

MTAT.07.017
Applied Cryptography

Online Certificate Status Protocol (OCSP)

University of Tartu

Spring 2018

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- Outdated status information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?
- At which point OCSP must 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

Follow TCP Stream

Stream Content

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

0v0t0M0K0I0...+.....1....6..2\ch-...a.I.....4E@=.00.>.....
...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@=.00.>.....
...9.7w+.....20120307181919Z...20120314181919Z
..*.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"
- Note that responseStatus is not signed

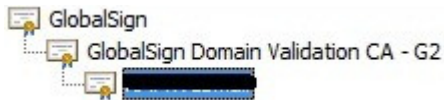
Response Syntax

```
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 response?



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

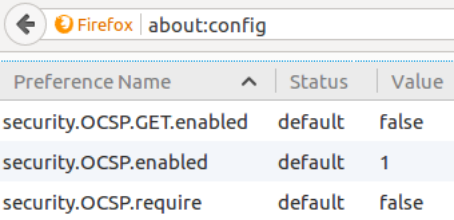
- CA who issued the certificate in question
- CA Authorized Responder who holds a specially marked certificate issued directly by the CA, indicating that the responder may issue OCSP responses for that CA
 - OCSP signing delegation SHALL be designated by the inclusion of `id-kp-OCSPSigning` flag in an `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

How to provide response freshness?

- Include signed timestamp in OCSP response (`producedAt` and `thisUpdate`)
 - What should be the allowed time difference?
 - Replay attacks
 - System clock in the Trusted Computing Base
- Include nonce in the OCSP request and check it in the response
 - OCSP nonce extension (optional)
 - Prevents replay attacks
 - Vulnerable to downgrade attacks

Revocation checking in browsers

- CRLs are not supported
- Problems with OCSP:
 - Privacy leakage
 - Slower initial page loading
 - Chrome does not use OCSP – uses CRLSets
 - Firefox is not brave enough to fail-safe:



The screenshot shows the Firefox 'about:config' page. The address bar contains the Firefox logo and the text 'about:config'. Below the address bar is a table of configuration preferences. The table has three columns: 'Preference Name', 'Status', and 'Value'. The 'security.OCS*P.enabled' row is highlighted in light blue. The 'security.OCS*P.require' row is also highlighted in light blue.

Preference Name	Status	Value
security.OCS*P.GET.enabled	default	false
security.OCS*P.enabled	default	1
security.OCS*P.require	default	false

- Solution is OCSP stapling (web server provides OCSP response to the browser)
- How frequently the OCSP status should be queried?

Server Sockets in Python

```
import socket
sserv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sserv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sserv.bind(('', 8888))
sserv.listen(0)
```

```
while True:
```

```
    (s, address) = sserv.accept()
```

```
    print "[+] Client %s:%s" % (address[0], address[1])
```

- `bind(('', 8888))` and `listen()` listens for client connections on all IPs on all network interfaces
- `accept()` will wait until client connects and returns tuple:
 - client socket (has `send()` and `recv()` methods)
 - address tuple – IP and port
- `SO_REUSEADDR` forces the kernel to reuse port even if it is in busy (`TIME_WAIT`) state (prevents error when rebinding)

<http://docs.python.org/2/howto/sockets.html>

Task: OCSP responder

Implement OCSP responder answering to HTTP POST requests.

```
$ ./ocsprresponder.py
```

```
usage: ocsprresponder.py --privkey privkey --cacert cacert --revoked cert [cert ...]
```

```
$ ./ocsprresponder.py --privkey priv.pem --cacert rootCA.pem --revoked issued1.pem issued2.pem
```

