MTAT.07.017
Applied Cryptography

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

University of Tartu

Spring 2023

Certificate validity

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

Examples:

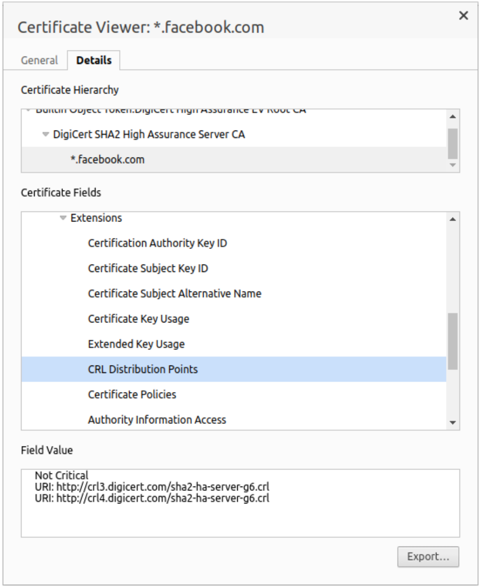
- Private key compromised
- Misissued certificate
- Data has changed

Solution – Certificate Revocation List (CRL):

List of unexpired certificates that have been revoked by CA

- Where can a relying party find the CRL?
- How can we assure the integrity of the CRL?
- How frequently should the CA issue the CRL?
- How frequently should the relying parties refresh the CRL?
- How can the relying party know that the CRL is fresh?

CRL Distribution Points



Certificate Viewer: *.facebook.com

Certificate Hierarchy

- Builtin Object Token:DigiCert High Assurance EV Root CA
  - DigiCert SHA2 High Assurance Server CA
    - *.facebook.com

Certificate Fields

- Extensions
  - Certification Authority Key ID
  - Certificate Subject Key ID
  - Certificate Subject Alternative Name
  - Certificate Key Usage
  - Extended Key Usage
  - CRL Distribution Points
  - Certificate Policies
  - Authority Information Access

Field Value

```
Not Critical
URI: http://crl3.digicert.com/sha2-ha-server-g6.crl
URI: http://crl4.digicert.com/sha2-ha-server-g6.crl
```

Certificate Revocation List (CRL)

