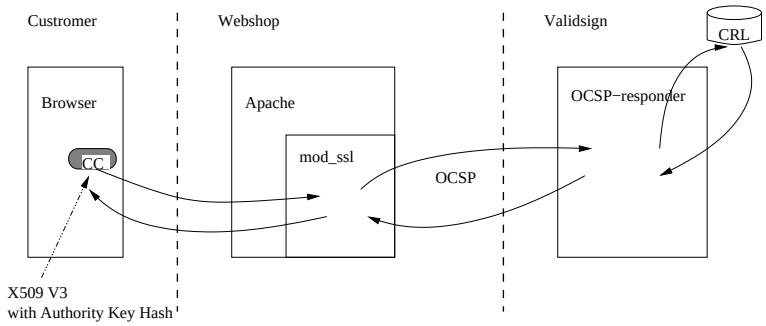
# Developing a Validation Authority Service for Apache

Norsk Regnesentral
ANVENDT DATAFORSKNING

Norwegian Computing Center/Applied Research and Development

NOTAT/NOTE




IMEDIA/03/02

Developing
a Validation
Authority Service
for Apache

Wolfgang Leister
Ole Aamot
Eirik Maus
Anund Lie (PKI CS)

Oslo
March 2002

Norsk Regnesentral
ANVENDT DATAFORSKNING

**Notat** / Notat

**Tittel**/Title:
Developing a Validation Authority Service for Apache

**Dato**/Date: March
**År**/Year: 2002
**Notat nr**/:
Note no: IMEDIA/03/02

**Forfatter**/Author:
Wolfgang Leister, Ole Aamot, Eirik Maus
Anund Lie (PKI Consulting Services as)

**Sammendrag**/Abstract:

Certificates are used in many applications. These certificates issued by a certificate authority must be checked by a relying party, in order to establish a chain of trust. One task thereby is to check whether certificates are revoked. In the SSL-enabled Apache server this is done by checking certificate revocation lists (CRL). We present a service which can check the status for a certificate online over OCSP, instead of accessing static files. A prototype is implemented for the Apache module mod_ssl using the OCSP protocol towards a validation authority service.

**Emneord**/Keywords: PKI, OpenSSL, Apache, Mozilla, certificate, signature, OCSP, X.509

**Målgruppe**/Target group: Validsign AS, PKI Consulting AS, NR

**Tilgjengelighet**/Availability: open

**Prosjektdata**/Project data: PKI-Validsign

**Prosjektnr**/Project no: 320080

**Antall sider**/No of pages: 26

Norsk Regnesentral/Norwegian Computing Center
Gaustadalléen 23, Postboks 114, N-0314 Oslo, Norway
Telefon (+47) 22 85 25 00, telefax (+47) 22 69 76 60

# Contents

# Developing a Validation Authority Service for Apache

Wolfgang Leister Ole Aamot Eirik Maus Anund Lie

Certificates are used in many applications. These certificates issued by a certificate authority must be checked by a relying party, in order to establish a chain of trust. One task thereby is to check whether certificates are revoked. In the Apache browser this is done by checking certificate revocation lists (CRL). We present a service which can check the status for a certificate online over OCSP, instead of accessing static files. A prototype is implemented for the Apache module mod_ssl using the OCSP protocol towards a validation authority service.

The following report presents the results of a recent project between NR and PKI Consulting Services / ValidSign AS. The document describes the implementation of a validation authority service, including the parts implemented in a prototype.

## 1. Description of the Validation Authority service

The validation authority service (VAS) is intended to provide the state of certificates whether they are revoked. This type of information is essential for both users (e.g., customers) and web sites (e.g., web shop, Internet bank). While the currently implemented certificate revocation lists (CRL[2]) have to be updated manually, the validation authority service can deliver up-to-date information. More on the concept of a validation authority service and its business model is beyond the scope of this document (see [3]).

### 1.1. Problem description

A public key infrastructure (PKI) provides the technical framework to support applications with the following security capabilities: user authentication, data confidentiality, data integrity, non-repudiation, and key management. In the implementation cryptographic methods are used. Public key cryptography uses key pairs, one of which is public, while the other is private. By binding the public key to some property, certificates are issued and signed digitally by a certificate authority (CA). These certificates are used on the Web in order to achieve the security capabilities mentioned above.

Introductions to the subject of PKI can be found in several text books and white papers [4], [5], [6], [7], [8]. There are several implementations of PKI. One of these implementations is OpenSSL. An introduction is given in [9] and [10].

