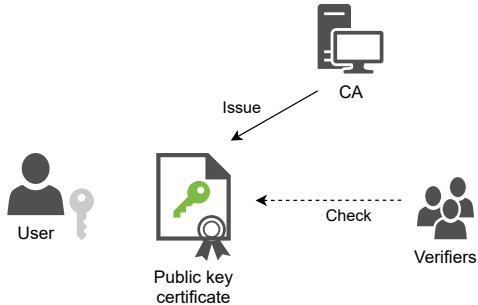
MTAT.07.017
Applied Cryptography

Public key infrastructure (PKI)
Public key certificates (X.509)

University of Tartu

Spring 2023

Public key infrastructure (PKI)



Public key certificate binds a public key to an identity

- Main actors:
  - User (subject, subscriber, end-entity)
  - Certificate authority (trusted third party, issuer)
  - Verifier (relying party)
- The passport analogy

X.509 certificate

Subject: "John Smith"

Issuer: Verisign, Inc. Root CA

Public key: BC F7 C6 74 F5 32 D0 34 ...

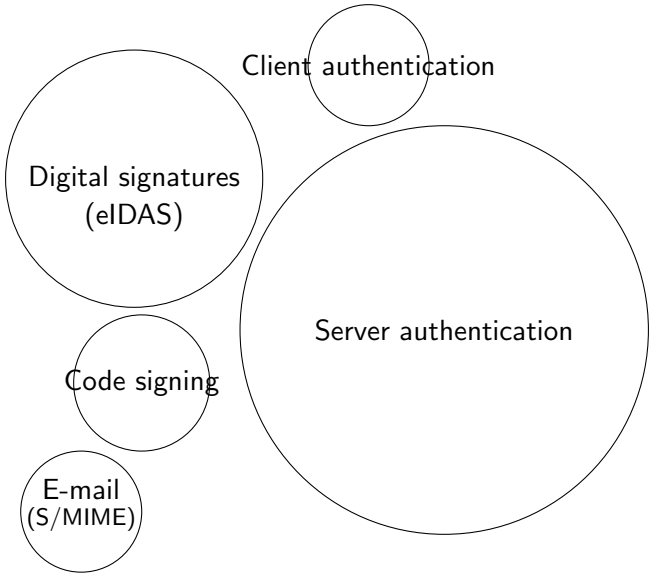
Serial #: 11:21:56:2D:2E

Valid: from 2020.01.01 15:00 to 2022.01.01 15:00

Other data: ...

Signed: CA's Signature

PKI use cases



Certificate Authority (CA)

Certificate Authority – **Trusted** Third Party. Where does the trust come from?

- Software vendors decide on our behalf (Mozilla, Google)
- EU Regulation 910/2014 (eIDAS)

How to get our CA trusted:

- Compliance audit (WebTrust, ETSI TS)
  - Ernst & Young or KPMG (15k EUR/year)
- Liability insurance (required by eIDAS)
  - Insurance industry reluctant (3k EUR/year)
- Use of Hardware Security Module (HSM)


`https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/policy/`

Certificate hierarchy

- Root CA – self-signed certificate (trust anchor)
- CA can delegate trust to subordinate/intermediate CAs
- Useful for risk limitation

X.509 certificate

```
Certificate  ::=  SEQUENCE  {
        tbsCertificate       TBSCertificate,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
        version         [0]  EXPLICIT Version DEFAULT v1(0),
        serialNumber         INTEGER,
        signature            AlgorithmIdentifier,
        issuer               Name,
        validity             Validity,
        subject              Name,
        subjectPublicKeyInfo SubjectPublicKeyInfo,
        extensions      [3]  EXPLICIT Extensions OPTIONAL -- v3(2) only }

Validity ::= SEQUENCE {
        notBefore      UTCTime,
        notAfter       UTCTime }

Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
Extension  ::=  SEQUENCE  {
        extnID      OBJECT IDENTIFIER,
        critical    BOOLEAN DEFAULT FALSE,
        extnValue   OCTET STRING  }
```

http://tools.ietf.org/html/rfc5280

X.509 certificate

- tbsCertificate – DER structure to be signed by CA
- version – X.509v1 or X.509v3 used
  - X.509 v3 introduces certificate extensions
- serialNumber – unique for every certificate issued by CA
- signature – AlgorithmIdentifier from outer Certificate sequence
- issuer – identity of CA who signed the certificate
- validity – period in which certificate should be assumed valid
- subject – identity of a subject whose public key in the certificate
- subjectPublicKeyInfo – subject's public key
- extensions – optional extensions providing more information

