# MTAT.07.017
# Applied Cryptography

Certificate Revocation List (CRL)
Online Certificate Status Protocol (OCSP)

University of Tartu

Spring 2020

# Certificate validity

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

Examples:

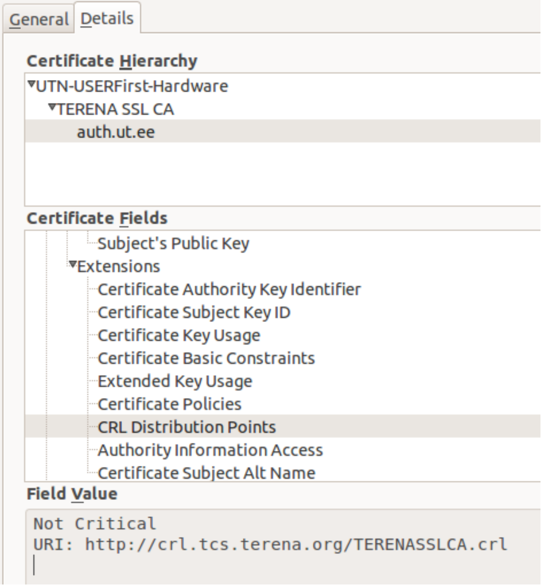
- Private key compromised or lost
- Misissued certificate
- Data has changed
- Contract ended

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**List of unexpired certificates that have been revoked by CA**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?

# CRL Distribution Points



Certificate Hierarchy

- UTN-USERFirst-Hardware
  - TERENA SSL CA
    - auth.ut.ee

Certificate Fields

- Subject's Public Key
- Extensions
  - Certificate Authority Key Identifier
  - Certificate Subject Key ID
  - Certificate Key Usage
  - Certificate Basic Constraints
  - Extended Key Usage
  - Certificate Policies
  - CRL Distribution Points
  - Authority Information Access
  - Certificate Subject Alt Name

Field Value

```
Not Critical
URI: http://crl.tcs.terena.org/TERENASSLCA.crl
```

# Certificate Revocation List (CRL)

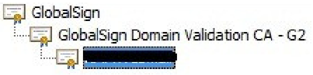
```
CertificateList  ::=  SEQUENCE  {
    tbsCertList          TBSCertList,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING  }

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

# Certificate chain



GlobalSign

GlobalSign Domain Validation CA - G2

[redacted]

- How to validate a certificate chain?
- Where to look whether subject's certificate is not revoked?
  - In CRL issued by intermediate CA (usually every 12h)
  - Grace period
- Where to look whether intermediate CA is not revoked?
  - In CRL issued by root CA (usually every 3 month)
  - Grace period?!
- Where to look whether the root CA is not revoked?
  - In CRL issued by root CA itself (flawed)

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

# Liability analysis

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

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

- 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 before CA has been informed about the key compromise?

# 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 to verify if root CA certificate has not been revoked?
- Is the subject liable for transactions made after certificate is revoked?
- Is the subject liable for transactions made in certificate validity period?

# Online Certificate Status Protocol

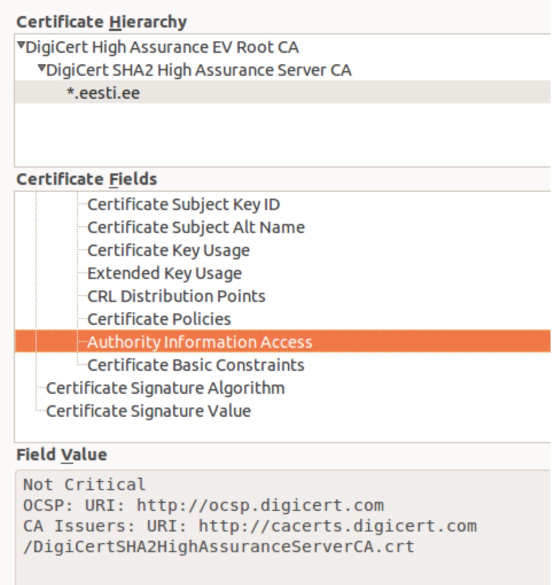
CRL shortcomings:

- Size of CRLs
- Client-side complexity
- Outdated status information

*"The Online Certificate Status Protocol (OCSP) enables applications to determine the (revocation) state of an identified certificate."*

- Where can the relying party find the OCSP responder?
- How is the certificate identified in the OCSP request?
- How is the integrity of OCSP response assured?
- How to ensure the freshness of OCSP response?
- How frequently certificate status should be checked?

# Authority Information Access



**Certificate Hierarchy**

- ▾DigiCert High Assurance EV Root CA
  - ▾DigiCert SHA2 High Assurance Server CA
    - *.eesti.ee

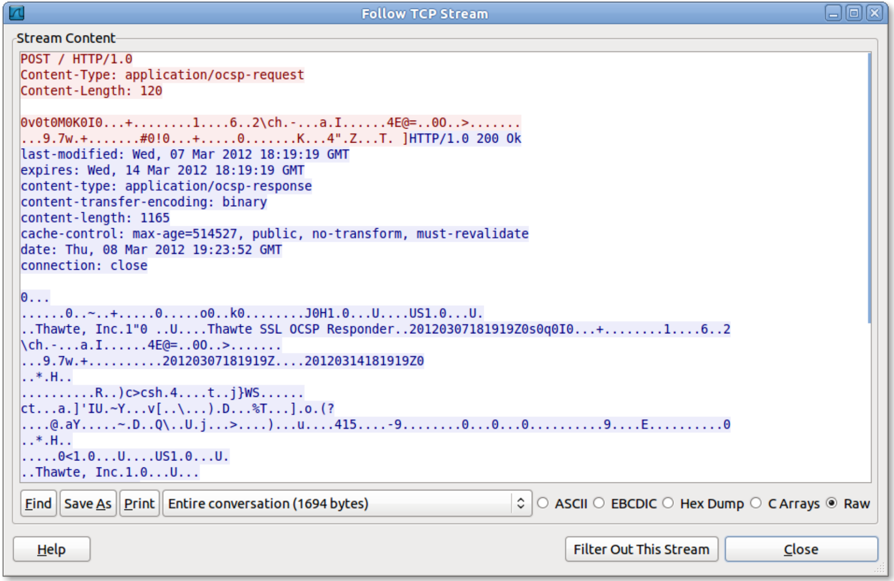
**Certificate Fields**

- Certificate Subject Key ID
- Certificate Subject Alt Name
- Certificate Key Usage
- Extended Key Usage
- CRL Distribution Points
- Certificate Policies
- Authority Information Access
- Certificate Basic Constraints
- Certificate Signature Algorithm
- Certificate Signature Value

**Field Value**