```
[+] Serial 1705406124 (issued1.pem) loaded
```

```
[+] Serial 3532215973 (issued2.pem) loaded
```

```
[+] Connection from 127.0.0.1:48318
```

```
[+] Connection from 172.17.57.208:45394
```

```
$ openssl ocspr -url http://127.0.0.1:8888/ -no_nonce -VAfile rootCA.pem
```

```
-issuer rootCA.pem -cert issued2.pem
```

```
Response verify OK
```

```
issued2.pem: revoked
```

```
  This Update: Apr 4 10:13:37 2018 GMT
```

```
  Reason: keyCompromise
```

```
  Revocation Time: Jan 1 12:12:00 2000 GMT
```

```
$ openssl ocspr -url http://127.0.0.1:8888/ -no_nonce -VAfile rootCA.pem
```

```
-issuer rootCA.pem -cert issued3.pem
```

```
Response verify OK
```

```
issued3.pem: good
```

```
  This Update: Apr 4 10:13:39 2018 GMT
```

```
$ openssl ocspr -url http://127.0.0.1:8888/ -no_nonce -VAfile rootCA.pem
```

```
-issuer rootCA2.pem -cert issued3.pem
```

```
Response verify OK
```

```
issued3.pem: unknown
```

```
  This Update: Apr 4 10:13:45 2018 GMT
```

Task: OCSP responder

- Bind your HTTP server on port 8888 on all network interfaces:

```
$ netstat -na | grep 8888
tcp      0      0 0.0.0.0:8888          0.0.0.0:*            LISTEN
```

- HTTP server should be able to process sequential connections
- Must support only single certificate in OCSP request
- Signature check in `load_serials()` not required.
- Omit `nextUpdate`. Use current time for `thisUpdate` and `producedAt`.
- Return `CertStatus`:
 - “unknown” if not issued by CA (`issuerNameHash` or `issuerKeyHash` in `CertID` does not match).
 - “revoked” if certificate revoked (set arbitrary `revocationTime` and `revocationReason`).
 - “good” otherwise.

Task: OCSP responder

```
$ openssl ocsp -url http://127.0.0.1:8888/ -no_nonce -VAfile rootCA.pem -issuer rootCA.pem  
-cert issued3.pem -text
```

OCSP Request Data:

```
Version: 1 (0x0)
```

Requestor List:

Certificate ID:

```
Hash Algorithm: sha1
```

```
Issuer Name Hash: 8350F92D60E6122B0112EF8E5381F3190BB2C703
```

```
Issuer Key Hash: 4396CDDBB018CC4DF32D699971706FF1639B3BFB
```

```
Serial Number: 577DC7A1
```

OCSP Response Data:

```
OCSP Response Status: successful (0x0)
```

```
Response Type: Basic OCSP Response
```

```
Version: 1 (0x0)
```

```
Responder Id: C = EE, O = University of Tartu, OU = IT dep, CN = Arnis Root CA
```

```
Produced At: Apr 4 10:45:20 2018 GMT
```

Responses:

Certificate ID:

```
Hash Algorithm: sha1
```

```
Issuer Name Hash: 8350F92D60E6122B0112EF8E5381F3190BB2C703
```

```
Issuer Key Hash: 4396CDDBB018CC4DF32D699971706FF1639B3BFB
```

```
Serial Number: 577DC7A1
```

```
Cert Status: good
```

```
This Update: Apr 4 10:45:20 2018 GMT
```

```
Signature Algorithm: sha1WithRSAEncryption
```

```
2c:30:21:51:c7:b8:98:d7:4b:5f:aa:1e:f5:62:fc:4a:6d:78:
```

```
Response verify OK
```

```
issued3.pem: good
```

```
This Update: Apr 4 10:45:20 2018 GMT
```

Task: OCSP responder

- Bonus point for supporting nonce extension:

```
$ openssl ocsp -url http://127.0.0.1:8888/ -nonce -VAfile rootCA.pem  
-issuer rootCA.pem -cert issued3.pem  
Response verify OK  
issued3.pem: good  
This Update: Apr 4 10:29:35 2018 GMT
```

- Half bonus point for returning response status “unauthorized” to clients from non-loopback IP (127.0.0.1):