It is expected that many Certificate Authorities (CA) and corresponding Registration Authorities (RA) operate in the same and/or overlapping domains. Therefore, End Entities (EE) may be forced to use certificates from several CAs to establish the level of trust needed, e.g., for Internet-banking, web-based shopping, authorisation, or tax reporting to public sector.

It is worth noticing that an End Entity has a certificate and the Relying Party trusts a certificate. In a simple business scenario involving buying and selling the buyer is playing the role as EE towards RP (relying party). But the buyer also wants to validate certificates used by the seller, so the buyer also plays the role of RP, and vice versa.

When EE and RP interact based on PKI, the RP needs to decide whether to accept the EE's certificate or not. A certificate is valid in a limited time frame. The time period is

stated in the certificate, but within this period the certificate can be suspended or revoked. An RP needs to validate each certificate it wants to accept. This validation is done by using a certificate revocation list (CRL[2]), an online validation check (e.g., OCSP[11], Online Certificate Status Protocol) or other access to the CA who has issued the certificate. Today the RPs need to establish certificate validation towards all relevant CAs.

When we speak of validation, we refer to the validation of the certificate itself, which includes checking syntax, dates, the signature of the issuer and revocation information. Other issues of validation, e.g., whether the use of a certificate corresponds to the certificate policy governing the certificate in question, or the authorisation of the holder of the certificate is beyond the scope of our purposes.

For our purposes, the main challenge consists in the following:

> From a juridical and organisational point of view the RP needs a simple mechanism to validate certificates from different CAs.

There are other challenges within the setting of PKI, which are very essential [3]. However, these are are beyond the scope of this technical note. We concentrate on the implementation of a validation authority service. Note, such a service can hide the complexity of a many-to-many relationship from the RP. The functionality of the service is for our point of view to supply technical validation of certificates, which is reasonably well covered by standards.

To both RPs and EEs the area of use for a certain certificate is of very high importance. Especially when trust is built based on use of PKI alone, there is also a need for categorising the strength of the certificate and if relevant, authorisation information based on the certificate. With the approx. 140 root certificates shipped with Windows 2000 in mind, we believe that the average RP and EE will not have the ability to distinguish between them without a reliable service helping them.

By implementing a validation authority service, we introduce one more infrastructure element to trust. But at the same time we reduce the individual RP's dependence on knowledge of the large number of CAs, trust structures between CAs, validation paths and strengths of certificates.

## 1.2. The situation today

In todays situation all certificates can be issued by many (several hundred) certificate authorities (CA). There is no unified policy in web browsers how to handle certificates. E.g., one of the web browsers comes with a list of signatures of about 150 certificate authorities, who have payed to be included in the browser's standard distribution. The web browser can verify the signature of the issuer on the certificate of the web site by trying to find the issuer in the issuer list of accepted CAs. If the issuer is not found the user receives a warning that the issuer is unknown.

However, it is still possible for the user to accept the certificate in question by accepting the certificate manually. It is known that one of the web browsers puts the issuer then into the list of approved issuers, whether it is a new, unknown one or possibly one that previously has been deleted due to missing trustworthy. To our knowledge, none of the browsers checks whether the web site's certificate is revoked.

For web sites the only possibility has been that the client certificate comes from an approved issuer (CA) (in this case the user has to use many passwords and certificates). Alternatively, the web site has to check all existing issuers whether the user has a valid certificate. For this purpose the web site has to check with all the existing issuers, whether a certificate is

(still) valid. Such a solution is usually too expensive for web sites, and therefore this solution is rather rare.

## 1.3. The Validation Authority service

The Validation Authority Service (VAS) maintains data on revoked certificates from all issuers. Both web sites and clients (browsers) can communicate with the VAS using the OCSP (Open Certificate Status Protocol). The service can also classify the Certificate Policy (CP) of the issuers. This includes the terms on which the certificate is issued, and a grade scale for the user. The service is supposed to be available for both users and sites to validate the certificates. Instead of implementing the validation routines in the browser or server, only a call from the client to the VAS server will be necessary.

### Usage case 1: Verifying client certificate

The web site uses the VAS to check whether the user's certificate is valid. To achieve this we have to "shortcut" the SSL-module of the web server, and send the certificate for further checking to the VAS server.

In this scenario we have to ensure that both partners do authentication towards each other. Mechanisms of OCSP, the use of SSL, secure lines or other mechanisms are required to exclude intruders into this communication.

### Usage case 2: Verifying the web site certificate

In this case the client (web browser) checks the certificate of the web site. For an example, this could be used by web shops (especially for first-time buyers). This makes a change in the SSL-module of the client necessary, which has to be configured to talk to the VAS server instead of verifying the certificates based on local (and probably outdated) file information only.

