
Certificate Log
1. Precertificate submission
2. Signed Certificate Timestamp (SCT)
3. SCT included in the certificate
4. Alice verifies the SCT
Charlie
Alice

Signed timestamp (SCT) in the certificate proves that CA has submitted its information to CT
→Browser does not need to access the log directly

Look for the certificate extension in web certificates. It is pretty-printed as *CT List* or *CT Precertificate SCTs*

Certificate Transparency (CT) details

- Anyone can submit certificates to the logs
- Typical process:
  - CA submits the certificate information (Precertificate) to one or more CT logs before issuing the certificate
  - CT log returns a Signed Certificate Timestamp (SCT) to CA
  - CA includes SCT in a certificate extension field
  - SCT proves to browser that the certificate is in CT; no access to the log is needed
- Browser has a list of trusted CT logs
  - Several commercial and free logs exist: Google, Cloudflare, Let's Encrypt, …
- Browser may reject or warn about certificates that are not in CT
- Enterprises may configure browsers to not enforce CT for certificates issued by their private CA (because the public logs would leak information)

Identify proofing in web PKI

- Identity proofing = checking subject identity before certification
- Commercial web certificates:
  - Typically, email to registered domain owner verifies the ownership
  - Extended validation certificates had a stronger identify proofing process – but discontinued because users do not notice the difference
- Let's Encrypt uses the ACME protocol for automated certificate management:
  - Before issuing the certificates, CA challenges the server admins to prove that they control the web server or DNS zone

```
$ host -t TXT _acme-challenge.vikaa.fi
_acme-challenge.vikaa.fi descriptive text "a8LUKy+IQzdsc2wjDCAvFS6tTmckUF8PImWNEKubrbI"
```

Cost of web certificates

- Commercial certificates used to cost hundreds of euros per year, more for wildcard names
- Google and others have pushed for everyone to use TLS, but earlier the cost was too high
- Let’s Encrypt CA, run by a non-profit organization, issues free certificates

SETTING UP YOUR OWN PKI

Setting up your own PKI

- Creating a root CA :
  - Anyone can set up a CA with OpenSSL or commercial software
  - Windows root domain controller can be a CA for the domain
  - Commercial PKI products and services exist for enterprises

- It will be a closed, private PKI:
  - Commercial CAs will not certify your sub-CA (why?)
  - You cannot ask users outside your own organization to install your root key to their web browsers (why?)

Setting up your own PKI

- The real costs:
  - Distributing the root key (root CA's self-signed certificate) to all who need to verify certificates, e.g. all web browsers

!

