MTAT.07.017
Applied Cryptography

Public Key Infrastructure (PKI)
Public Key Certificates (X.509)

University of Tartu

Spring 2015
Key Management

- The hardest problem
- How to obtain the key of the other party?
  - Symmetric encryption?
    - Vulnerable to passive and active attacks
    - Confidential and authentic channel needed
  - Asymmetric encryption?
    - Vulnerable to active attacks
    - Authentic channel needed
- Trust models:
  - Trust on first use (e.g., SSH)
  - Decentralized model - web of trust (e.g., PGP)
  - Centralized model - Trusted third party (e.g., TLS)
TOFU: Trust On First Use

• Used by SSH (encrypted telnet)

• For the first time:

$ ssh user@cs.ut.ee
The authenticity of host 'cs.ut.ee (193.40.36.81)' can't be established.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'cs.ut.ee,193.40.36.81' (RSA) to the list of
known hosts.
user@cs.ut.ee’s password:

$ cat ~/.ssh/known_hosts
cs.ut.ee,193.40.36.81 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA2HotF0bR9U8MgTE67bGJr
math.ut.ee,193.40.36.2 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAzPcVb60Q8QV0s3hdoFaO

• In the future:

$ ssh user@cs.ut.ee
user@cs.ut.ee’s password:
Tofu: Trust On First Use

• If the key has changed:

```
$ ssh user@cs.ut.ee
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@$ WARNING: POSSIBLE DNS SPOOFING DETECTED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The RSA host key for cs.ut.ee has changed, and the key for the corresponding IP address 193.40.36.2 is unchanged. This could either mean that DNS SPOOFING is happening or the IP address for the host and its host key have changed at the same time. Offending key for IP in /home/user/.ssh/known_hosts:2
  remove with: ssh-keygen -f "/home/user/.ssh/known_hosts" -R 193.40.36.2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@$ 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 dd:0b:e5:17:f9:9d:41:24:49:df:bc:54:37:ab:54:7e.
Please contact your system administrator.
```

• Threat model?
• How to improve ssh client to be even more secure?
WOT: Web of trust
PKI: Public key infrastructure

Public key certificate binds a public key with an identity

- Main actors:
  - Subject (user, end-entity)
  - Trusted third party (certificate authority, issuer)
  - Relying party (verifier)

- The passport analogy
Certificate Authorities

Certificate Authority – **Trusted** Third Party

Where the trust comes from?

- Software vendors decided on your behalf (MS, Mozilla)

• Root CA – self-signed certificate (trust anchor)
• CA can delegate trust to subordinate/intermediate CAs
• CA can constrain issued CA and end-entity certificates
Estonian ID card case

- Subject – Estonian residents
- Registration Authority – Police and Border Gate Board
- Manufacturer – Trüb Baltic AS
  - In addition generates RSA key pair for you
- Certificate Authority – AS Sertifitseerimiskeskus
- Relying party – anyone (e.g., your bank)
How to become a CA?

• The objective: profit (Mark Shuttleworth, Thawte (VeriSign), $575 million, Ubuntu)
• Get your root CA trusted:
  • Compliance audit (WebTrust, ETSI TS)
    • Ernst & Young or KPMG (15k EUR/year)
  • Liability insurance (required by EU directive)
    • Insurance industry reluctant (3k EUR/year)
  • Use of Hardware Security Modules (HSM)

http://bugzilla.mozilla.org/show_bug.cgi?id=414520
Hardware Security Module (HSM)

- Physically protected private key storage
- Smart card (miniHSM)

- Certifications:
  - Common Criteria
  - FIPS 140-2:
    - Level 1 – no protection
    - Level 2 – tamper evident
    - Level 3 – tamper resistant
    - Level 4 – tamper reactive
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  }

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 certificate
- subjectPublicKeyInfo – subject’s public key
- extensions – optional extensions providing more information
Distinguished Name (DN) in X.509 Certificate

"The issuer and subject field MUST contain a non-empty distinguished name (DN). The issuer field is defined as the X.501 type Name. Name is defined by the following ASN.1 structures:"

\[
\text{Name ::= \text{RDNSequence}} \\
\text{RDNSequence ::= SEQUENCE OF RelativeDistinguishedName} \\
\text{RelativeDistinguishedName ::= SET OF AttributeTypeAndValue} \\
\text{AttributeTypeAndValue ::= SEQUENCE \{} \\
\text{\quad type \text{ OBJECT IDENTIFIER,}} \\
\text{\quad value \text{ ANY -- DEFINED BY type}} \text{ \}}
\]