There are several levels of validation services. The most basic one is online revocation check (ORC), which is supported by OCSP v1. With this service, it is still the responsibility of the relying party to build and check the validation chain from the certificate to be validated up to a trusted root. Other proposed protocols (e.g., OCSP v2 [12], SCVP [13], XKMS [14]) offer higher level services such as delegated path discovery (DPD) and delegated path validation (DPV), where larger parts of the certificate validation process are delegated to the validation service. So far, OCSP v1 is the only protocol with an official status, and therefore this prototype primarily targets online revocation check.

## 1.4. Additional Problem Areas

A certificate for a web site or for a user does not need to be issued by a CA directly, but could have been issued by a registration authority (RA) to which the CA has transferred its rights. In this case the RA's certificate will be issued by the CA. Possibly the RA can have delegated this further to other parties. In such a way a "hierarchy of trust" is built up, and it will be necessary to process a chain of certificates.

In other cases several CAs could sign their certificates mutually to ensure their trustworthy to each other. In this case graphs of trust are built between equal partners. These graphs have to be processed until a trustworthy signature can be found. Note the problem with the termination of the verification process while processing the graph.

## 1.5. Description of the prototype

The prototype shows the principles of the concept of VAS within the context of web sites. The prototype shows that client certificates can be checked on the browser using the VAS. The use of the VAS in browsers in order to check the web site's certificate would be possible. However, we leave this for further exploration within a future project.

The prototype does not comprise of all the functionality of a working VAS. The main focus was put on checking whether certificates have been revoked. The prototype has its focus to validate a given certificate towards a given CRL, implementing OCSP servers and clients, and integrate these with a web site (e.g., a web shop).

## 1.6. Software Overview

We study several components and software packages that are available to implement the functionality of the VAS. An overview on the software will be given in the following. Much of the work was done on open source software, as the source code is available. The source code can be altered and adapted to our needs. It is intended to donate our work back to the open source community.

For the prototype we intend to use the Apache HTTPD server on a Linux platform. The module `mod_ssl` which is the OpenSSL-module that will be used for the SSL connections will be altered in order to contain the code for the OCSP client. The OpenSSL package will be used as far as possible within the project. For all these parts the source code is available.

This section presents software and other elements worth being mentioned. The software in the following list is especially important for our work:

- Apache HTTPD Server
- Apache SSL module (we use `mod_ssl`)
- OpenSSL
- ssldump (produces log files of SSL sessions)
- Mozilla

### 1.6.1. Apache HTTPD Server

The Apache HTTPD server (see `http://httpd.apache.org/`) is used to implement the Web site in our project. Apache is published under the Apache Software License, Version 1.1 (see `http://www.apache.org/LICENSE`), and falls under the Free and Open Source Software labels.

We use Apache version 1.3.22, which is checked in in the project repository. We did not make changes in this part of the software.

### 1.6.2. Apache SSL module

`mod_ssl` is the OpenSSL module for the Apache HTTPD server (see `http://www.modssl.org/`). The software resides in a separate directory, and is maintained separately. However, during compilation the code is copied into the Apache directory.

We use `mod_ssl`, version 2.8.5. This can be regarded as a extension to Apache, which handles ssl requests. This software is checked in in the project repository. This module has been subject to changes from our side in order to include code for the validation of Client certificates.

mod_ssl is published under a BSD-style license (see `http://www.modssl.org/source/exp/mod_ssl/pkg.mod_ssl/LICENSE`).

### 1.6.3. OpenSSL

We use a development snapshot of OpenSSL 0.9.7. (Daily experimental snapshots are available from `ftp://ftp.openssl.org/snapshot/`) We could observe problems with this snapshot. However, we need the snapshot, because OCSP is only implemented in the newer snapshots.

We did not make changes in this software. However, we need some of the implementation in order to program OCSP functionality. We link against the library to implement the OCSP validation functionality in mod_ssl.

The OpenSSL package contain strong cryptography, so even if it is created, maintained and distributed from liberal countries in Europe (where it is legal to do this), it falls under certain export/import and/or use restrictions in some other parts of the world.

### 1.6.4. ssldump

The program ssldump is used to produce dumps of the communication to the `mod_ssl` module of Apache (or general SSL applications) in a human readable form. The program can be found on `http://www.rtfm.com/ssldump/` and downloaded at no cost. The software is also included with a Book on SSL and TLS [1]

Some changes (see A.2.1) had to be made due to a compilation error against the OpenSSL 0.9.7 development snapshot.

### 1.6.5. Mozilla

Mozilla is a web browser that is chosen to implement the server certificate with on the client.

We have not identified the changes to be made to Mozilla so far. See 4.