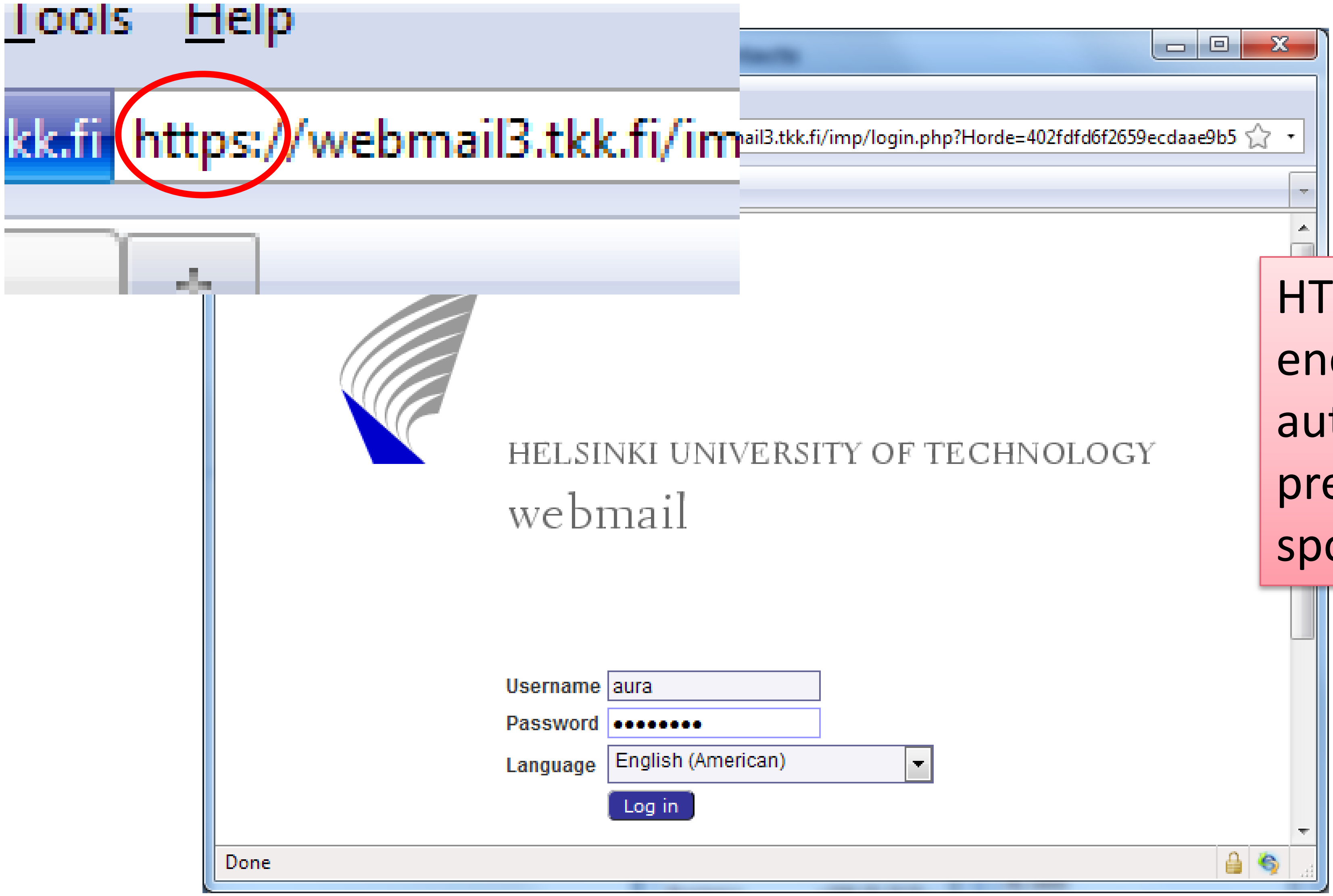
  - Certificate enrolment —issuing certificates for each web site, user, computer, mobile device etc. that needs them
  - Administering a secure CA and CRL/OCSP server
- No security advantage unless your remove all other trusted CAs, but then you cannot access the public web

Experience: PKI at home

- Used OpenSSL to set up a PKI for my home network
  - TLS server certificates for web UI and APIs in the SDN controller, NAS, printer
  - Wireless network access with WPA2-Enterpise
  - 802.1X (RADIUS) access control for some Ethernet ports
- OpenSSL command line is not easy to master, but scripting helps
  - No CRL or OCSP; in emergency, need to reissue all certificates
- Lack of support in consumer hardware:
  - Some devices do not support TLS or RADIUS; some only use self-signed certificates
  - RADIUS server in my router only supports PEAP and no EAP-TLS
- Debugging access issues is hard; insufficient logging of failures and their reasons
- Guest access became a major headache:
  - Guests must install my root CA certificate to their trusted list; is that safe?
  - Guests and family members may not have root access to their work laptops
  - The root certificate has a name constraint, but few understand its meaning

TRANSPORT LAYER SECURITY (TLS 1.2)

Secure web site (https)