Distinguished Name (DN) in X.509 Certificate

The issuer and subject field is defined as the X.501 type Name:

```
Name ::= RDNSequence
RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
RelativeDistinguishedName ::= SET OF AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {
  type     OBJECT IDENTIFIER,
  value    ANY -- DEFINED BY type  }
```

- Yet another notation for unique identifiers
- Used in LDAP and related protocols
- Example: `CN=John Doe, OU=Helpdesk, O=Burgers Inc., C=US`

Distinguished Name (DN) in X.509 Certificate

```
 2  74: SEQUENCE {
 4  11:   SET {
 6   9:     SEQUENCE {
 8   3:       OBJECT IDENTIFIER countryName (2 5 4 6)
13   2:       PrintableString 'US'
      :       }
      :     }
17  18:   SET {
19  16:     SEQUENCE {
21   3:       OBJECT IDENTIFIER organizationName (2 5 4 10)
26   9:       UTF8String 'Burgers Inc.'
      :       }
      :     }
37  20:   SET {
39  18:     SEQUENCE {
41   3:       OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
46  11:       UTF8String 'Helpdesk'
      :       }
      :     }
59  17:   SET {
61  15:     SEQUENCE {
63   3:       OBJECT IDENTIFIER commonName (2 5 4 3)
68   8:       UTF8String 'John Doe'
      :       }
      :     }
      :   }
```

```
(2 5 4 4)                : surname (SN)
(2 5 4 42)               : givenName (GN)
(2 5 4 5)                : serialNumber
(2 5 4 7)                : localityName (L)
(2 5 4 8)                : stateOrProvinceName (ST)
(1 2 840 113549 1 9 1)   : emailAddress
```

Certificate extensions (X.509v3 only)

```
Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
Extension  ::=  SEQUENCE  {
        extnID      OBJECT IDENTIFIER,
        critical    BOOLEAN DEFAULT FALSE,
        extnValue   OCTET STRING  }
```

- Every extension has an OID
- RFC 5280 defines several standard extensions

"Each extension in a certificate is designated as either critical or non-critical. A certificate-using system MUST reject the certificate if it encounters a critical extension 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."

`http://tools.ietf.org/html/rfc5280`

Certificate extensions (X.509v3 only)

- Key usage – limits the purpose of the key contained in the certificate

```
KeyUsage ::= BIT STRING {
       digitalSignature        (0),
       nonRepudiation          (1), -- contentCommitment
       keyEncipherment         (2),
       dataEncipherment        (3),
       keyAgreement            (4),
       keyCertSign             (5),
       cRLSign                 (6),
       encipherOnly            (7),
       decipherOnly            (8) }
```

  - If extension is not present the key may be used for all purposes
- Extended key usage – indicates a more specific purpose of the key

```
ExtKeyUsageSyntax ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId
KeyPurposeId ::= OBJECT IDENTIFIER
id-kp-serverAuth      OBJECT IDENTIFIER ::= { 1 3 6 1 5 5 7 3 1 }
id-kp-clientAuth      OBJECT IDENTIFIER ::= { 1 3 6 1 5 5 7 3 2 }
id-kp-codeSigning     OBJECT IDENTIFIER ::= { 1 3 6 1 5 5 7 3 3 }
id-kp-emailProtection OBJECT IDENTIFIER ::= { 1 3 6 1 5 5 7 3 4 }
```

  - Usage must be consistent with the key usage extension

Certificate extensions (X.509v3 only)

- Basic constraints – identifies whether subject is CA
  - For CA certificate identifies maximum subordinate CAs it may have

```
id-ce-basicConstraints OBJECT IDENTIFIER ::=  { id-ce 19 }
BasicConstraints ::= SEQUENCE {
   cA                  BOOLEAN DEFAULT FALSE,
   pathLenConstraint   INTEGER (0..MAX) OPTIONAL }
```