- Yet another notation for unique identifiers
- Used in LDAP and related protocols
- Example:

CN=John Doe, OU=Infernal IT, O=Evil 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 ’Evil Inc.’
    :     }
    :     }
37  20:   SET {
39  18:     SEQUENCE {
41  3:       OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
46 11:       UTF8String ’Infernal IT’
    :     }
    :     }
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 its OID
- RFC 5280 defines several standard extensions
- Certificate (path) validation algorithm must handle those

"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."

Certificate Extensions (X.509v3 only)

- **Key Usage**
  - Defines the purpose of the key contained in the certificate

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

  - key may be used for all purposes if extension absent

- **Extended Key Usage**
  - Indicates more specific purpose of the key
  - Usage must be consistent with Key Usage extension

  \[
  \text{ExtKeyUsageSyntax} ::= \text{SEQUENCE SIZE (1..MAX) OF KeyPurposeId} \\
  \text{KeyPurposeId} ::= \text{OBJECT IDENTIFIER} \\
  \text{id-kp-serverAuth} \ \text{OBJECT IDENTIFIER} ::= \{ 1 3 6 1 5 5 7 3 1 \} \\
  \text{id-kp-clientAuth} \ \text{OBJECT IDENTIFIER} ::= \{ 1 3 6 1 5 5 7 3 2 \} \\
  \text{id-kp-codeSigning} \ \text{OBJECT IDENTIFIER} ::= \{ 1 3 6 1 5 5 7 3 3 \} \\
  \text{id-kp-emailProtection} \ \text{OBJECT IDENTIFIER} ::= \{ 1 3 6 1 5 5 7 3 4 \}
  \]
Certificate Extensions (X.509v3 only)

- **Basic Constraints**
  - Identifies whether subject is CA – may sign certificates
  - For CA 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 not present in v3 certificate then not a CA
  - If cA boolean is TRUE then keyUsage must be absent or must have keyCertSign bit set

- **Name Constraints**
  - In CA certificate indicates a name space constraint in all subsequent certificates
  - ”.example.com” matches both host.example.com and my.host.example.com
  - Must be marked critical
  - Pain to process – not used in practice
Certificate Extensions (X.509v3 only)

- **Certificate Policies**
  - Contains policy information terms under which the certificate has been issued and the purposes for which the certificate may be used
    - URL to certificate practice statement (CPS)
    - OID of the CPS document version
    - Explicit notice text

- **Policy Mappings**
  - Maps equivalent policy OIDs
  - Pain to process – not used in practice

- **Policy Constraints**
  - Constrain policies that may be included in CA issued certificates
  - Pain to process – not used in practice
Certificate Extensions (X.509v3 only)

- **Subject Alternative Name**
  - Identifies subject alternatively to the subject name
  - Include email, DNS name, IP addresses, URI, etc.
  - New standards promote use of this extension

- **Authority Key Identifier and Subject Key Identifier**
  - Uniquely identifies subject and issuer
  - KeyIdentifier, GeneralNames, CertificateSerialNumber

- **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.
TOFU Fallback
### Server Certificate

This certificate has been verified for the following uses:

<table>
<thead>
<tr>
<th>SSL Server Certificate</th>
</tr>
</thead>
</table>

#### Issued To
- Common Name (CN): sso.ut.ee
- Organization (O): University of Tartu
- Organizational Unit (OU): ITO
- Serial Number: 16:49

#### Issued By
- Common Name (CN): KLASS3-SK 2010
- Organization (O): AS Sertifitseerimiskeskus
- Organizational Unit (OU): Sertifitseerimisteenused

#### Validity
- Issued On: 09/09/2011
- Expires On: 10/03/2012

#### Fingerprints
Server Certificate

Certificate Hierarchy
- Juur-SK
  - KLASS3-SK 2010
    - sso.ut.ee

Certificate Fields
- sso.ut.ee
  - Certificate
    - Version
    - Serial Number
    - Certificate Signature Algorithm
  - Issuer
  - Validity
    - Not Before
    - Not After
  - Subject
  - Subject Public Key Info
Field Value
- CN = KLASS3-SK 2010
- OU = Sertifitseerimisteenused
- O = AS Sertifitseerimiskeskus
- C = EE
Server Certificate