## 2. Checking Client Certificates in the Web Server

We implement the VAS including the calls to the VAS within the SSL-enabled Apache server. We followed the this work plan in order to be able to check the revocation state of certificates within Apache:

- Install Apache, mod_ssl, and OpenSSL source code, and make sure compilation works well. See A.1 for details.
- Generate a CA certificate signed by CA, and install a client certificate on client and server. See A.3.1 for further details.
- In mod_ssl we locate where the verification of the client certificate takes place. The client certificate is sent from the client to the server in the SSL negotiation. The verification towards the CRL in mod_ssl was then added with a query to the VAS, using the OCSP protocol. Likewise the reply from the VAS was received and processed in order to produce an answer for the verification of the client certificate.
- The VAS responder was be implemented. Its task is to receive an OCSP request, check the certificate towards CRL, and give the reply of the validation by OCSP back to the server.

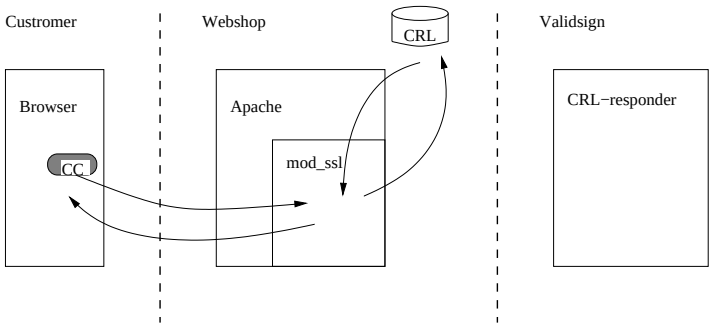



Figure 1: Original implementation of CRL check

## 2.1. Using files for CRL check

The checking of the CRL is performed in the mod_ssl package in the file `mod_ssl/pkg.sslmod/ssl_engine_kernel.c`. The function to check the certificate towards the CRL is done in the function `ssl_callback_SSLVerify_CRL`, which is called by `ssl_callback_SSLVerify`.

The routine `ssl_callback_SSLVerify_CRL` does the following:

- Extracts subject and issuer (X.509) of current certificate.
- Retrieves and verifies the signature of CRL.
- Check date of CRL to make sure it's not expired.
- Try to retrieve a CRL corresponding to the issuer of the current certificate in order to check for revocation.
- Check if the current certificate is revoked by this CRL.

The function `ssl_callback_SSLVerify_CRL` is called twice: first for the (current certificate's) issuer certificate and then for the current certificate itself. This implies that `ssl_callback_SSLVerify` is called twice. Therefore, we have to be prepared to check both certificates.

In Figure 1 we show how the original implementation does the CRL check on files. In the following sections we describe how this CRL check can be done outside of the web server by using another server that checks on revocation.[1]

## 2.2. Using OCSP for CRL check

Instead of checking the CRL directly, we use the OCSP [11] in order to send information from the current certificate to a server, which does the CRL checking[2]. The server answers with the CRL-status of the certificate in question. This service is implemented as an online revocation check (see Section 1.3). The API within the existing program is not altered.

This concept is illustrated in Figure 2. The `mod_ssl` module of apache produces an OCSP request, which is sent to a OCSP responder. The OCSP responder checks the certificate against its database and answers back with one of the responses *ok*, *invalid* or *unknown*.

For the implementation code from the application program `ocsp` in the applications directory of OpenSSL is used. OCSP needs four values for the request, that have to be retrieved from the certificate:

[1] It is intended to move more of the certificate processing to this server at a later stage.
[2] The RFC 2560 defines the OCSP version 1.

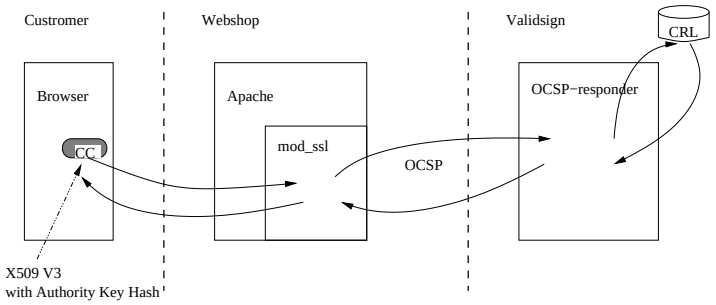



Figure 2: CRL check for X.509 V3 certificates using OCSP

- *certificate serial number*,
- *issuer name hash*,
- *issuer key hash*,
- *hash algorithm*.

