MTAT.07.017
Applied Cryptography

Certificate Revocation List (CRL)

University of Tartu

Spring 2018

Certificate Validity

It may be required to invalidate a certificate before its expiration.

Examples:

- Private key compromised or lost
- Misissued certificate
- Affiliation has changed

Solution – Certificate Revocation List (CRL):

List of unexpired certificates that have been revoked by CA

- How can the relying party find the CRL?
- How is the integrity of CRL data assured?
- How frequently the CA should issue CRL?
- How frequently the relying party should refresh CRL?
- How can the relying party know that CRL is fresh?
- Legal terms: suspension and revocation

Electronic Identification and Trust Services for Electronic Transactions Act

§ 16. Certificate [..] period of validity
(4) A certificate is valid from the beginning of the period of validity set out in the certificate but *not before* the data of the certificate are entered in a certificate database kept by the issuer of the certificate.
(5) The validity of a certificate ends on the validity end date set out in the certificate *or upon revocation of the certificate*.

§ 17. Suspension of certificates
(1) A trust service provider has the right to suspend a certificate if there is doubt that incorrect data have been entered in the certificate or that it is possible to use the private key corresponding to the public key contained in the certificate *without the consent of the certificate holder*.
(5) E-signatures or e-seals given during the period when a certificate is suspended are *invalid*.

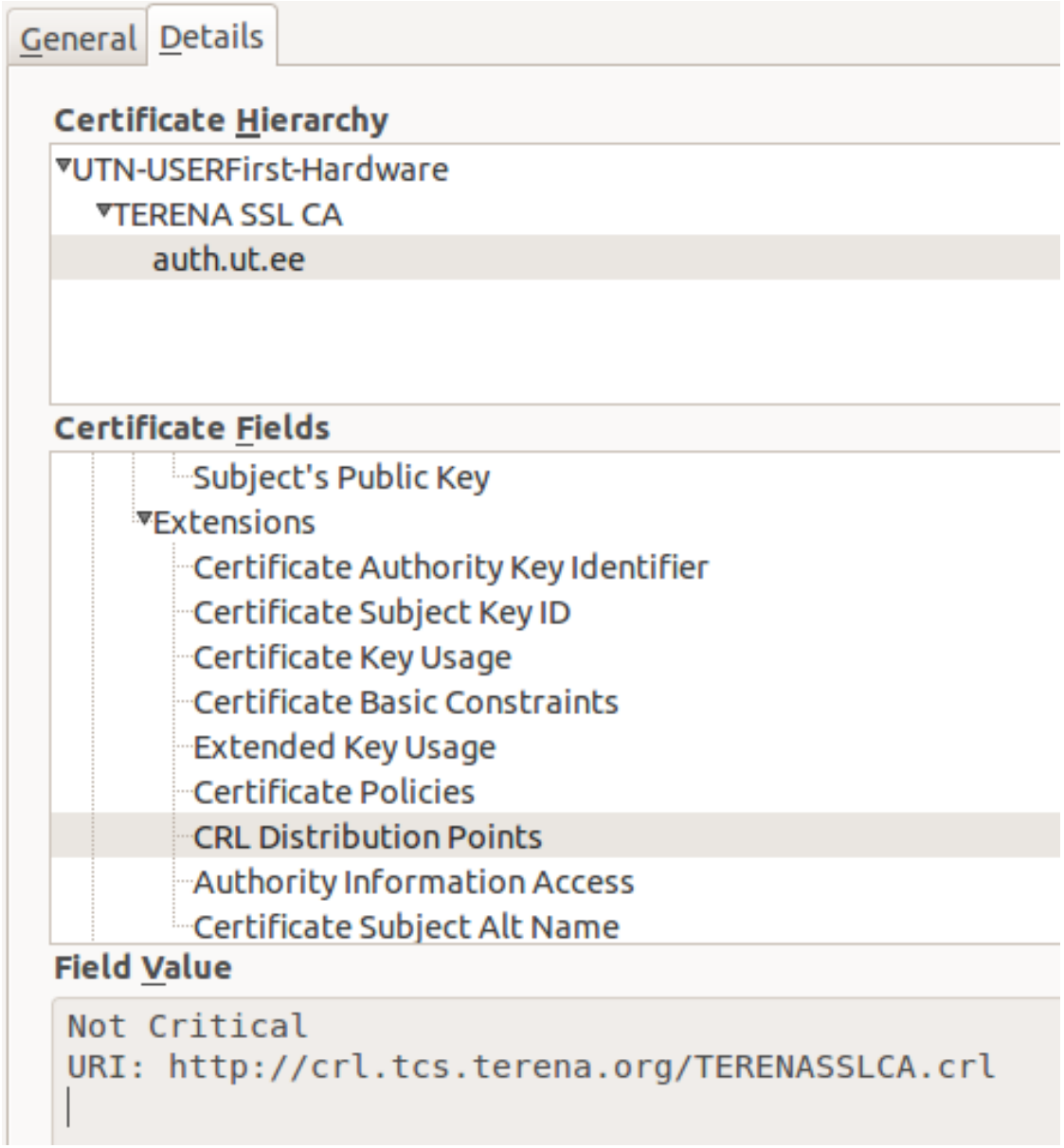
Electronic Identification and Trust Services for Electronic Transactions Act