$ openssl x509 -in sso.ut.crt -text
Version: 3 (0x2)
Serial Number: 5705 (0x1649)
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=EE, O=AS Sertifitseerimiskeskus, OU=Sertifitseerimisteenused, CN=KLASS3-SK 2010

Validity
    Not Before: Sep 9 12:56:30 2011 GMT
    Not After : Oct 3 12:56:00 2012 GMT
Subject: C=EE, ST=Tartumaa, L=Tartu, O=University of Tartu, OU=ITO, CN=sso.ut.ee

Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (2048 bit)
    Modulus:
        Exponent: 65537 (0x10001)

X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    X509v3 Certificate Policies:
        Policy: 1.3.6.1.4.1.10015.7.1.2.2
        CPS: http://www.sk.ee/cps
    X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
    X509v3 Extended Key Usage:
        TLS Web Server Authentication
    X509v3 Authority Key Identifier:
    X509v3 CRL Distribution Points:
        Full Name:
            URI:http://www.sk.ee/crls/klass3/klass3-2010.crl
    X509v3 Subject Key Identifier:
Signature Algorithm: sha1WithRSAEncryption
9f:dd:3e:1a
Use in HTTPS (TLS)

- Requesting party (website owner):
  - Key generation
  - Certificate request submission
- Certificate Authority (CA):
  - Distribution of root certificates
  - Requesting party identity verification
  - Certificate signing (issuance)
- Relying party (website visitor):
  - Certificate verification
## Certificate Authorities

![Certificate Manager](image.png)

You have certificates on file that identify these certificate authorities:

<table>
<thead>
<tr>
<th>Certificate Name</th>
<th>Security Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>AffirmTrust Networking</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>America Online Inc.</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>America Online Root Certification Authority</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>America Online Root Certification Authority</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>AS Sertifitseerimiskeskus</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>Juur-SK</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>KLASS3-SK</td>
<td>Software Security Device</td>
</tr>
<tr>
<td>KLASS3-SK 2010</td>
<td>Software Security Device</td>
</tr>
<tr>
<td>Autoridad de Certificacion Firmaprofesio</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>Autoridad de Certificacion Firmaprofesio</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>Autoridad de Certificacion Firmaprofesio</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>Baltimore</td>
<td>Built-in Object Token</td>
</tr>
<tr>
<td>Baltimore CyberTrust Root</td>
<td>Built-in Object Token</td>
</tr>
</tbody>
</table>

[View...][Edit Trust...][Import...][Export...][Delete or Distrust...]
Identity Verification

- Domain Validation (DV): $20/year $0/year
  Checks whether you control the domain
  
  ![pilet.ee](https://www.pilet.ee/cgi-bin/splususer/splus)

- Organization Validation (OV): $200/year
  Checks whether you operate the organization
  
  ![eesti.ee](https://www.eesti.ee/est)

- Extended Validation (EV): $500/year
  Checks whether you operate the organization 2x
  
  ![Swedbank AS (EE)](https://www.swedbank.ee/private)
Domain Validated vs Organization Validated

Certificate Viewer: '*.pilet.ee'

This certificate has been verified for the following uses:
SSL Server Certificate

Issued To
Common Name (CN) *.pilet.ee
Organization (O) *.pilet.ee
Organizational Unit (OU) Domain Control Validated

Issued By
Common Name (CN) Go Daddy Secure Certification Authority
Organization (O) GoDaddy.com, Inc.
Organizational Unit (OU) http://certificates.godaddy.com/repository

Validity
Issued On 01/20/2012
Expires On 01/20/2014

Fingerprints

Certificate Viewer: '*.eestl.ee'

This certificate has been verified for the following uses:
SSL Server Certificate

Issued To
Common Name (CN) *.eestl.ee
Organization (O) Estonian Informatics Centre
Organizational Unit (OU) <Not Part Of Certificate>

Issued By
Common Name (CN) Thawte Premium Server CA
Organization (O) Thawte Consulting cc
Organizational Unit (OU) Certification Services Division

Validity
Issued On 02/08/2010
Expires On 03/10/2012

Fingerprints
Certificate Signing Request (CSR)