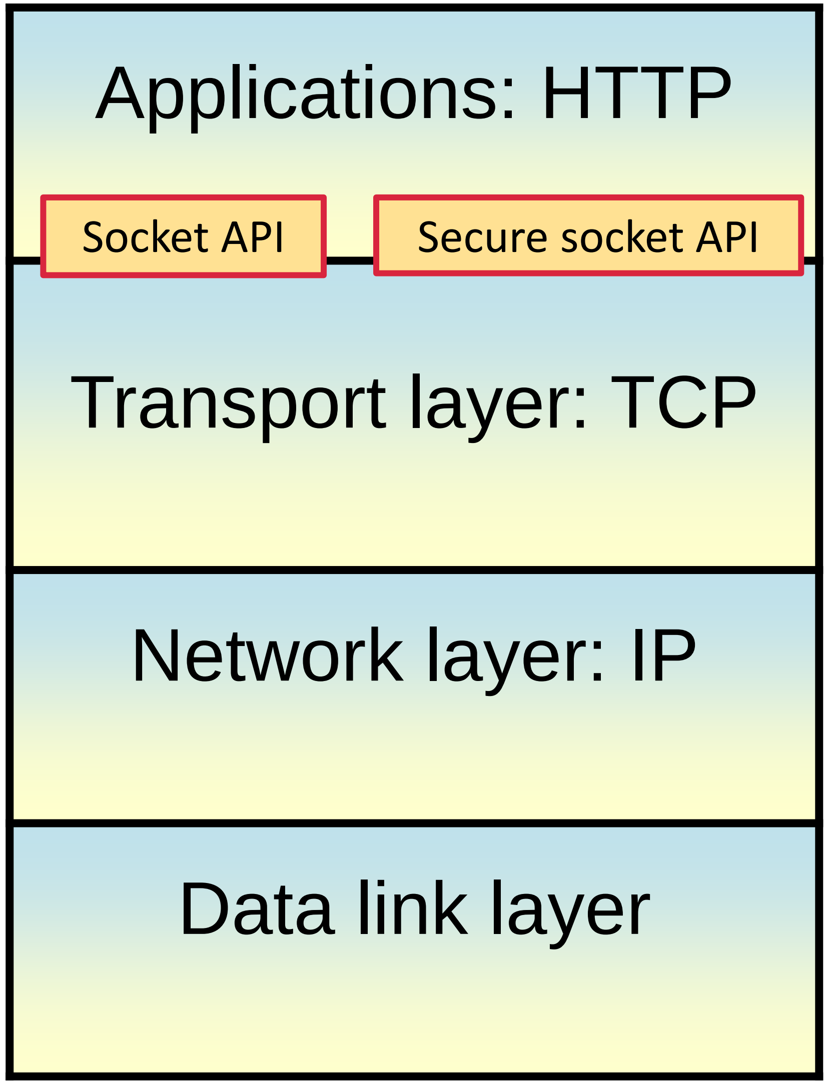

Tools Help
kk.fi https://webmail3.tkk.fi/im
ail3.tkk.fi/imp/login.php?Horde=402fdfd6f2659ecdaae9b5
HELSINKI UNIVERSITY OF TECHNOLOGY
webmail
Log in
Done

Username aura
Password ●●●●●●●●
Language English (American)

HTTPS connections are encrypted and authenticated to prevent sniffing and spoofing

TLS in the protocol stack

- TLS implements cryptographic encryption and authentication for TCP connections
  - Secure socket API for applications
- TLS 1.3 is the latest standard
  - SSL was TLS' historical predecessor
- DTLS for UDP



Handshake and session

- Two stages of a typical network security protocol:
  - Handshake = authenticated key exchange creates a shared session key
  - Session protocol protects the confidentiality and integrity of the session data with symmetric cryptography and the session key

!

- TLS handshake
  - Client and server create a shared secret key with Diffie-Hellman
  - Server authenticates to the client with a certificate chain and signature
  - Client authentication optional, usually left to the application layer
- TLS session protocol uses symmetric encryption and HMAC to protect the application data

TLS_DHE_DSS handshake

1. Negotiation
2. Diffie-Hellman
3. Nonces
4. Server certificates
5. Server signature
6. Key confirmation

1. C → S: Versions, N_C, SessionId, CipherSuites

2. S → C: Version, N_S, SessionId, CipherSuite
   CertChain_S, g, n, g^y mod n, $\text{Sign}_S(N_C, N_S, g, n, g^y \bmod n)$

3. C → S: g^x mod n
   ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$(“client

4. S → C: ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$("server

This is the TLS handshake that creates shared session keys for the server and client

Our goal: to understand how it works

- Shared secret: g^{xy} mod n
- master_secret = h(g^{xy} mod n, “master secret”, N_C, N_S)
- ChangeCipherSpec turns on session protection with the new key

TLS_DHE_DSS handshake

1. Negotiation
2. Diffie-Hellman
3. Nonces
4. Server certificates
5. Server signature
6. Key confirmation

1. C → S: Versions, N_C , SessionId, CipherSuites

2. S → C: Version, N_S , SessionId, CipherSuite
   CertChain_S , g, n, g^y mod n, $\text{Sign}_S(N_C, N_S, g, n, g^y \bmod n)$

3. C → S: g^x mod n
   ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$("client finished", all previous messages)

4. S → C: ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$("server finished", all previous messages)

- Shared secret: g^{xy} mod n
- master_secret = h(g^{xy} mod n, "master secret", N_C, N_S)
- ChangeCipherSpec turns on session protection with the new key