§ 18. Restoration of validity of suspended certificate
(1) A trust service provider shall restore the validity of a suspended certificate at the request of the certificate holder or a person or authority that applied for the suspension, by entering the information on restoration of validity in the certificate database kept by the trust service provider.

Electronic Identification and Trust Services for Electronic Transactions Act

(4) The following are the bases for revocation of a certificate:
1) request of the certificate holder;
2) a possibility of using the private key corresponding to a public key contained in the certificate without the consent of the certificate holder;
3) appointment of a guardian to the certificate holder to an extent that precludes the use of the certificate;
4) death of the certificate holder or declaration of death of the certificate holder;
7) submission of false data to a trust service service provider by the certificate holder in order to obtain the certificate;
8) termination of provision of trust services;
13) request of a court, the Prosecutor's Office or a pre-trial investigation authority in a criminal matter;
14) expiry or termination of a contract on the basis whereof the certificate is held;
15) other cases provided by law.

CRL Distribution Points



Certificate Revocation List (CRL)

```
CertificateList  ::=  SEQUENCE  {
    tbsCertList           TBSCertList,
    signatureAlgorithm    AlgorithmIdentifier,
    signatureValue        BIT STRING  }

TBSCertList  ::=  SEQUENCE  {
    version             Version OPTIONAL, -- if present, MUST be v2(1)
    signature           AlgorithmIdentifier,
    issuer              Name,
    thisUpdate          UTCTime,
    nextUpdate          UTCTime OPTIONAL,
    revokedCertificates SEQUENCE OF SEQUENCE  {
         userCertificate     CertificateSerialNumber,
         revocationDate      UTCTime,
         crlEntryExtensions  Extensions OPTIONAL -- in v2 }  OPTIONAL,
    crlExtensions       [0] EXPLICIT Extensions OPTIONAL  -- in v2 }
```

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

Certificate Revocation List (CRL)

- tbsCertList – DER structure to be signed by CRL issuer
- version – for v1 absent, for v2 contains 1
  - v2 introduces CRL and CRL Entry extensions
- signature – AlgorithmIdentifier from tbsCertList sequence
- issuer – identity of issuer who issued (signed) the CRL
  - CRL issued not by CA itself – indirect CRL
- thisUpdate – date when this CRL was issued
- nextUpdate – date when next CRL will be issued
- revokedCertificates – list of revoked certificates
  - userCertificate – serial number of revoked certificate
  - revocationDate – time when CA processed revocation request
  - crlEntryExtensions – provides additional revocation information
- crlExtensions – provides more information about CRL

CRL Entry Extensions

Provide methods for associating additional attributes with CRL entries. May be designated as critical or non-critical.

- Reason Code
  - Identifies the reason for the certificate revocation:

```
CRLReason ::= ENUMERATED {
     unspecified             (0),
     keyCompromise           (1),
     cACompromise            (2),
     affiliationChanged      (3),
     superseded              (4),
     cessationOfOperation    (5),
     certificateHold         (6),
          -- value 7 is not used
     removeFromCRL           (8),
     privilegeWithdrawn      (9),
     aACompromise           (10) }
```

- Invalidity Date
  - Provides the date on which the private key was compromised
  - How is it different from the revocation date?
- Certificate Issuer
  - Includes issuer name in case of indirect CRL

CRL Extensions

Provide methods for associating additional attributes with CRLs. May be designated as critical or non-critical.