The certificate's serial number, issuer name, and hash algorithm can be retrieved from the certificate, and thus the the key hash of the issuer name is available. In the original `ocsp`-application, the issuer key hash is retrieved from the issuer certificate. However, the issuer certificate is not available at this point in the `mod_ssl` program.

To build an OCSP v1 query, an identification of the certificate issuer (issuer key hash of the public key) is required. The way `mod_ssl` is organized, we had substantial technical difficulties getting access to this information. This illustrates that ORC puts greater demands on the client than DPV.

We extracted the issuer key hash value from the X.509 V3 authority key identifier extension in the certificate to be validated. However, this will not always work. (Even if present, the authority key identifier could be something different from a usable key hash.) A better strategy would be to extract the public key from the real issuer certificate and generate the hash from that, but we had problems accessing it from the validation callback in OpenSSL.

Different certificate profiles and issuers use this field in different ways. Therefore this value should be used with caution. For more discussion on this subject see also Peter Gutmans X.509 Style Guide [15].

For certificates with the X.509 V3 extensions `keyid` we implemented a request to an OCSP responder. The response is retrieved by the function `OCSP_resp_find_status` and wrapped into a routine that uses the same API as the original CRL code.

The task of an OCSP responder was performed by the program `ocsp` of the application directory of OpenSSL using the parameters for the server mode.

The use of the issuer certificate in the web server (apache) is not desirable by two reasons: (a) The issuer certificate is not available at this point in the program. A bigger rewrite action of the code would be necessary to achieve this. However, the issuer certificate would be accessible at some point. For practical reasons in an ideal situation it should not be necessary that the web server has access to this certificate. (b) Therefore it is a wish to do even more of the certificate processing outside the server, and put more functionality into the OCSP responder.[3]

[3]The issuer certificate is still necessary for certificate processing. How to remove the necessity of the issuer certificate in the web server will be looked at at a later stage.

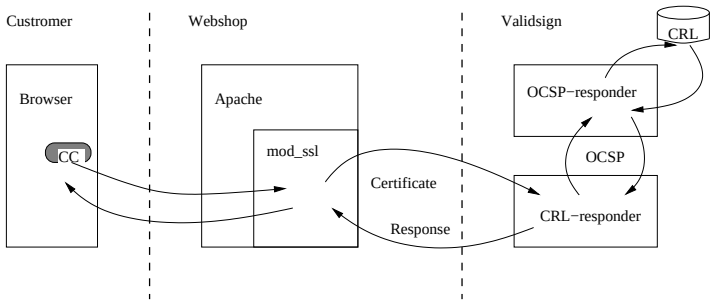



Figure 3: CRL check by sending the certificate to the VAS server

## 2.3. Sending certificates to CRL server

An alternative implementation was done by sending the entire client certificate to a CRL responder. The certificates are available in DER representation and can be written to a BIOS data structure (i.e. TCP connection, or file). As return value an integer value is sent, from which the three possible states can be retrieved. This services is referred to as delegated path validation (DPV) (see Section 1.3).

The routine `ssl_callback_SSLVerify_CRL` is called twice, one time for the issuer certificate and thereafter for the client certificate. For DPV this is unwanted behavior. The reason for that is that the callback is implemented for online revocation check, while the current method implements a DPV. The proper hook into OpenSSL would be the use of `SSL_CTX_set_cert_verify_callback()`, which is currently not used within `mod_ssl`.

On the CRL-server side we implemented a small server application that receives a certificate in DER format. The end of the transmission is recognised by the `END CERTIFICATE` line. The certificate is written to a temporary file, and the `ocsp` application is called (which can check towards a OCSP responder or towards files).

## 2.4. The OCSP Responder

The OpenSSL OCSP application is located in `openssl/apps/ocsp.c`. The OCSP Response verification follows the rules specified in RFC 2560. The call syntax of the application is to some extent described in `man1/ocsp.1` (`http://www.openssl.org/docs/apps/ocsp.html`).

The OCSP responder locates the certificate and checks the signature of the OCSP request using the responder certificate's public key. Then the certificate is verified on the OCSP responder certificate, building up a certificate chain in the process. If this initial verification should fail the OCSP application halts with an error.

The locations of the trusted certificates used to build the chain can be specified by the CAfile and CApath options. Otherwise they will be looked up in the standard OpenSSL certificates directory.

If the issuing CA certificate in the request matches to the OCSP responder certificate the OCSP verification succeeds. Otherwise, the OCSP responder certificate's CA is checked against the issuing CA certificate in the request. If there is a match and the OCSPSigning extended key usage is present in the OCSP responder certificate then the OCSP verification succeeds. If this fails, the root CA of the OCSP responders CA is checked to see whether it is trusted for OCSP signing in the next step. If so the OCSP verification succeeds.