TLS_DHE_DSS handshake

1. Negotiation
2. Diffie-Hellman
3. Nonces
4. Server certificates
5. Server signature
6. Key confirmation

1. C → S: Versions, N_C, SessionId, CipherSuites

2. S → C: Version, N_S, SessionId, CipherSuite
   CertChain_S, g, n, g^y mod n, $\text{Sign}_S(N_C, N_S, g, n, g^y \bmod n)$

3. C → S: g^x mod n
   ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$("client finished", all previous messages)

4. S → C: ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$("server finished", all previous messages)

- Shared secret: g^{xy} mod n
- master_secret = $h(g^{xy} \bmod n$, "master secret", $N_C, N_S)$
- ChangeCipherSpec turns on session protection with the new key

TLS_DHE_DSS handshake

1. Negotiation
2. Diffie-Hellman
3. Nonces
4. Server certificates
5. Server signature
6. Key confirmation

1. $C \rightarrow S$: Versions, N_C, SessionId, CipherSuites
2. $S \rightarrow C$: Version, N_S, SessionId, CipherSuite
   CertChain$_S$, g, n, $g^y \bmod n$, Sign$_S$(N_C, N_S, g, n, $g^y \bmod n$)
3. $C \rightarrow S$: $g^x \bmod n$
   ChangeCipherSpec
   MAC$_{\text{master\_secret}}$("client finished", all previous messages)
4. $S \rightarrow C$: ChangeCipherSpec
   MAC$_{\text{master\_secret}}$("server finished", all previous messages)

- Shared secret: $g^{xy} \bmod n$
- master_secret = h($g^{xy} \bmod n$, "master secret", N_C, N_S)
- ChangeCipherSpec turns on session protection with the new key

TLS_DHE_DSS handshake

1. Negotiation
2. Diffie-Hellman
3. Nonces
4. Server certificates
5. Server signature
6. Key confirmation

1. $C \rightarrow S$: Versions, N_C, SessionId, CipherSuites
2. $S \rightarrow C$: Version, N_S, SessionId, CipherSuite
   CertChain_S, g, n, $g^y \bmod n$, $\text{Sign}_S(N_C, N_S, g, n, g^y \bmod n)$
3. $C \rightarrow S$: $g^x \bmod n$
   ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$(“client finished”, all previous messages)
4. $S \rightarrow C$: ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$("server finished“, all previous messages)

- Shared secret: $g^{xy} \bmod n$
- master_secret = $h(g^{xy} \bmod n$, “master secret”, $N_C, N_S)$
- ChangeCipherSpec turns on session protection with the new key

TLS_DHE_DSS handshake

1. Negotiation
2. Diffie-Hellman
3. Nonces
4. Server certificates
5. Server signature
6. Key confirmation

1. $C \rightarrow S$: Versions, N_C, SessionId, CipherSuites
2. $S \rightarrow C$: Version, N_S, SessionId, CipherSuite
   CertChain_S, g, n, $g^y \bmod n$, $\text{Sign}_S(N_C, N_S, g, n, g^y \bmod n)$
3. $C \rightarrow S$: $g^x \bmod n$
   ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$("client finished", all previous messages)
4. $S \rightarrow C$: ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$("server finished", all previous messages)

- Shared secret: $g^{xy} \bmod n$
- master_secret = $h(g^{xy} \bmod n$, "master secret", N_C, $N_S)$
- ChangeCipherSpec turns on session protection with the new key

TLS_DHE_DSS handshake

1. Negotiation
2. Diffie-Hellman
3. Nonces
4. Server certificates
5. Server signature
6. Key confirmation

1. C → S: Versions, N_C , SessionId, CipherSuites

2. S → C: Version, N_S , SessionId, CipherSuite
CertChain$_S$, g, n, g^y mod n, Sign$_S$(N_C, N_S, g, n, g^y mod n)

3. C → S: g^x mod n
ChangeCipherSpec
MAC$_{master\_secret}$(“client finished”, all previous messages)

4. S → C: ChangeCipherSpec
MAC$_{master\_secret}$("server finished“, all previous messages)

- Shared secret: g^{xy} mod n
- master_secret = h(g^{xy} mod n, “master secret”, N_C, N_S)
- ChangeCipherSpec turns on session protection with the new key