$ openssl genrsa -out priv.pem 1024
$ openssl req -new -key priv.pem -out sso.ut.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]: EE
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) []: sso.ut.ee
Email Address []:

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

$ cat req.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBZzCBOQIBADAoMQswCQYDVQQGGEwJFRTEZMBcGA1UEAwwQd3d3LmFwcGNyeXB0I2Vmz+8IpKax5en8M29CGwuL4e1Ou6LejVE
-----END CERTIFICATE REQUEST-----
Certificate Signing Request (CSR)

"A certification request consists of a distinguished name, a public key, and optionally a set of attributes, collectively signed by the entity requesting certification. Certification requests are sent to a certification authority, which transforms the request into an X.509 public-key certificate."

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

- Why does a subject have to prove a possession of the corresponding private key?
Certificate Signing Request (CSR)

$ openssl req -in sso.ut.ee.csr -text
Certificate Request:
  Data:
    Version: 0 (0x0)
    Subject: C=EE, CN=sso.ut.ee
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (1024 bit)
      Modulus:
      Exponent: 65537 (0x10001)
    Attributes:
      challengePassword :unable to print attribute
    Signature Algorithm: sha1WithRSAEncryption
      35:44
Certificate Signing Request (CSR)

```bash
$ openssl req -in sso.ut.csr -outform der -out sso.ut.csr.der
$ dumpasn1 sso.ut.csr.der
  0 375: SEQUENCE {
  4 225:  SEQUENCE {
  7  1:   INTEGER 0
 10  33:  SEQUENCE {
 12  11:   SET {
 14   9:    SEQUENCE {
 16   3:     OBJECT IDENTIFIER countryName (2 5 4 6)
 21   2:      PrintableString 'EE'
  25  18:   SET {
 27  16:    SEQUENCE {
 29   3:     OBJECT IDENTIFIER commonName (2 5 4 3)
 34   9:      UTF8String 'sso.ut.ee'
  207 23: [0] {
 209 21:  SEQUENCE {
 211  9:   OBJECT IDENTIFIER challengePassword (1 2 840 113549 1 9 7)
 222  8:    SET {
 224  6:      UTF8String 'asdasd'
  232 13:  SEQUENCE {
 234  9:   OBJECT IDENTIFIER sha1WithRSAEncryption (1 2 840 113549 1 1 5)
 245  0:    NULL
  247 129: BIT STRING
  207 23: [0] {
```
Enrollment: Step 1

Billing Contact Information

The billing contact will receive the receipt for the purchase when a credit card is used.

First Name * Foo
Last Name * Bar
Title
E-mail * [redacted]
Phone Number * +3722934312
Company Name
Address * [redacted] Line 1
[redacted] Line 2 (optional)
City * Tartu
State/Province * Armed Forces Africa, Eu
Postal Code * EE1000
Country * ESTONIA

Back Continue
Enrollment: Step 2

Certificate Signing Request

BEGIN CERTIFICATE REQUEST

[certificate content]

END CERTIFICATE REQUEST
RapidSSL Enrollment

Verify Server URL

The CSR you generated is designed to work with the following URL:

https://sso.ut.ee

If this is not the correct URL (computed from the Common name in the CSR), or if any of the CSR Information below is incorrect, then please generate a new CSR and click the Replace CSR button.

Replace CSR

CSR Information

Common Name: sso.ut.ee
Organization:
Org. Unit:
Locality:
State:
Country: EE

Note: The value for the Common Name must exactly match the name of the server you plan to secure.

Continue
Enrollment: Step 4

Approval of Your Certificate Request

The RapidSSL.com RapidSSL® service relies upon the Subscriber or the Subscriber's authorized administrator to approve all certificate requests for all hosts in the domain. It is important that you select the correct authorized administrator below. By selecting an authorized administrator, you warrant that the individual is authorized to approve the request. Your request for a RapidSSL® server certificate will not be processed beyond this point if you select an incorrect email address.

Registered Domain Contacts

Unfortunately, we were not able to obtain the domain contacts from the registrar. It is possible that the registration information has not yet been published for this domain, or that the registrar is not currently available to RapidSSL.

Alternate Approval Email Addresses

The following approval email addresses can be used. You must make sure that the email account has been set up and is available before you submit this order, or the approval email will not be delivered.