We used the following parameters to launch the OpenSSL OCSP responder application

with our Self Signed CA SSL Certificate (see A.3.2):

```
/opt/bin/openssl ocsp \
    -CA /opt/CA/cacert.pem \
    -index /opt/CA/index.txt \
    -rsigner /opt/CERTS/cert-ocsp-responder-crt.pem \
    -rkey /opt/CERTS/cert-ocsp-responder-privkey.pem \
    -nmin 1 \
    -validity_period 1 \
    -status_age 1 \
    -port 9091
```

# 3. Contributing to Open Source

The implementation had two concerns:

- implementation of the functionality.
- integration of the code with `mod_ssl`.

The integration with `mod_ssl` can be done in several ways:

- Use of a macro definition for changes in the code.
- The use of the EAPI.
- The use of vendor extensions.

Generally, the EAPI and the vendor extensions are not well-documented, as there is no manual or complete code that can be used as a reference. There are some explanations in the file `README.EAPI` of `mod_ssl`, including a partial implementation in the directory `pkg.eapi`. The naming conventions of the functions are not very obvious. The EAPI implementation is invented by the implementor of `mod_ssl`, Ralph S. Engelschall.

The EAPI is a library that makes it possible to register procedure calls with a string. Functions are called by using this string in a special call-function. This functionality implements a sort of "late binding" for functions that might be implemented in dynamically linked libraries.

We found that the EAPI is not very widely used within the code base. We also noticed a tendency that the EAPI is used less in newer versions.

The EAPI is used to implement the `VENDOR_EXTENSIONS`, which are standardised calls to functions that a vendor might specify. The calls of these vendor extensions are built into `ssl_engine_kernel`. However, these calls are not at suitable locations in the code for our purposes. Unfortunately, we only found calls to `ap_hook_use`, but no implementations of the callbacks.

The EAPI is also used in `ssl_engine_ext`. However, the calls in `ssl_engine_ext` are different from those elsewhere in the code.

The API to the EAPI is somewhat difficult to understand. The first parameters include the following values:

- a string that denotes the procedure to be called.
- a bit string that describes the type and number of parameters for in-values.
- a bit string that describes the result values.
- standard parameters that always are used. These parameters could contain context variables or other data types that are attached to the implemented module. However, as there are no examples available, these mechanisms remain somewhat unclear to us.

When using the EAPI separate modules can be implemented. This makes it possible to define separate parameter files for each module. When using macros only, the parameters to `mod_ssl` and its extensions are in the file `httpd.conf`

For our implementation we decided to use a macro `VALIDATION_AUTHORITY`, which denotes all the changes in the code necessary to implement the validation authority. The code embraced by this macro includes the functionality, and the calls to read the parameters from the configuration file. The configuration file includes the following new values:

**UseVA:** Set 1 to switch on, 0 to switch off. This switch does not affect the ordinary CRL processing. The following switches are only in use, when `UseVA` is set to 1.

**VAServiceType:** Set 1 to use use the OCSP protocol for the validation authority service; set 2 to use the WVP protocol[4]. The default value is 1.

**VAPath:** Set the relative location path. The default value is '/'.

**VAHost:** Set the name of the host where the validation authority responder is located.

**VAPort:** Set the number of the port to the validation authority responder.

## 4. Results and Further Work

As a result of this current project we present a working Validation Authority Service. This service consists of a configurable add-on in the `mod_ssl` module of the Apache HTTPD Server that interfaces the service. A simple Validation Authority Server was implemented, is based on of OpenSSL.

We implemented two versions of the VAS. The first version only works with X.509 V3 extensions and uses OCSP, while the second version uses a very simple protocol, but should work with any certificate. A test run example is documented in Appendix A.5 in detail, including excerpts from the log files.

There were several issues not covered within the scope of the current project:

- The OCSP response is not signed. The code base of `ocsp` of OpenSSL contains functionality that provides signing the OCSP response.
- There are several other protocols possible to exchange the certificate and revocation status information between Apache and the validation authority server.
- At a next stage the answer could be extended to contain attributes to a certificate. This functionality is discussed in [3] to some extent.
- The functionality of checking the revocation state can be implemented into web browsers. The Mozilla web browser could be a candidate, as it is an open source product. There are some indications that OCSP functionality already is integrated into Mozilla (see `http://www.mozilla.org/projects/security/pki/psm/help_20/using_certs_help.html#using_certs_validation`).

[4]This denotes a very simple protocol, whereby the certificate is sent to the VAS in DER encoding, and the response is simply the letter '1' for success and the letter '0' for failure.