```
Not Critical
OCSP: URI: http://ocsp.digicert.com
CA Issuers: URI: http://cacerts.digicert.com
/DigiCertSHA2HighAssuranceServerCA.crt
```

# OCSP over HTTP



```
POST / HTTP/1.0
Content-Type: application/ocsp-request
Content-Length: 120

0v0t0M0K0I0...+........1....6..2\ch.-...a.I......4E@=..0O..>.......
...9.7w.+.......#0!0...+.....0.......K...4".Z...T. ]HTTP/1.0 200 Ok
last-modified: Wed, 07 Mar 2012 18:19:19 GMT
expires: Wed, 14 Mar 2012 18:19:19 GMT
content-type: application/ocsp-response
content-transfer-encoding: binary
content-length: 1165
cache-control: max-age=514527, public, no-transform, must-revalidate
date: Thu, 08 Mar 2012 19:23:52 GMT
connection: close

0...
......0..~..+.....0.....o0..k0........J0H1.0...U....US1.0...U.
..Thawte, Inc.1"0 ..U....Thawte SSL OCSP Responder..20120307181919Z0s0q0I0...+........1....6..2
\ch.-...a.I......4E@=..0O..>.......
...9.7w.+.........20120307181919Z....20120314181919Z0
..*.H..
..........R..)c>csh.4....t..j}WS......
ct...a.]'IU.~Y...v[..\...).D...%T...].o.(?
....@.aY.....~.D..Q\..U.j...>....)...u....415....-9........0...0...0..........9....E..........0
..*.H..
.....0<1.0...U....US1.0...U.
..Thawte, Inc.1.0...U...
```

Find
Save As
Print
Entire conversation (1694 bytes)
○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw
Help
Filter Out This Stream
Close

# Request syntax

```
OCSPRequest ::= SEQUENCE {
  tbsRequest TBSRequest,
  optionalSignature [0] Signature OPTIONAL }

Signature ::= SEQUENCE {
  signatureAlgorithm AlgorithmIdentifier,
  signature          BIT STRING,
  certs              [0] SEQUENCE OF Certificate OPTIONAL }