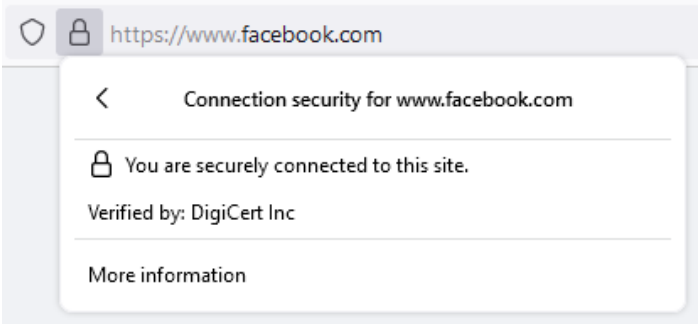
  - If `cA` is `TRUE`, the key usage extension must be absent or must have `keyCertSign` bit set
- Certificate policies – contains pointer to policy information
  - URL to certificate practice statement (CPS)
  - OID of the CPS document version
  - Explicit notice text

Certificate extensions (X.509v3 only)

- Subject alternative name
  - Used to alternatively identify a subject
  - Can include email, DNS name, IP addresses, URI, etc.
  - Latest standards promote use of this extension
- Authority key identifier and subject key identifier
  - Uniquely identifies subject and issuer
- CRL distribution points
  - Includes URI where CRL is available (HTTP or LDAP)
- Authority information access
  - Indicates how to access information about CA services
- Subject information access
  - Indicates how to access information about subject

Extensions may include a picture of the subject, attributes, roles, etc.

Use in HTTPS (TLS)



- TLS server certificates – the most popular use case
- What does the browser verify before the connection is considered secure?
  - Certificate signed by a trusted CA
  - Host name in the address bar matches the CN in the certificate
  - Validity date, extensions, etc.

Server certificate

Certificate

*.facebook.com | DigiCert SHA2 High Assurance Server CA | DigiCert High Assurance EV Root CA

Subject Name

Country	US
State/Province	California
Locality	Menlo Park
Organization	Facebook, Inc.
Common Name	*.facebook.com

Issuer Name

Country	US
Organization	DigiCert Inc
Organizational Unit	www.digicert.com
Common Name	DigiCert SHA2 High Assurance Server CA

Validity

Not Before	Tue, 12 Jul 2022 00:00:00 GMT
Not After	Mon, 10 Oct 2022 23:59:59 GMT

Subject Alt Names

DNS Name	*.facebook.com
DNS Name	*.facebook.net
DNS Name	*.fbcdn.net
DNS Name	*.fbsbx.com
DNS Name	*.m.facebook.com
DNS Name	*.messenger.com
DNS Name	*.xx.fbcdn.net
DNS Name	*.xy.fbcdn.net
DNS Name	*.xz.fbcdn.net
DNS Name	facebook.com
DNS Name	messenger.com

Public Key Info

Server certificate

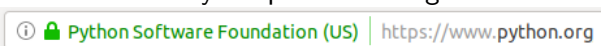
```
$ openssl x509 -in facebook-com.pem -text
Version: 3 (0x2)
Serial Number:
    01:6e:40:cc:d8:87:80:9d:b6:21:30:0e:3b:e0:b6:d1
Signature Algorithm: sha256WithRSAEncryption
Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assurance Server CA
Validity
    Not Before: Jul 12 00:00:00 2022 GMT
    Not After : Oct 10 23:59:59 2022 GMT
Subject: C = US, ST = California, L = Menlo Park, O = "Facebook, Inc.", CN = *.facebook.com
Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
        Public-Key: (256 bit)
        pub:
            04:43:82:04:65:9a:86:61:6f:53:bf:48:a1:8a:8a:
            7f:76:be:d1:e8:89:32:ac:a0:68:61:3c:71:5b:18:
            73:9d:d7:4e:41:e5:bf:3b:51:44:5c:6d:36:d8:25:
            39:01:68:cd:f1:64:34:66:00:0d:c0:ed:ab:5e:59:
            b0:8d:7c:5b:73
        ASN1 OID: prime256v1
        NIST CURVE: P-256
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Key Usage: critical
        Digital Signature
    X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
    X509v3 Subject Alternative Name:
        DNS:*.facebook.com, DNS:*.facebook.net, DNS:*.fbcdn.net, DNS:*.fbsbx.com, DNS:*.m.facebook.com
Signature Algorithm: sha256WithRSAEncryption
     6a:f3:8f:ad:44:76:ae:a3:7f:80:56:51:3e:03:3b:0d:7b:2e:
```