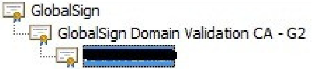
```
CertificateList  ::=  SEQUENCE  {
    tbsCertList          TBSCertList,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING  }

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
- `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 the CRL

Certificate chain



GlobalSign
GlobalSign Domain Validation CA - G2
[redacted]

- How to validate a certificate chain?
- Where to check whether the subject's certificate is not revoked?
  - In the CRL issued by the intermediate CA (usually every 12h)
  - Grace period
- Where to check whether the intermediate CA is not revoked?
  - In the CRL issued by the root CA (usually every 3 months)
  - Grace period?!
- Where to check whether the root CA is not revoked?
  - In the CRL issued by the root CA itself (flawed)
  - Must be revoked by out-of-band means

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

Liability analysis

Let's assume that a 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 the CRL?
- in the time period after the CRL has been issued but not available to relying parties (e.g., CA server downtime)?
- in the time period before the next CRL has been issued?
- in the time period before the CA has marked the certificate revoked in their internal database?
- in the time period before the CA has been informed about the key compromise?

Questions

- How can a relying party find the CRL?
- How is the integrity of CRL data assured?
- How frequently should the CA issue a CRL?
- How frequently should the relying parties refresh the CRL?
- How can the relying party know that the CRL is fresh?
- How can it be verified that the root CA certificate has not been revoked?
- Is the subject liable for the transactions made after the certificate is revoked?
- Is the subject liable for the transactions made in the certificate validity period?

Online Certificate Status Protocol

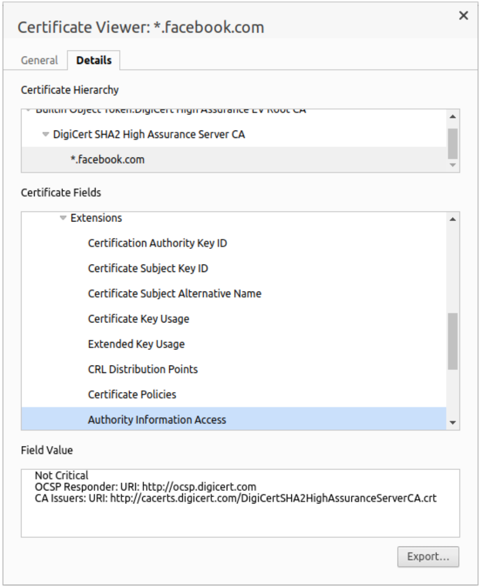
CRL shortcomings:

- 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.”

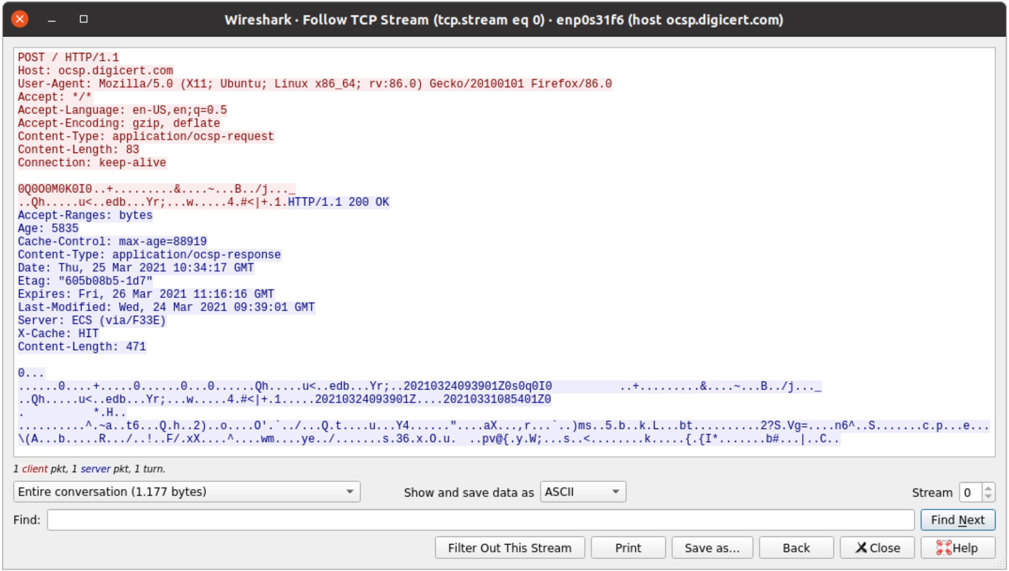
- Where can the relying parties find the OCSP responder?
- How is a certificate identified in the OCSP request?
- How is the integrity of an OCSP response assured?
- How can the freshness of an OCSP response be ensured?

Authority Information Access


Certificate Viewer: *.facebook.com
×
General
Details
Certificate Hierarchy
Builtin Object Token:DigiCert High Assurance EV Root CA
DigiCert SHA2 High Assurance Server CA
*.facebook.com
Certificate Fields
Extensions
Certification Authority Key ID
Certificate Subject Key ID
Certificate Subject Alternative Name
Certificate Key Usage
Extended Key Usage
CRL Distribution Points
Certificate Policies
Authority Information Access
Field Value
Export...

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

OCSP over HTTP



Wireshark · Follow TCP Stream (tcp.stream eq 0) · enp0s31f6 (host ocsp.digicert.com)

```
POST / HTTP/1.1
Host: ocsp.digicert.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:86.0) Gecko/20100101 Firefox/86.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/ocsp-request
Content-Length: 83
Connection: keep-alive

0Q0O0M0K0I0..+.........&....~...B../j..._
..Qh.....u<..edb...Yr;...w.....4.#<|+.1.HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 5835
Cache-Control: max-age=88919
Content-Type: application/ocsp-response
Date: Thu, 25 Mar 2021 10:34:17 GMT
Etag: "605b08b5-1d7"
Expires: Fri, 26 Mar 2021 11:16:16 GMT
Last-Modified: Wed, 24 Mar 2021 09:39:01 GMT
Server: ECS (via/F33E)
X-Cache: HIT
Content-Length: 471

0...
......0....+.....0......0...0......Qh.....u<..edb...Yr;..20210324093901Z0s0q0I0          ..+.........&....~...B../j..._
..Qh.....u<..edb...Yr;...w.....4.#<|+.1.....20210324093901Z....20210331085401Z0
.           *.H..
..........^.~a..t6...Q.h..2)..o....O'.`../...Q.t....u...Y4......"....aX...,r...`..)ms..5.b..k.L...bt..........2?S.Vg=....n6^..S.......c.p...e...
\(A...b.....R.../..!..F/.xX....^....wm....ye../.......s.36.x.O.u.  ..pv@{.y.W;...s..<........k.....{.{I*.......b#...|..C..
```

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (1.177 bytes)

Show and save data as ASCII

Stream 0

Find:

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

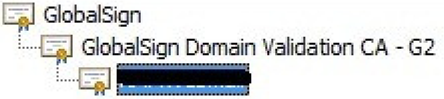
```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL }

   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 }
```

Who signs OCSP responses?



GlobalSign

GlobalSign Domain Validation CA - G2

[redacted]

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

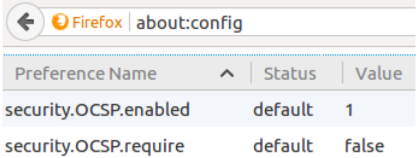
- CA who issued the certificate in question
- CA Authorized Responder who holds a specially marked certificate issued directly by the CA, indicating that the responder may issue OCSP responses for that CA
    - OCSP signing delegation SHALL be designated by the inclusion of `id-kp-OCSPSigning` flag in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

How can the freshness of a response be checked?

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

Revocation checking by browsers

- CRLs are not supported
- Problems with OCSP:
  - Privacy leakage
  - Initial page loading slower
  - OCSP checks are not, generally, performed by Chrome
  - Blacklist distributed using browser updates: CRLSets (Chrome), OneCRL (Firefox)
  - 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)
  - OCSP must-staple x509v3 extension to prevent downgrade attacks
- How fresh should the OCSP response be?
- Shorter certificate validity period may help

Questions

- Where can a relying party find the OCSP responder?
- How is a certificate identified in the OCSP request?
- How is the integrity of the OCSP response assured?
- How can the freshness of the OCSP response be ensured?
- How frequently should the validity status be checked?
- What problem does the OCSP nonce extension solve?
- What is a replay attack?
- 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: Thu, 11 Oct 2022 11:39:23 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
>>> 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 is empty (blocking by default)
- `recv()` will return 0 bytes if the connection is closed
- We must know how many bytes we 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

Implement a utility that queries an OCSP responder for a certificate's validity:

```
$ ./ocsp_check.py valid.pem
[+] URL of OCSP responder: http://ocsps.ssl.com
[+] Downloading issuer certificate from: http://cert.ssl.com/SSLcom-SubCA-SSL-RSA-4096-R1.cer
[+] OCSP request for serial: 16340626425735156093300147472379883536
[+] Connecting to ocsps.ssl.com...
[+] OCSP producedAt: 2022-10-09 21:24:49 +00:00
[+] OCSP thisUpdate: 2022-10-09 21:24:49 +00:00
[+] OCSP nextUpdate: 2022-10-16 21:24:48 +00:00
[+] OCSP status: good


$ ./ocsp_check.py revoked.pem
[+] URL of OCSP responder: http://ocsps.ssl.com
[+] Downloading issuer certificate from: http://cert.ssl.com/SSLcom-SubCA-SSL-RSA-4096-R1.cer
[+] OCSP request for serial: 141806724451593186148692230332761788677
[+] Connecting to ocsps.ssl.com...
[+] OCSP producedAt: 2022-10-09 19:44:45 +00:00
[+] OCSP thisUpdate: 2022-10-09 19:44:45 +00:00
[+] OCSP nextUpdate: 2022-10-16 19:44:44 +00:00
[+] OCSP status: revoked
```

Task: OCSP checker

- Extract OCSP responder's URL and CA certificate's URL from certificate's Authority Information Access (AIA) extension
- Send HTTP requests using Python sockets (**the correct way!** – see slide **??**)
- 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("http://example.com/abc").netloc
'example.com'
```

- Use regular expression to extract the length of an 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
- To construct `issuerKeyHash` (`CertID`) encode `subjectPublicKey` bits to bytes
- OCSP response parsing code is in the template
- Signature verification checks can be skipped

Task: OCSP checker

- OCSP requests must include “`Content-Type: application/ocsp-request`”
- To debug HTTP errors use Wireshark’s “Follow → TCP Stream” feature
- OCSP responder may return “unauthorized” for unrecognized CertIDs
- OCSP request for `valid.pem`:

```
$ dumpasn1 valid.pem_ocsp_req
  0  81: SEQUENCE {
  2  79:   SEQUENCE {
  4  77:     SEQUENCE {
  6  75:       SEQUENCE {
  8  73:         SEQUENCE {
 10   9:           SEQUENCE {
 12   5:             OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
 19   0:             NULL
       :             }
 21  20:           OCTET STRING
       :             D4 92 94 BE 2B 4A 19 85 23 31 FE 69 82 67 BE 94
       :             A9 D8 D4 C5
 43  20:           OCTET STRING
       :             26 14 7E E0 DC D7 A6 F7 E2 D4 04 27 DF 61 F1 C2
       :             EC E7 32 CA
 65  16:           INTEGER 0C 4B 17 15 AA 53 CC 2F DD 0A 7E D7 8F 43 30 10
       :           }
       :         }
       :       }
       :     }
       :   }
```

Comments

The **wrong** way of downloading 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."

- 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 the header "`Connection: close`" was specified by the client.

  - `s.recv()` will hang until the timeout configured by the server is reached