TLS_DHE_DSS handshake

1. Negotiation
2. Diffie-Hellman
3. Nonces
4. Server certificates
5. Server signature
6. Key confirmation

1. $C \rightarrow S$: Versions, N_C, SessionId, CipherSuites
2. $S \rightarrow C$: Version, N_S, SessionId, CipherSuite
   CertChain_S, g, n, $g^y \bmod n$, $\text{Sign}_S(N_C, N_S, g, n, g^y \bmod n)$
3. $C \rightarrow S$: $g^x \bmod n$
   ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$(“client finished”, all previous messages)
4. $S \rightarrow C$: ChangeCipherSpec
   $\text{MAC}_{\text{master\_secret}}$("server finished“, all previous messages)

- Shared secret: $g^{xy} \bmod n$
- master_secret = $h(g^{xy} \bmod n$, “master secret”, $N_C, N_S)$
- ChangeCipherSpec turns on session protection with the new key

TLS 1.3

Extra material

- https://tls13.ulfheim.net/
- Support only the best known protocols and cryptographic algorithms
  - Ephemeral Diffie-Hellman
  - Ephemeral elliptic curve Diffie-Hellman (ECDHE)
  - AEAD authenticated encryption
- 1-RTT handshake
  - Encrypted server certificate, but SNI still plaintext
- Fast session resumption with session tickets, even 0-RTT

Old RSA handshake

Extra material

- The older RSA-based handshake protocol:
  1. The server sends its certificate chain to the client (e.g. web browser), so that the client learns the server name and its public RSA key
  2. The client generates random bytes, encrypts them with the servers RSA key, and sends to the server
  3. The session keys are created from these secret random bytes

Trust chain

!

- In the handshake, browser receives a certificate chain from the server
- Browser checks that the chain start with a (usually self-signed) certificate that is in its trusted CA list
- Browser checks the certificate chain:
  - Verifies the signature on each certificate using the subject public key of the certificate above
  - Checks that all but the last certificate are CA certificates
  - Many other details, e.g. validity time, CRL/OCSP, key usage, constraints
  - Checks that certificate appears in CT log
- If the certificate chain is valid, the last certificate binds together the host name and public key of the server
  - Public key is used for server authentication in the TLS handshake
  - Host name shown to user in the browser address bar

Certificate checking details

!

Certificate verification is quite complex and difficult to implement correctly:

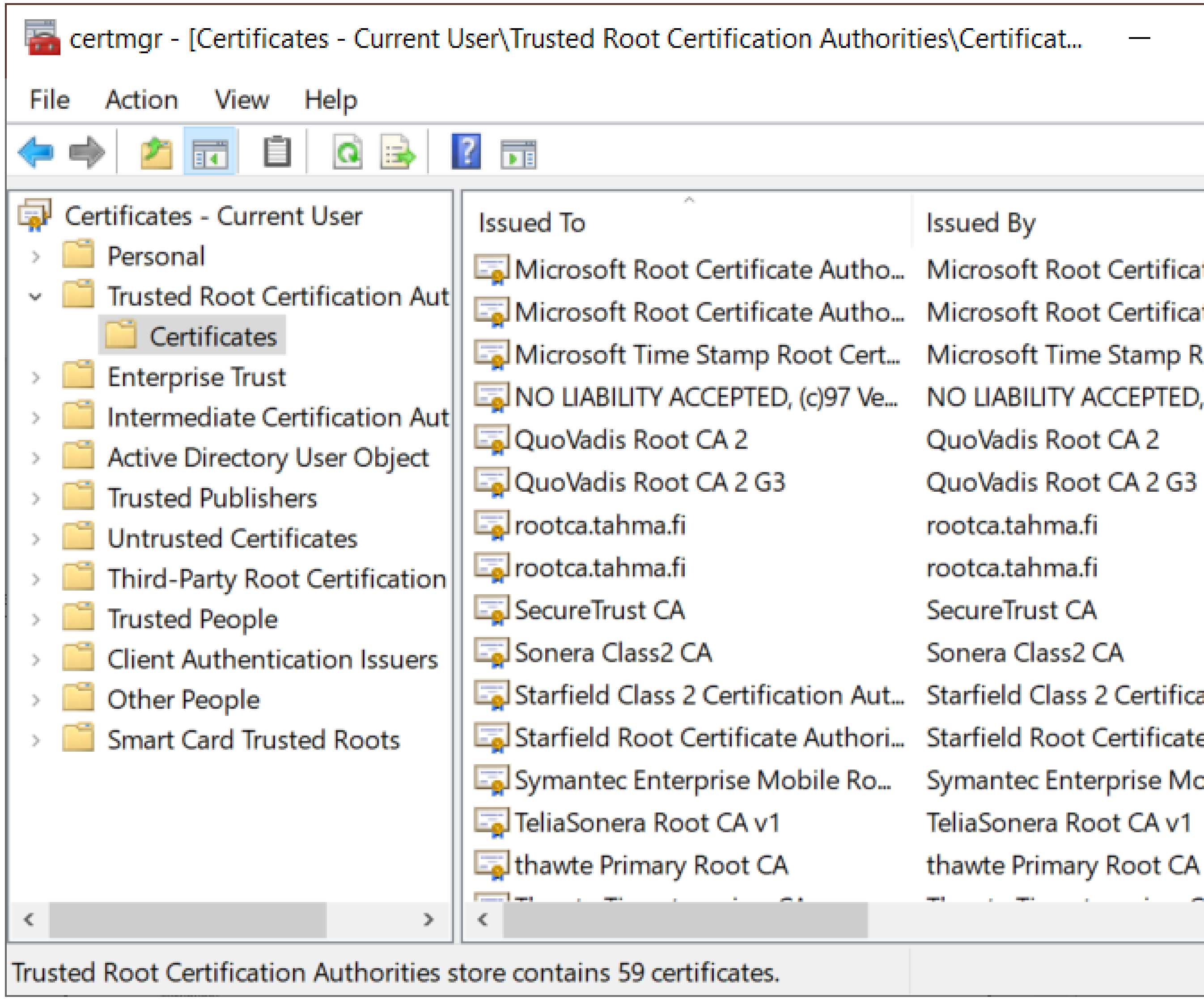
1. Browser has a list of self-signed certificates for trusted root CAs, and it may have lists of certificates for trusted sub-CAs and servers.
2. In the TLS handshake, the browser may tell the server which root CAs it recognizes.
3. The browser receives a certificate chain from the server.
4. Browser checks the validity of the certificate chain backwards (“upwards”) from the end-entity-certificate towards the root:
    A. There must be exactly one end-entity certificate at the bottom of the chain. The other certificates in the chain must be CA certificates.
    B. Issuer of each certificate must match the subject of the CA certificate above it.
    C. The browser verifies the signature of each certificate with the subject public key of the certificate above, i.e., from the issuer’s CA certificate.
    D. Browser checks for certificate revocation from the OCSP server or CRL of if the certificate specifies these.
    E. Browser checks that the certificate is in a CT log.
    F. There may be constraints in the certificates, which must also be checked. Name constraint limits the authority of a CA to specific names, usually a domain suffix. The name in the end-entity certificate must match all the name constraints in the certificate chain.
    G. The browser must recognize and process all critical extension fields in the certificates, but it may ignore non-critical extensions. (For example, name constraint and key usage are critical extensions, but CLR distribution point and Certificate Transparency timestamps are non-critical.)

    If a trusted certificate is found, stop going up the chain and move to the next step. On the other hand, if the root of the chain is reached and no trusted certificate is found, the chain verification fails.
5. Browser checks that the certificate has been issued for the right purpose: extended key usage field of the end-entity certificate must specify TLS server authentication.
6. Browser checks that the host name in the browser’s address bar or requested ULR matches the subject name of the end-entity certificate. (Subject name matching rules are pretty complex, too. There can be many names and wildcards in the certificate.)
7. Browser uses the subject key from the end-entity certificate to authenticate the server in the TLS handshake (authenticated key exchange).
8. The session key created in the handshake is used to encrypt and authenticate data between the browser and server for the duration of the TLS session.

→ This process proves that the web page shown in the browser comes from the server whose name is in the address bar.

Where is the root CA list?