Identity verification

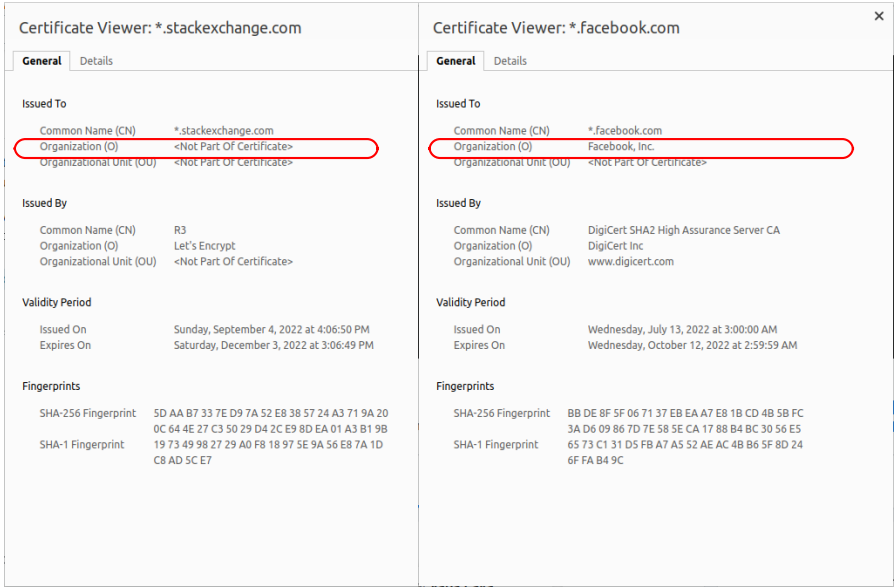
- Domain Validation (DV): ~~$20/year~~ $0/year
- Checks whether you control the domain

- Organization Validation (OV): $100/year
- Checks whether you operate the organization

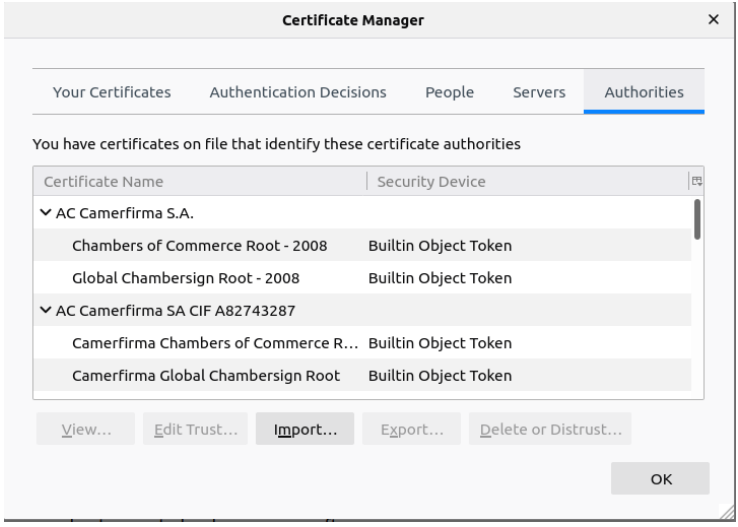
- Extended Validation (EV): $200/year
- Checks whether you operate the organization x2



Domain Validation (DV) vs Organization Validation (OV)

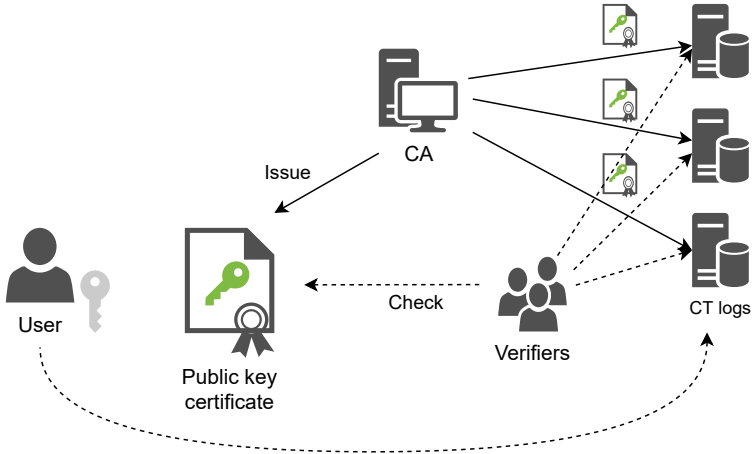


Trusted CAs



https://ccadb-public.secure.force.com/mozilla/CACertificatesInFirefoxReport

Certificate Transparency (CT)