- CRL Number
  - Monotonically increasing sequence number
  - Essential for delta CRL processing
- Authority Key Identifier
  - Identifies the CA certificate more precisely
- Issuing Distribution Point
  - Identifies the scope for a particular CRL

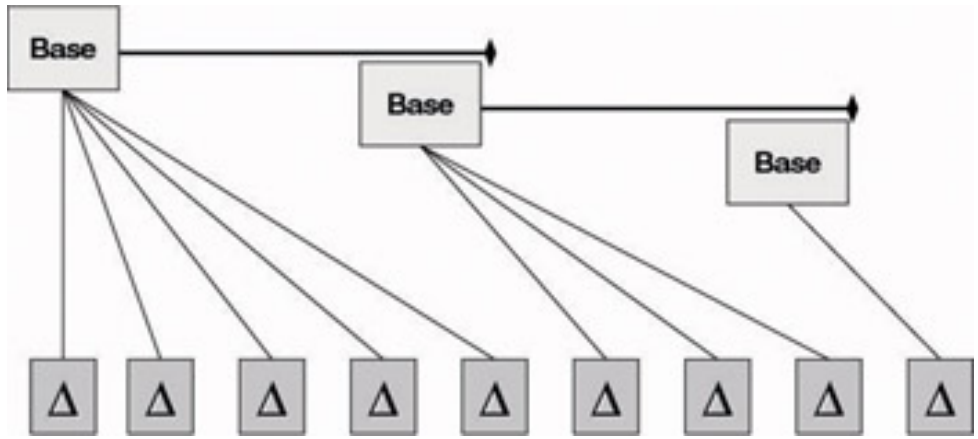
```
IssuingDistributionPoint ::= SEQUENCE {
  distributionPoint          [0] DistributionPointName OPTIONAL,
  onlyContainsUserCerts      [1] BOOLEAN DEFAULT FALSE,
  onlyContainsCACerts        [2] BOOLEAN DEFAULT FALSE,
  onlySomeReasons            [3] ReasonFlags OPTIONAL,
  indirectCRL                [4] BOOLEAN DEFAULT FALSE,
  onlyContainsAttributeCerts [5] BOOLEAN DEFAULT FALSE }
```

  - If absent contains all revoked unexpired certificates

Delta CRLs

CRLs can contain huge list of revoked certificates:
`http://www.sk.ee/crls/esteid/esteid2015.crl` (64MB – 1'365'262)
`http://www.sk.ee/repository/crls/esteid2011.crl` (59MB – 1'256'790)

Delta CRL – CRL that provides changes to base CRL



- Delta CRL Indicator extension
  - Identifies a CRL as being a delta CRL
  - Contains BaseCRLNumber of complete CRL that is being updated
- Delta CRL Distribution Point extension
  - Identifies how delta CRL is obtained
- Complicated to process – not supported widely

Liability Analysis

Let's assume subject's private key has been compromised.

Who (subject, CA or relying party) is liable for actions made:

- in the time period after revocation information has appeared in CRL?
- in the time period after CRL has been issued but not available to relying parties (e.g., CA server downtime)?
- in the time period before next CRL has been issued?
- in the time period before CA has marked the certificate revoked in their internal database?
- in the time period while CA helpline not answering?
- in the time period before CA has been informed about the key compromise?

Code of Civil Procedure

§ 277. Contestation of authenticity of documents
(3) Authenticity of an electronic document bearing a digital signature may be contested only *by substantiating the circumstances* which give reason to presume that the document has not been prepared by the holder of the digital signature.

Certificate Chain

- How to validate a certificate chain?
- Where to look whether the end entity certificate is revoked?
  - CRL issued by intermediate CA (usually every 12h)
  - Grace period
- Where to look whether the intermediate CA is revoked?
  - CRL issued by root CA (usually every 3 month)
  - Grace period?!
- Where to look whether the root CA is revoked?
  - CRL issued by root CA itself (flawed)

Who is liable for actions made after the root CA private key has been compromised?

Hypertext Transfer Protocol (HTTP)

- Application layer client-server, request-response protocol
- Runs over TCP (Transmission Control Protocol) port 80

Client request (`http://kodu.ut.ee/~arnis/`):

```
GET /~arnis/ HTTP/1.1
Host: kodu.ut.ee
Connection: close
```

```
POST /~arnis/ HTTP/1.1
Host: kodu.ut.ee
Content-Length: 24
Connection: close

sending_this_binary_blob
```

Server response:

```
HTTP/1.1 200 OK
Date: Wed, 28 Mar 2018 17:50:07 GMT
Server: Apache/2.0
Content-Length: 48
Connection: close
Content-Type: text/html; charset=ISO-8859-15

<html><body><img src="c.png"></body></html>
```

- Request lines must all end with <CR><LF> ("\r\n")
- Header lines are separated from the body by empty line
- POST requests have non-empty request body