TBSRequest ::= SEQUENCE {
  version            [0] Version DEFAULT v1(0),
  requestorName      [1] GeneralName OPTIONAL,
  requestList        SEQUENCE OF SEQUENCE {
     reqCert                   CertID,
     singleRequestExtensions  [0] Extensions OPTIONAL }
  requestExtensions  [2] Extensions OPTIONAL }

CertID ::= SEQUENCE {
  hashAlgorithm       AlgorithmIdentifier,
  issuerNameHash      OCTET STRING, -- Hash of Issuer's DN
  issuerKeyHash       OCTET STRING, -- Hash of Issuer's public key
                (i.e., hash of subjectPublicKey BIT STRING content)
  serialNumber        CertificateSerialNumber }
```

http://tools.ietf.org/html/rfc6960

# Response syntax

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL }

   OCSPResponseStatus ::= ENUMERATED {
       successful            (0),  --Response has valid confirmations
       malformedRequest      (1),  --Illegal confirmation request
       internalError         (2),  --Internal error in issuer
       tryLater              (3),  --Try again later
                                   --(4) is not used
       sigRequired           (5),  --Must sign the request
       unauthorized          (6)   --Request unauthorized
   }

ResponseBytes ::=       SEQUENCE {
       responseType   OBJECT IDENTIFIER, --id-pkix-ocsp-basic
       response       OCTET STRING }
```

- `responseBytes` provided only if `responseStatus` is “successful”

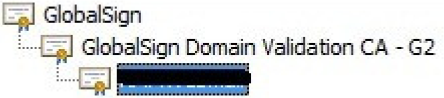
# Response syntax

```
response ::= SEQUENCE {
    tbsResponseData      ResponseData,
    signatureAlgorithm   AlgorithmIdentifier,
    signature            BIT STRING,
    certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

ResponseData ::= SEQUENCE {
    version             [0] EXPLICIT Version DEFAULT v1,
    responderID         [1] Name,
    producedAt          GeneralizedTime,
    responses           SEQUENCE OF SEQUENCE {
          certID           CertID,
          certStatus       CertStatus,
          thisUpdate       GeneralizedTime,
          nextUpdate       [0] EXPLICIT GeneralizedTime OPTIONAL,
          singleExtensions [1] EXPLICIT Extensions OPTIONAL }
    responseExtensions  [1] EXPLICIT Extensions OPTIONAL }

CertStatus ::= CHOICE {
    good        [0]     IMPLICIT NULL,
    revoked     [1]     IMPLICIT SEQUENCE {
          revocationTime   GeneralizedTime,
          revocationReason [0] EXPLICIT CRLReason OPTIONAL }
    unknown     [2]     IMPLICIT NULL }
```

# How to check the freshness of response?

- Check the signed timestamp (`producedAt` and `thisUpdate`)
  - What should be the allowed time difference?
  - Replay attacks
  - Reliance on the correctness of system clock
- Include nonce in the OCSP request and check it in the response
  - OCSP nonce extension (optional)
  - Prevents replay attacks
  - Vulnerable to downgrade attacks

# Who signs OCSP response?



GlobalSign

GlobalSign Domain Validation CA - G2

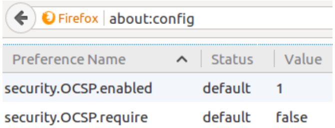
[redacted]

The key used to sign the response MUST belong to one of the following:

- CA who issued the certificate in question
- CA Authorized Responder who holds a specially marked certificate issued directly by the CA, indicating that the responder may issue OCSP responses for that CA
  - OCSP signing delegation SHALL be designated by the inclusion of `id-kp-OCSPSigning` flag in an `extendedKeyUsage` extension of the responder's certificate
  - How to check the revocation status of this certificate?
- Trusted Responder whose public key is trusted by the requester
  - Trust must be established by some out-of-band means

# Revocation checking in browsers

- CRLs are not supported
- Problems with OCSP:
  - Privacy leakage
  - Slower initial page loading
  - Chrome uses OCSP only to check EV certificates (uses CRLSets)
  - Firefox is not brave enough to fail-safe:



| Preference Name | Status | Value |
|---|---|---|
| security.OCSP.enabled | default | 1 |
| security.OCSP.require | default | false |

- Solution is OCSP stapling (web server provides OCSP response to the browser)
- Shorter certificate validity period may help
- How frequently the OCSP status should be queried?

# Questions

- Where can the relying party find the OCSP responder?
- How is the certificate identified in the OCSP request?
- How is the integrity of OCSP response assured?
- How to ensure the freshness of OCSP response?
- How frequently the validity status should be checked?
- What problem does the OCSP nonce extension solve?
- What is a downgrade attack?