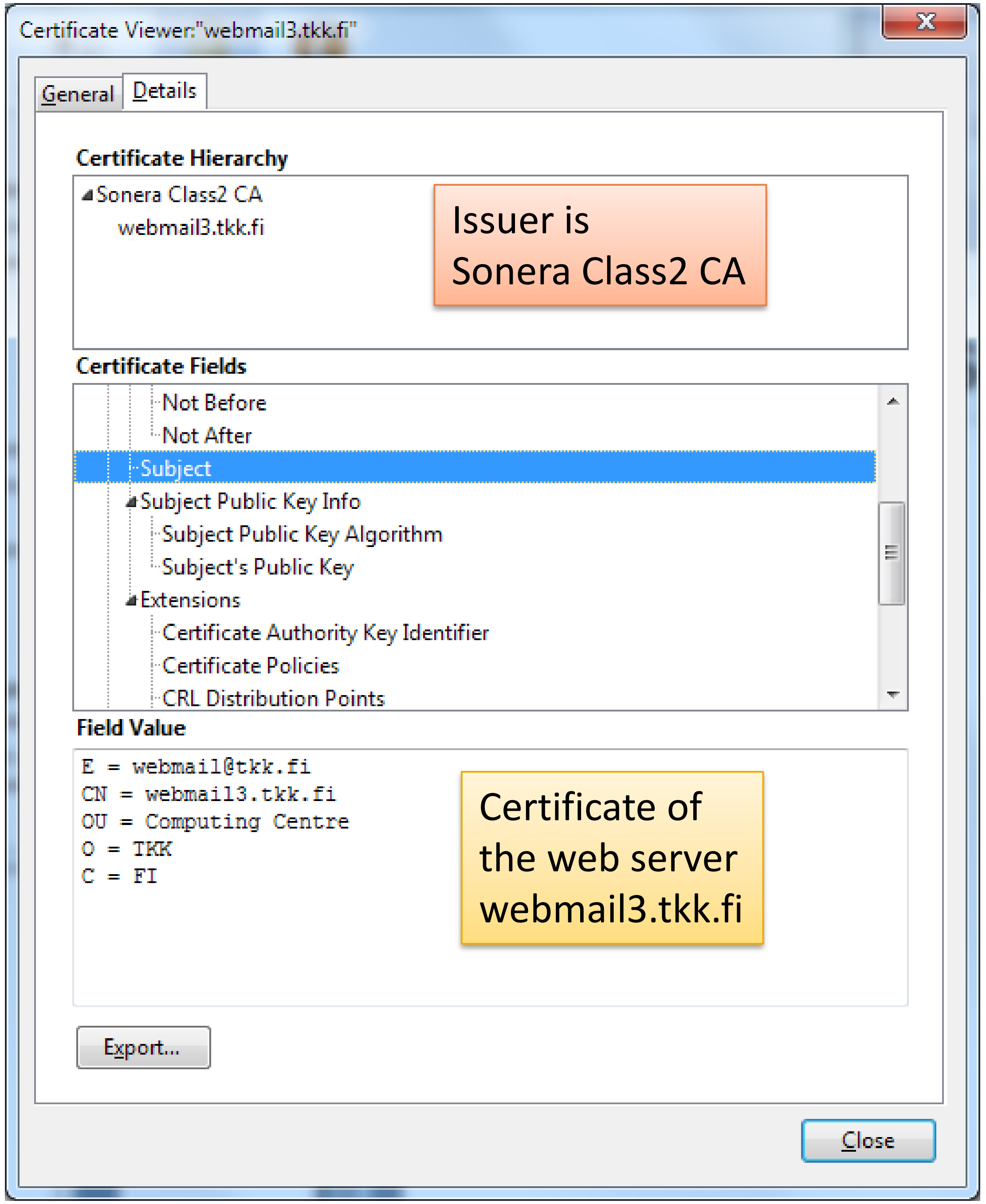
- Windows 10:
  - Manage user certificates / Manage computer certificates
  - Some browsers maintain their own list (e.g. Firefox)
- In Linux, the location varies, e.g. `/usr/share/ca-certificates/mozilla/`


certmgr - [Certificates - Current User\Trusted Root Certification Authorities\Certificat...

Certificates - Current User
- Personal
- Trusted Root Certification Aut
  - Certificates
- Enterprise Trust
- Intermediate Certification Aut
- Active Directory User Object
- Trusted Publishers
- Untrusted Certificates
- Third-Party Root Certification
- Trusted People
- Client Authentication Issuers
- Other People
- Smart Card Trusted Roots

Issued To	Issued By
Microsoft Root Certificate Autho...	Microsoft Root Certifica
Microsoft Root Certificate Autho...	Microsoft Root Certifica
Microsoft Time Stamp Root Cert...	Microsoft Time Stamp R
NO LIABILITY ACCEPTED, (c)97 Ve...	NO LIABILITY ACCEPTED,
QuoVadis Root CA 2	QuoVadis Root CA 2
QuoVadis Root CA 2 G3	QuoVadis Root CA 2 G3
rootca.tahma.fi	rootca.tahma.fi
rootca.tahma.fi	rootca.tahma.fi
SecureTrust CA	SecureTrust CA
Sonera Class2 CA	Sonera Class2 CA
Starfield Class 2 Certification Aut...	Starfield Class 2 Certifica
Starfield Root Certificate Authori...	Starfield Root Certificate
Symantec Enterprise Mobile Ro...	Symantec Enterprise Mo
TeliaSonera Root CA v1	TeliaSonera Root CA v1
thawte Primary Root CA	thawte Primary Root CA

Trusted Root Certification Authorities store contains 59 certificates.