`https://crt.sh/`

Certificate signing request (CSR)

```
$ openssl genpkey -algorithm RSA -out priv.pem -pkeyopt rsa_keygen_bits:2048
$ openssl req -new -key priv.pem -out example.com.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:.
Organization Name (eg, company) [Internet Widgits Pty Ltd]:.
Organizational Unit Name (eg, section) []:.
Common Name (e.g. server FQDN or YOUR name) []:example.com
Email Address []:.

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:asdasd
An optional company name []:.
```

```
$ cat example.com.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICfzCCAWcCAQAwIzELMAkGA1UEBhMCVVMxFDASBg
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQ
FqwKhDHTcgyHWCNgxBI7L7e9IsQ6E9pfUUSdaEHfFr
PEuUNwmobGw4oYS/vUv6Ua0/YTqitVu2M9N4bU/Yk5
xzASSJ4qKxOF543RmKivpveiE7aoMkRxnQ5sh2nWjz
ulkxBIGo9sQysIGVpr9TDJUPiqPeTyik2fRKO6umA0
iVkqHI17FoLI+lflozWdkfM7oBFDu3L3DYe6ZkuxwS
HMQVeZQfNUQNNPSO9lotXywOE2inAo9dgCajPnioK6
KBpgqzAkff7H9mQb72a2MVYTESo9jJt4Xu1DJuW+PV
fYrlUMIRiKyH9glbq1cnbtPpqQ==
-----END CERTIFICATE REQUEST-----
```

Certificate signing request (CSR)

```
$ openssl req -in example.com.csr -text
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = US, CN = example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:d0:0b:d6:34:0a:6e:cc:38:fb:7d:fd:5f:5a:a9:
                    88:16:ac:0a:84:31:d3:72:0c:87:58:23:60:c4:12:
                    3b:2f:b7:bd:22:c4:3a:13:da:5f:51:44:9d:68:41:
                    df:16:7b:b1:ad:55:98:33:be:fa:61:68:45:4e:8e:
                    01:7a:e0:ae:3c:4b:94:37:09:a8:6c:6c:38:a1:84:
                    2a:1c:8d:7b:16:82:c8:fa:57:e5:a3:35:9d:91:f3:
                    3b:a0:11:43:bb:72:f7:0d:87:ba:66:4b:b1:c1:22:
                    87:a7:92:6a:72:09:e8:09:27:dc:a9:06:24:d0:74:
                    87:03
                Exponent: 65537 (0x10001)
        Attributes:
            challengePassword        :asdasd
    Signature Algorithm: sha256WithRSAEncryption
         1a:23:45:c1:6e:ba:57:49:de:73:28:00:bd:8c:1e:e8:29:fa:
         21:d9:7d:1a:22:db:56:fe:1a:93:e1:e9:0e:82:d0:05:47:e9:
         f3:93:b4:89:f4:4d:7a:fc:24:55:46:58:e0:b3:3d:80:ad:01:
```

Certificate signing request (CSR)