`http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol`

Sockets in Python

```
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> s.connect(("kodu.ut.ee", 80))
>>> s.send("GET /~arnis/ HTTP/1.1\r\nHost: kodu.ut.ee\r\n\r\n")
43
>>> print s.recv(20)
HTTP/1.1 200 OK
Dat
```

- `recv()` returns bytes that are available in the read buffer
- `recv()` will wait if the read buffer empty (blocking by default)
- `recv()` will return 0 bytes if the connection is closed
- You must know how many bytes you must get
- Correct way to read HTTP response:
  - Read byte-by-byte until full response header received
  - Extract body size from `Connection-Length` header
  - Read byte-by-byte until full response body received

`http://docs.python.org/2/howto/sockets.html`

Task 1: Check certificates against CRL

Implement utility that checks if certificates are revoked.

```
$ ./crlcheck.py
usage: crlcheck.py --issuer issuer --certificates cert [cert ...] url

$ wget https://sk.ee/upload/files/EE_Certification_Centre_Root_CA.pem.crt
-O EECCRCA.pem

$ ./crlcheck.py https://sk.ee/crls/eeccrca/eeccrca.crl --issuer EECCRCA.pem
--certificates revoked.pem valid.pem nonissued.pem
[+] Downloading https://sk.ee/crls/eeccrca/eeccrca.crl
[+] CRL signature check successful!
[+] Serial 14970267057323525573973128269296929 1298 (revoked.pem) loaded
[+] Serial 13425817054688995380882267264047226074 (valid.pem) loaded
[-] Serial 9979740719785... (nonissued.pem) not loaded: not issued by CA
[-] Certificate 14970267057323... revoked: 2016-07-07 12:37:58 (unspecified)

$ ./crlcheck.py http://kodu.ut.ee/~arnis/appcrypto2018/outdated.crl
--issuer EECCRCA.pem --certificates revoked.pem valid.pem nonissued.pem
[+] Downloading http://kodu.ut.ee/~arnis/appcrypto2018/outdated.crl
[+] CRL signature check successful!
[-] CRL outdated (nextUpdate: 2014-05-25 08:18:44) (now: 2018-03-28 17:29:56)

$ ./crlcheck.py http://kodu.ut.ee/~arnis/appcrypto2018/badsign.crl
--issuer EECCRCA.pem --certificates revoked.pem valid.pem nonissued.pem
[+] Downloading http://kodu.ut.ee/~arnis/appcrypto2018/badsign.crl
[-] CRL signature verification failed!
```

Hints

- Download CRL using Python sockets (in the correct way)
- Convert UTCTime to datetime object:

```
>>> import datetime
>>> dateobj = datetime.datetime.strptime(utctimestr, '%y%m%d%H%M%SZ')
>>> print datetime.datetime.utcnow() > dateobj
True
```

- Use `urlparse` for easy URL parsing:

```
>>> import urlparse
>>> urlparse.urlparse("http://kodu.ut.ee/~arnis/some.file")
ParseResult(scheme='http', netloc='kodu.ut.ee', path='/~arnis/some.file',
params='', query='', fragment='')
>>> urlparse.urlparse("http://kodu.ut.ee/~arnis/some.file").netloc
'kodu.ut.ee'
```

- Use regular expression operation to extract body size:

```
>>> import re
>>> re.search('content-length:\s*(\d+)\s', header, re.S+re.I).group(1)
```

- For signature verification use pyasn1 `encoder.encode()` to get decoded substructure as DER:

```
>>> from pyasn1.codec.der import decoder, encoder
>>> tbsCertList = encoder.encode(decoder.decode(crl)[0][0])
```

Comments

- **Wrong** ways to download HTTP response body:
  - Reading response in one go (**wrong!**):

```
body = s.recv(content_length)
```

"The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested." (man page recv section 2)

  - Reading until socket closed (**wrong!**):

```
body = ""
buf = s.recv(1024)
while len(buf):
      buf = s.recv(1024)
      body+= buf
```

After sending a response, an HTTP/1.1 server will wait for more request/response exchanges, unless header "Connection: close" was specified by the client.
Therefore `s.recv()` will hang until the timeout configured by the server is reached.

Questions

- How can the relying party find the CRL?
- How is the integrity of CRL data assured?
- How frequently the CA should issue CRL?
- How frequently the relying party should refresh CRL?
- How can the relying party know that CRL is fresh?
- How is suspension different from revocation?
    - How to distinguish in CRL between revoked and suspended certificate?
- How to verify if root CA certificate has not been revoked?
- Is the subject liable for signatures made after certificate is revoked?
- Is the subject liable for signatures made in certificate validity period?