Certificate Viewer:"webmail3.tkk.fi"

General | Details

Certificate Hierarchy

- Sonera Class2 CA
  - webmail3.tkk.fi

Issuer is Sonera Class2 CA

Certificate Fields

- Not Before
- Not After
- Subject
- Subject Public Key Info
  - Subject Public Key Algorithm
  - Subject's Public Key
- Extensions
  - Certificate Authority Key Identifier
  - Certificate Policies
  - CRL Distribution Points

Field Value

```
E = webmail@tkk.fi
CN = webmail3.tkk.fi
OU = Computing Centre
O = TKK
C = FI
```

Certificate of the web server webmail3.tkk.fi

What does TLS achieve?

Thanks to the trust chain, I know that this server really is webmail3.tkk.fi

Sonera root CA was not pre-installed in the browser; so I downloaded the self-signed certificate from the web (insecurely) and added it to the list of trusted root CAs

How do I know that the webmail server should have the name webmail**3**?

(An old but enlightening example)

TLS session protocol

- After the handshake, data is protected with the session protocol
- Data confidentiality is protected with symmetric encryption, e.g. AES in CBC mode
- Data integrity is protected with message authentication codes (MAC)
- Secret session keys for encryption and authentication in each direction are derived from the master_secret

SUMMARY

Some lessons

- Cryptography turns a security problem into a key distribution problem
- PKI turns a key distribution problem into a naming problem

  – How do I know the name of the server/client/user that I need to talk with?
  – Does a name (or identifier) uniquely identify the intended entity?
  – Who is the authority that assigns or certifies names?

Avoiding common mistakes

!

Some facts to avoid surprisingly common misconceptions:

- Certificate is NOT “*encrypted* with the CA private key”
  - There is no encryption or secrets in the certificate
  - The certificate is *signed* with the CA private key and *verified* with the CA public key
- Certificates are NOT retrieved from the CA on demand
  - Instead, the subject stores the certificate chain for the validity period (e.g., one year) and presents it to verifiers
  - However, the verifier retrieves the certificate *revocation list* on demand and then caches it (e.g., for a day), or the verifier queries the OCSP server on demand
- The certificate *alone* does NOT prove anyone’s identity
  - Certificate is public information that can be copied
  - Certificate + signature *together* can be used for authentication
  - E.g., a signed message with a certificate proves the identity of the sender, and TLS with server certificate proves the identity of the web server

List of key concepts

- Certificate, identity certificate, certification authority CA, issuer, subject, validity, authority vs trusted third party TTP
- Public-key infrastructure PKI, X.509, CA hierarchy, certificate chain or path, root CA, end entity, self-signed certificate, root of trust
- Revocation, certificate revocation list CRL, OCSP, Certificate Transparency
- Transport layer security TLS, SSL, secure socket API, security protocol, secure connection, handshake, Diffie-Hellman, session, trust chain

Exercises

- Set up your own CA with OpenSSL (or a commercial CA implementation if you have access to one) and try to use it for protecting web access; what were the difficult steps?
- What are Extended Validation Certificates, how were they supposed to improve security, and why were they found to not help much?
- Find several web and user certificates and compare the names and certification paths on them
- Why do almost all web sites have certificate chains with a sub-CA, rather than using the root CA directly to sign end-entity certificates?
- What information does the signature on the self-signed root certificate convey? Hint: there is more than just the public key
- Previously, many website front pages were insecure (http) even though the password entry and/or service access were secure (https)? What security problems did this cause?
- What TLS-related actions are required from the *user* when logging into a secure bank web site?
- Learn how *Let's Encrypt* does identity proofing. If you have a domain name and a web server, set it up to use https. Are there any potential weaknesses in the process?
- How should a browser creator select the default root CAs?
- What kind of compromises of CAs have been in the news?

Related reading

- Stallings and Brown: Computer security, principles and practice, 4th ed., 22, 23.2-3
  - other Stallings books have similar sections
- Certificate Transparency: http://www.certificate-transparency.org/what-is-ct
- Survival guides - SSL/TLS and X.509 (SSL) Certificates: http://www.zytrax.com/tech/survival/ssl.html