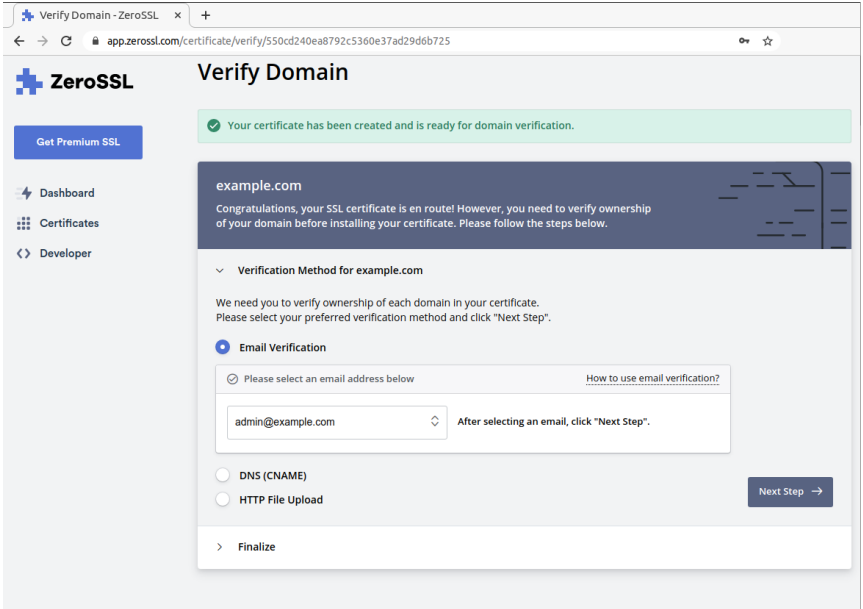
```
$ openssl req -in example.com.csr -outform der -out example.com.csr.der
$ dumpasn1 example.com.csr.der
  0 639: SEQUENCE {
  4 359:   SEQUENCE {
  8   1:     INTEGER 0
 11  35:     SEQUENCE {
 13  11:       SET {
 15   9:         SEQUENCE {
 17   3:           OBJECT IDENTIFIER countryName (2 5 4 6)
 22   2:           PrintableString 'US'
        :           }
        :         }
 26  20:       SET {
 28  18:         SEQUENCE {
 30   3:           OBJECT IDENTIFIER commonName (2 5 4 3)
 35  11:           UTF8String 'example.com'
        :           }
        :         }
        :       }
 48 290:     SEQUENCE {
 52  13:       SEQUENCE {
 54   9:         OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
 65   0:         NULL
        :         }
 67 271:       BIT STRING, encapsulates {
 72 266:         SEQUENCE {
 76 257:           INTEGER
                     00 D0 0B D6 34 0A 6E CC 38 FB 7D FD 5F 5A A9 88
                     [ Another 129 bytes skipped ]
337   3:           INTEGER 65537
        :           }
        :         }
        :       }
342  23:     [0] {
344  21:       SEQUENCE {
346   9:         OBJECT IDENTIFIER challengePassword (1 2 840 113549 1 9 7)
357   8:         SET {
359   6:           UTF8String 'asdasd'
        :           }
        :         }
        :       }
        :     }
367  13:   SEQUENCE {
369   9:     OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840 113549 1 1 11)
```

```
CertificationRequest ::= SEQUENCE {
   certificationRequestInfo CertificationRequestInfo,
   signatureAlgorithm AlgorithmIdentifier,
   signature          BIT STRING
}

CertificationRequestInfo ::= SEQUENCE {
   version       INTEGER v1(0),
   subject       Name,
   subjectPKInfo SubjectPublicKeyInfo,
   attributes    [0] IMPLICIT Attributes
}
```

PKCS#10: https://tools.ietf.org/html/rfc2986

Certificate enrollment



Task: Certificate issuer

Implement a utility that issues a TLS server certificate based on a certificate signing request.

```
$ ./issue_cert.py
usage: issue_cert.py CA_cert_file CA_private_key_file csr_file output_cert_file

$ ./issue_cert.py CA_cert.pem CA_priv.pem example.com.csr issued.pem
[+] Issuing certificate for "example.com"

$ openssl verify -CAfile CA_cert.pem -purpose sslserver issued.pem
issued.pem: OK

$ openssl verify -CAfile CA_cert.pem -purpose smimesign issued.pem
CN = example.com
error 26 at 0 depth lookup: unsupported certificate purpose
error issued.pem: verification failed
```

- Must support PEM/DER inputs, PEM output
- Sign subject's certificate using the CA's private key
- Use `sha256WithRSAEncryption` algorithm (1.2.840.113549.1.1.11)

Task: Certificate issuer

- Specify any certificate serial number you want
- Validity dates at least 3 month apart from today (may hardcode)
- Fetch issuer's distinguished name from CA certificate
- Copy only the CN from the subject's CSR DN (other fields must be ignored!)
- Fetch subject's public key from CSR (subjectPublicKeyInfo)
- Certificate extensions (critical:TRUE):
  - basic constraints CA:FALSE
  - key usage: `digitalSignature`
  - extended key usage: `id-kp-serverAuth`
- No need to verify CSR's signature
- Use your own DER encoder and `pyasn1` for decoding