```
$ openssl ocsp -url http://172.17.57.208:8888/ -nonce -VAfile rootCA.pem  
-issuer rootCA.pem -cert issued3.pem  
Responder Error: unauthorized (6)
```

- Half bonus point for serving nice response to GET requests:



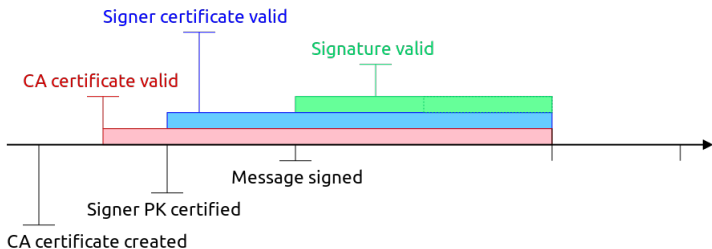
Hints

- Look on `resp_revoked.der`, `resp_good.der`, `resp_unknown.der`, `resp_nonce.der` and `resp_unauthorized.der` for response examples.
- Use openssl “-respout” parameter for debugging.
- Error “bad signature” may be caused by wrong DER encoding.
- DER encoding for ENUMERATED is the same as for INTEGER, just universal tag 10 instead of 2.
- DER encoding for GeneralizedTime is the same as for UTCTime, just 4-digit year encoding and universal tag 24 instead of 23.
- Datetime object conversion to GeneralizedTime string:
`datetime.datetime.utcnow().strftime("%Y%m%d%H%M%SZ")`
- CertStatus is implicitly tagged NULL value (for good/unknown) or SEQUENCE (for revoked)
 - Implicit tagging replaces “type” byte of original value:
 - Class bits – context-defined (1 0)
 - Form bit – from value to be tagged (primitive/constructed)
 - Tag bits – tag number

Certificate Status

CRL and OCSP allows to verify if certificate valid now.

- Simple for authentication
- What about digital signature?
- Signature valid as long as certificate has not expired?



- Was the certificate valid at the time of signing?
- How to find out the time of signing?

Trusted Timestamping

Signed statement of timestamping authority (TSA):

```
> This data [data] was presented to me at this time: [time]
> Yours,
> --
> TSA
> [signature]
```

<http://tools.ietf.org/html/rfc3161>

- data – usually a hash of the signature value
- Proves that the signature was given *before* the time specified
- Digital signature containers usually contain:
 - Signed files
 - Signature of files
 - Timestamp of the signature
 - OCSP response (acquired right after timestamping)
- How to verify digital signature after TSA/OCSP cert expires?
- Why is certificate suspension a bad idea?


DigiDoc Client

Logistika_ja_Transiidi_Assotsiatsiooni_lepung.ddoc

DigiDoc³
KLIENT

Settings | Help | About | English ▾

Use ID-card Use Mobile ID


 **EESTI VABARIIK**
REPUBLIC OF ESTONIA

No card in reader

Check if the ID-card is inserted correctly to the reader.
New ID-cards have chip on the back side of the card.


Container:C:\Users\user\AppData\Local\Temp\Logistika_ja_Transiidi_Assotsiatsiooni_lepung.ddoc [Save](#)


Container content:

Logistika_ja_Tra...ooni_lepung.docx 21 KB 

[Save files to disk](#)

Signatures

 **Marika Priske**
kantsler
Signed on 09. September 2011 time 15:37
Signature is **valid** [Show details](#)
[Remove](#)

 **Andres Valgerist**
Signed on 09. September 2011 time 11:26
Signature is **valid** [Show details](#)
[Remove](#)

[Send container to email](#) [Print summary](#)
[Browse container location](#) [Encrypt document](#)

[Add signature](#) [Close](#)

XML Signature