# A. Manuals

## A.1. Installation

### A.1.1. Overview of modules

We maintained the critical software modules in CVS[16] on cvs.nr.no.
To obtain them, run

```
$ cvs -d :ext:$USER@cvs.nr.no:/nr/project/imedia/PKI-val/cvs co .
```

The latest versions of the modules at the time of this writing were available from

```
ftp://ftp.openssl.org/snapshot/openssl-0.9.7-stable-SNAP-20020313.tar.gz
http://www.apache.org/dist/httpd/apache_1.3.23.tar.gz
http://www.modssl.org/source/mod_ssl-2.8.7-1.3.23.tar.gz
```

### A.1.2. OpenSSL library with OCSP support

```
$ cd openssl
$ sh config --prefix=/opt/
$ make
$ make test
$ make install
$ cd ..
```

### A.1.3. Apache HTTPD with SSL support

```
$ cd mod_ssl
$ ./configure --with-apache=../apache \
              --with-ssl=../openssl \
              --prefix=/opt/ \
              --enable-shared=ssl
$ cd ..
$ cd apache
$ make
$ make install
```

We generated a self signed CA (A.3.2), issued a server certificate (A.3.1), and configured it in `/opt/conf/httpd.conf` as described in Appendix A.5.

## A.2. Launching Apache with SSL support

```
$ /opt/bin/apachectl startssl
```

### A.2.1. Building the ssldump utility for traffic analysis

The following patch was necessary to build ssldump 0.9b2 from `http://www.rtfm.com/ssldump/` against our snapshot version of openssl from `ftp://ftp.openssl.org/snapshot/`.

```
--- ssldump-0.9b2/ssl/ssl_rec.c Fri Nov  3 07:38:06 2000
+++ ssldump-0.9b2.oka/ssl/ssl_rec.c Fri Feb  8 11:11:58 2002
@@ -249,7 +249,7 @@
     if(memcmp(mac,buf,l))
       ERETURN(SSL_BAD_MAC);

+/*     HMAC_cleanup(&hm); */
-    HMAC_cleanup(&hm);
     return(0);
   }

--- ssldump-0.9b2/ssl/ssldecode.c Sat Sep 15 22:49:16 2001
+++ ssldump-0.9b2.oka/ssl/ssldecode.c Fri Feb  8 11:14:41 2002
@@ -52,7 +52,6 @@
 #include <openssl/hmac.h>
 #include <openssl/evp.h>
 #include <openssl/x509v3.h>
+#include <openssl/md5.h>
 #endif
 #include ``ssldecode.h''
 #include ``ssl_rec.h''
@@ -594,7 +593,7 @@
       left-=tocpy;
     }

+/*     HMAC_cleanup(&hm); */
-    HMAC_cleanup(&hm);

     CRDUMPD(``P_hash out'',out);
```

We applied the patch after email discussions with the author and configured ssldump on the server, to trace and verify SSL connections.

```
$ ./configure --prefix=/opt/ --with-openssl=/opt/
$ make install
$ sudo /opt/bin/ssldump -i eth0 >>/var/log/ssldump.log
```

## A.3. Certificates

### A.3.1. Managing PKI Certificates with OpenSSL, Mozilla and Apache

In the following we describe how to generate certificates for a Certification Authority (CA), a Server and a Client in our simple trust chain. The client and server certificates are signed by the CA. For our purposes, we decided to just be our own CA and sign requests and generate new certificates ourselves.

If we had used an official CA to sign our certificates for us, we would send them a certificate request, they would sign it, and send us a public key and a private key back. We would then use the certificates and could jump directly over the steps described in the following two sections.

### A.3.2. Self Signed SSL CA Certification Authority

We choose to use self-signed certificates. We generated our CA using the CA.pl script provided in OpenSSL. We put the public CA key in `/opt/CA/cacert.pem` and the private CA key in `/opt/CA/private/cakey.pem`, copied the public CA key to `/opt/CERTS/cacert.pem` on the

web shop, and enabled it along with the Validation Authority parameters in `/opt/conf/httpd.conf` using the SSLCACertificateFile statement.

### A.3.3. Generating Test Certificates using a Self Signed CA

We created two shell scripts to assist the request generation and signing of certificates using our Self Signed SSL CA Certificate.