Level 2 Domain Addresses
- admin@ut.ee
- administrator@ut.ee
- hostmaster@ut.ee
- webmaster@ut.ee
- postmaster@ut.ee

Level 3 Domain Addresses
- admin@sso.ut.ee
- administrator@sso.ut.ee
- hostmaster@sso.ut.ee
- webmaster@sso.ut.ee
- postmaster@sso.ut.ee
What if CA goes bad?

- Comodo hack (2011 March)
  - Google, Skype, Mozilla - MITM in Iran
  - Fraudulent certificates revoked

- DigiNotar hack (2011 September)
  - Google wildcard - MITM in Iran
  - Root certificates revoked
  - Went bankrupt

- TrustWave spycerts (2012 February)
  - Subordinate root certificate for client’s DLP
  - Subordinate root certificate revoked
  - Mozilla CA policy changed

- Turktrust mistake (2013 January)
  - Left Basic Constraints cA:TRUE for 2 end-entity certificates
  - Certificates in question revoked
  - Punished by Mozilla/MS/Google by not including new CA

Certificate Authorities too big to fail?
Task 1: Self-signed CA Root Certificate

Implement utility that creates self-signed CA root certificate.

$ ./selfsigned.py
usage: selfsigned.py private_key_file output_cert_file

$ openssl genrsa -out priv.pem 1024
$ ./selfsigned.py priv.pem rootCA.pem
$ openssl verify -check_ss_sig -CAfile rootCA.pem rootCA.pem: OK

- Must support PEM/DER inputs, PEM output
- Signature has to verify successfully
- Use sha1WithRSAEncryption (1.2.840.113549.1.1.5)
- Put subject name that identifies you
- Put whatever serial you want
- Critical extension: basic constraints CA:TRUE
- Critical extension: key usage: keyCertSign, cRLSign
- Certificate must be valid at least ± 3 months
- Use your own DER encoder and pyasn1
$ openssl x509 -in rootCA.pem -text
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
  Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=EE, O=University of Tartu, OU=IT dep, CN=Arnis Root CA
  Validity
  Subject: C=EE, O=University of Tartu, OU=IT dep, CN=Arnis Root CA
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
      Public-Key: (1024 bit)
        Modulus:
        Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Basic Constraints: critical
      CA:TRUE
    X509v3 Key Usage: critical
      Certificate Sign, CRL Sign
  Signature Algorithm: sha1WithRSAEncryption
    f4:c6
Task 2: Certificate Signer (Bonus +4 points)

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

$ ./signcert.py
usage: signcert.py private_key_file CA_cert_file csr_file output_cert_file

$ ./signcert.py priv.pem rootCA.pem req.csr issued.pem
$ openssl verify -CAfile rootCA.pem -purpose sslserver issued.pem
issued.pem: OK
$ openssl verify -CAfile rootCA.pem -purpose smimesign issued.pem
issued.pem: C = EE, CN = www.appcrypto.ee
error 26 at 0 depth lookup:unsupported certificate purpose

- Fetch subject’s CN from CSR (other fields may be arbitrary)
- Fetch subject’s public key from CSR (subjectPublicKeyInfo)
- Fetch issuer’s distinguished name from CA certificate
- Sign subject’s certificate using CA private key
- Critical extensions:
  - basic constraints CA:FALSE
  - key usage: digitalSignature
  - extended key usage: id-kp-serverAuth
- Use your own DER encoder and pyasn1
Task 2: Certificate Signer (Bonus +4 points)

$ openssl x509 -in issued.pem -text
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=EE, O=University of Tartu, OU=IT dep, CN=Arnis Root CA
  Validity
  Subject: C=EE, CN=www.appcrypto.ee
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
      Public-Key: (1024 bit)
        Modulus:
        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: sha1WithRSAEncryption
    34:ca
Hints

• pyasn1 fails to decode CSR since it contains implicit tagging:
  • Use your own ASN1 DER decoder
    • non-implicit tagged substructures can be decoded by pyasn1
  • Use pyasn1 decoder by specifying structure definition
    decoder.decode(substrate, asn1Spec=...)

• You might want to implement asn1_bitstring_der() which takes byte string (instead of bitstring) as input

• Read ASN.1 definitions or dumpasn1 example certificates to find out DER encoding of certificate and its extensions
  openssl x509 -inform pem -in cert.pem -outform der -out cert.der