```
<?xml version="1.0" encoding="UTF-8"?>
<SignedDoc format="DIGIDOC-XML" version="1.3" xmlns="http://www.sk.ee/DigiDoc/v1.3.0#">
  <DataFile Filename="document.doc" Id="D0">UESDBBQABGA...AS1EAAAAA</DataFile>
  <Signature Id="S0">
    <SignedInfo>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="#D0">
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>Q43ti5R/wgi8q0oHsygLFTXE0qU=</DigestValue>
      </Reference>
      <Reference URI="#S0-SignedProperties">
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>G0HmQqHCqMxULzfWSONIL2i0mIU=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue Id="S0-SIG">kgsCQ6...M4rkCj8=</SignatureValue> - signature of <SignedInfo>
    <X509Certificate>IID4z...V8APa</X509Certificate>
  <SignedProperties Id="S0-SignedProperties">
    <SigningCertificate>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>RRFMpf0Xr5ZRyEs49m4S8M3oRnw=</DigestValue>
    </SigningCertificate>
    <SignatureProductionPlace>
      <City>Tallinn</City>
    </SignatureProductionPlace>
  </SignedProperties>
  <OCSPValues>
    ...
  </Signature>
  ...
</SignedDoc>
```


DigiDoc Crypto

The screenshot shows the Krüpto application window. The title bar reads "ID hello.cdoci". The application header includes the name "Krüpto" and navigation links for "Settings", "Help", "About", and "English". The main content area features the Estonian coat of arms and the text "EESTI VAB REPUBLIC OF EST". A user information box displays: "Name: Arnis Paršovs", "Personal code: 38608050013", "Card number: E0044843", and "Auth certificate is valid". A "Save" button is located to the right of this box. Below the header, the container path is "Container:E:\hello.cdoci". The "Container content:" section shows a file named "hello.bt" with a size of "21 B". The "Keys" section displays the key "PARŠOV,ARNIS,38608050013,ID-CARD" with a "Show details" link. At the bottom, there are links for "Send container to email" and "Browse container location", and two buttons: "Decrypt" and "Close".

hello.cdoci

Krüpto

Settings | Help | About | English ▾

 EESTI VAB REPUBLIC OF EST

Name: Arnis Paršovs
Personal code: 38608050013
Card number: E0044843
Auth certificate is **valid**



Container:E:\hello.cdoci [Save](#)

Container content:

hello.bt	21 B
----------	------

Keys

PARŠOV,ARNIS,38608050013,ID-CARD

[Show details](#)

[Send container to email](#)
[Browse container location](#)

[Decrypt](#) [Close](#)

XML Encryption

```
<?xml version="1.0" encoding="UTF-8" ?>
<denc:EncryptedData xmlns:denc="http://www.w3.org/2001/04/xmlenc#">

  <denc:EncryptedKey Recipient="PARŠOVS,ARNIS,38608050013,ID-CARD">
    <denc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <ds:X509Certificate>A1UdeEQaM...sQkDYzTwuVzxc800</ds:X509Certificate>
    <denc:CipherValue>AZrh0j...QOY=</denc:CipherValue>
  </denc:EncryptedKey>

  <denc:EncryptedKey Recipient="...
  <denc:EncryptedKey Recipient="...

  <denc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
  <denc:CipherValue>QGKSSP...Wg20fGDSJal/Dr6Y1x4Fgg==</denc:CipherValue>

  <denc:EncryptionProperties>
    <denc:EncryptionProperty Name="LibraryVersion">CDigiDoc|3.8.0.1136</denc:EncryptionProperty>
    <denc:EncryptionProperty Name="DocumentFormat">ENCDOC-XML|1.0</denc:EncryptionProperty>
    <denc:EncryptionProperty Name="orig_file">hello.txt|21||DO</denc:EncryptionProperty>
  </denc:EncryptionProperties>

</denc:EncryptedData>
```

- Content encrypted using random 128-bit AES key
- Key encrypted using receivers RSA public key
- IV stored as a first ciphertext block
- Integrity protection not provided

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?
- At which point OCSP must be checked?
- What problem does the OCSP nonce extension solves?
- What is downgrade attack?
- What is needed in order to prove that the data was signed when the certificate was valid?