# Hypertext Transfer Protocol (HTTP)

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

Client request (`http://example.com/hello`):

```
GET /hello HTTP/1.1
Host: example.com
Connection: close
```

```
POST /hello HTTP/1.1
Host: example.com
Content-Length: 24
Connection: close

sending_this_binary_blob
```

Server response:

```
HTTP/1.1 200 OK
Date: Wed, 24 Mar 2020 19:35:42 GMT
Server: Apache
Content-Length: 7033
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Tran...
```

- Header lines must all end with `<CR><LF>` (`b"\r\n"`)
- Header lines are separated from the body by an empty line
- POST requests have a 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(("example.com", 80))
>>> s.send(b'GET / HTTP/1.1\r\nHost: example.com\r\n\r\n')
37
>>> print(s.recv(20))
b'HTTP/1.1 200 OK\r\nAge'
```

- `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 the full response header is received
    - Extract body size from `Content-Length` header
    - Read byte-by-byte until the full response body is received
    - Avoid endless loops by checking the return value of `recv()`

`http://docs.python.org/3/howto/sockets.html`

# Task: OCSP checker – 5p

Implement a utility that queries OCSP for certificate validity:

```
$ ./ocsp_check.py valid.pem
[+] URL of OCSP responder: http://aia.sk.ee/esteid2018
[+] Downloading issuer certificate from: http://c.sk.ee/esteid2018.der.crt
[+] Querying OCSP for serial: 132457411991227041950906933396399710966
[+] Connecting to aia.sk.ee...
[+] OCSP producedAt: 2020-03-23 16:36:40
[+] OCSP thisUpdate: 2020-03-23 16:36:40
[+] OCSP status: good

$ ./ocsp_check.py revoked.pem
[+] URL of OCSP responder: http://aia.sk.ee/esteid2015
[+] Downloading issuer certificate from: http://c.sk.ee/ESTEID-SK_2015.der.crt
[+] Querying OCSP for serial: 165400448864000643393593611773932020928
[+] Connecting to aia.sk.ee...
[+] OCSP producedAt: 2020-03-23 16:36:44
[+] OCSP thisUpdate: 2020-03-23 16:36:44
[+] OCSP status: revoked
```

# Task: OCSP checker

- Extract OCSP responder's URL and CA certificate URL from certificate's Authority Information Access (AIA) extension
- Send HTTP requests using Python sockets (the correct way**!**)
- Use `urlparse` for easy URL parsing:

```
>>> from urllib.parse import urlparse
>> urlparse("http://example.com/abc")
ParseResult(scheme='http', netloc='example.com', path='/abc', params='', query='', fragment='')
>>> urlparse.urlparse("http://example.com/abc").netloc
'example.com'
```

- Use regular expression to get length of HTTP response body:

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

- Construct OCSP request using your ASN.1 DER encoder
- OCSP response parsing code is in the template
- Signature verification checks can be skipped

# Task: OCSP checker

- OCSP request must include “`Content-Type: application/ocsp-request`”
- `aia.sk.ee` returns “revoked” for unrecognized CertIDs
- `dumpasn1` fails to decode OCSP request
  - use `openssl asn1parse`
- OCSP request for `valid.pem`:

```
$ openssl asn1parse -inform der -in valid.pem_ocsp_req
    0:d=0  hl=2 l=  81 cons: SEQUENCE
    2:d=1  hl=2 l=  79 cons: SEQUENCE
    4:d=2  hl=2 l=  77 cons: SEQUENCE
    6:d=3  hl=2 l=  75 cons: SEQUENCE
    8:d=4  hl=2 l=  73 cons: SEQUENCE
   10:d=5  hl=2 l=   9 cons: SEQUENCE
   12:d=6  hl=2 l=   5 prim: OBJECT            :sha1
   19:d=6  hl=2 l=   0 prim: NULL
   21:d=5  hl=2 l=  20 prim: OCTET STRING      [HEX DUMP]:455DBEF01E1E2B1058EF1F969918A80708A62182
   43:d=5  hl=2 l=  20 prim: OCTET STRING      [HEX DUMP]:D9AC70DB5F7EBE94F8A0E4BE47A2D034AD9A2A12
   65:d=5  hl=2 l=  16 prim: INTEGER           :63A65E9ED37BF0115C2C8928DF0FE2F6
```

# Comments

**Wrong** way to download HTTP response body:

- Reading the 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 the socket is closed (**wrong!**):

  ```
  body = b''
  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.