Task: Certificate issuer

```
$ openssl x509 -in issued.pem -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4138208570 (0xf6a80d3a)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = Trustworthy Inc, OU = IT dep, CN = Trustworthy Root CA
        Validity
            Not Before: Jan  1 00:00:00 2022 GMT
            Not After : Jan  1 00:00:00 2023 GMT
        Subject: CN = example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c1:2b:d7:0b:98:b0:61:22:b1:92:e7:57:fe:4e:
                    c1:f5:1c:01:38:da:a0:dd:03:ed:ed:72:30:df:09:
                    13:34:1f:e7:85:c9:dc:0d:9b:7d:83:6a:e5:37:60:
                    8c:7a:13:95:f5:5d:15:e1:aa:8d:e5:f3:30:4d:fd:
                    20:f1
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage: critical
                TLS Web Server Authentication
    Signature Algorithm: sha256WithRSAEncryption
         1a:3e:98:70:9f:00:5f:fd:d9:7b:62:2d:f6:b0:39:2f:ce:dd:
         91:c6:45:b9:a8:65:16:0e:a8:70:75:64:51:f3:22:28:0f:15:
         23:b8:0c:ce:79:fb:c8:f8:ac:e0:e0:a4:65:24:a9:d3:51:c6:
         1b:e7:36:6a:8a:64:71:59:5a:fc:aa:31:69:d9:82:8c:d5:40:
```

Task: Hints

- `pyasn1` will fail to decode CSR if it contains no attributes (challenge password) since it expects implicit tagging:
  - Make sure your CSR contains a challenge password
- `pyasn1` can easily encode decoded substructures:
  `encoder.encode(decoder.decode(der)[0][0][5])`
- You may want to implement `asn1_bitstring_der()` which takes a byte string as input (padding always 0x00)
- Read ASN.1 definitions or `dumpasn1` example certificates to find out the encoding of the certificate and its extensions

  ```
  openssl x509 -inform pem -in cert.pem -outform der -out cert.der
  ```

- For debugging use two windows to compare your `dumpasn1` output with reference output

Trust on first use (TOFU)

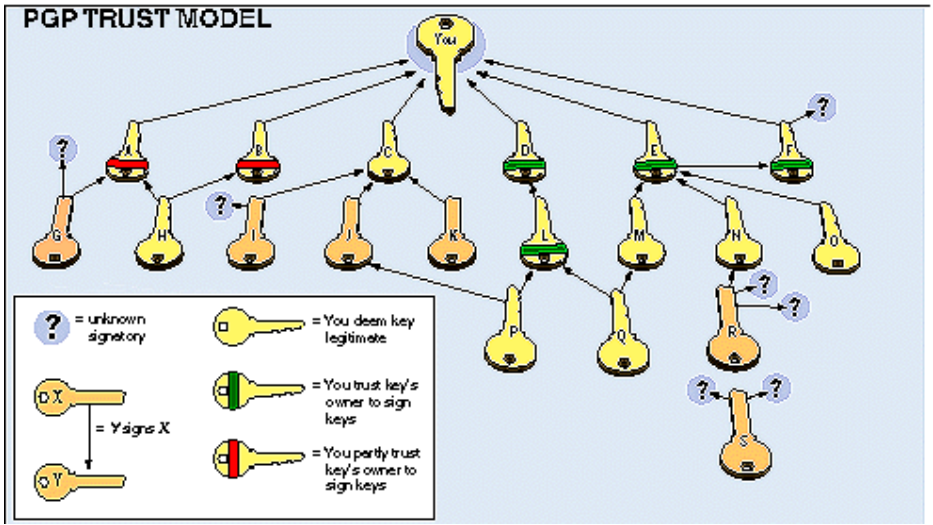
```
$ ssh user@example.com
The authenticity of host 'example.com (93.184.216.34)' can't be established.
RSA key fingerprint is SHA256:2x7va2E9JDr1xwWRemr5gYQrguFjBGikei9bXDi6K44.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'example.com,93.184.216.34' (RSA) to the list of known hosts.
user@example.com's password:

$ cat ~/.ssh/known_hosts
example.net,93.184.216.33 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEA2SGQCzV/vcs1su6eSM52Skksu2n9J3zdFjmSfgB
example.com,93.184.216.34 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA+EuCKTMZU9LYhNqBLfz8KGgqvLv90wiadUOAHv2

$ ssh user@example.com
user@example.com's password:

$ ssh user@example.com
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:4JlsTx1vbxHYaF6ALHD/dTkbX5N6ViZZQtNItAKdO4k.
Please contact your system administrator.
```

Web of trust (WOT)



Questions

- What does PKI and X.509 certificates solve?
- Which are the two most important fields in the X.509 certificate?
- Who defines trusted CAs for digital signature certificates?
- What is the Hardware Security Module useful for?
- What does the browser check in a certificate received from the server?
- Who defines trusted CAs for web server certificates?
- How are DV certificates different from OV certificates?
- How does a CA verify whether an entity owns the domain?