### A.3.4. /opt/bin/generate-cert.sh

```
#!/bin/sh
# Author: Ole Aamot <oka@nr.no>
#
# USAGE
# generate-cert.sh <command> <identifier>
#
# COMMANDS
#       create - Create new certificate request
#       sign   - Sign certificate with CA private key
#       export - Export the certificate as PKCS12
#       all    - Do all of the stuff above
#
# REQUIRES
#       OpenSSL binary in /opt/bin/openssl
#               config in /opt/nr-ca.cnf
#               caroot in /opt/CA
#               ca key in /opt/CA/private/cakey.pem
#               cacert in /opt/CA/cacert.pem

set -e

case "$1" in
    create)
    echo "Creating certificate..."
    /opt/bin/openssl req -config /opt/nr-ca.cnf \
-new -keyout /opt/CERTS/cert-$2-privkey.pem \
-out /opt/CERTS/cert-$2-req.pem \
-days 365
    ;;
    sign)
    echo "Signing certificate..."
    /opt/bin/openssl ca -config /opt/nr-ca.cnf \
    -keyfile /opt/CA/private/cakey.pem  \
                    -cert /opt/CA/cacert.pem \
                    -in /opt/CERTS/cert-$2-req.pem \
                    -out /opt/CERTS/cert-$2-crt.pem
    ;;
    export)
    echo "Exporting PKCS12 ccertificate..."
    /opt/bin/openssl pkcs12 -export \
    -in /opt/CERTS/cert-$2-crt.pem \
    -inkey /opt/CERTS/cert-$2-privkey.pem \
    -out /opt/CERTS/cert-$2.p12
    ;;
    all)
    $0 create $2
    $0 sign $2
    $0 export $2
```

```
    ;;
    *)
    echo "Usage $0 [create|sign|export|all] <ID>"
    exit 1
    ;;
esac
exit 0
```

### A.3.5. /opt/bin/revoke-cert.sh

```
#!/bin/sh
# Author: Ole Aamot <oka@nr.no>
#
# USAGE
# revoke-cert.sh <filename>
#
# REQUIRES
#       OpenSSL binary in /opt/bin/openssl
#               config in /opt/nr-ca.cnf
#               caroot in /opt/CA
#               ca key in /opt/CA/private/cakey.pem
#               cacert in /opt/CA/cacert.pem

set -e

/opt/bin/openssl ca -revoke $1 -config /opt/nr-ca.cnf

exit 0
```

We added our newly generated certificates for the SSL server in the /opt/CERTS/ directory.

## A.4. Configuring Certificates In a Web Browser

When we generated our new certificates with the scripts (mentioned in A.3.4), we also got a PKCS12 encoded certificates for the server and the client that we imported into the Mozilla web browser. We verified that client authentication worked against the server.

### A.4.1. Managing Certificate Revocation Lists with OpenSSL

A CRL is a widely published list that contains serial numbers and time stamps of certificates expired or revoked by a Certification Authority. The CRL was signed with the private CA key using a secure one-way hash algorithm, such as md5WithRSAEncryption.

The casual way of updating a CRL database with new lists in OpenSSL seems to be to add the new PEM-encoded X.509 CRLs for SSL to a safe repository and update it by running make.

```
$ cp -a verisign.crl /opt/CRL
$ cd /opt/CRL/
$ make
```

To generate a new CRL using our CA (see A.3.2) we used

```
#!/bin/sh
set -e
/opt/bin/openssl ca \
```

```
    -config /opt/nr-ca.cnf \
    -gencrl \
    -out /opt/CRL/nr-ca.crl
```

We found the following usage useful to view the ingredients of a particular CRL file in plain text:

```
$ /opt/bin/openssl crl -noout -text -in /opt/CRL/nr-ca.crl
Certificate Revocation List (CRL):
        Version 1 (0x0)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no
        Last Update: Feb  8 11:51:36 2002 GMT
        Next Update: Mar 10 11:51:36 2002 GMT
Revoked Certificates:
    Serial Number: 03
        Revocation Date: Nov 28 16:49:10 2001 GMT
    Serial Number: 09
        Revocation Date: Nov 28 16:17:57 2001 GMT
    Serial Number: 0B
        Revocation Date: Dec 17 19:02:40 2001 GMT
    Serial Number: 0C
        Revocation Date: Dec 17 19:03:11 2001 GMT
    Serial Number: 11
        Revocation Date: Dec 19 12:20:05 2001 GMT
    Serial Number: 12
        Revocation Date: Dec 19 12:09:03 2001 GMT
    Serial Number: 14
        Revocation Date: Dec 19 12:34:49 2001 GMT
    Serial Number: 15
        Revocation Date: Dec 19 12:46:32 2001 GMT
    Serial Number: 16
        Revocation Date: Dec 19 12:50:46 2001 GMT
    Serial Number: 17
        Revocation Date: Dec 19 13:00:35 2001 GMT
    Serial Number: 19
        Revocation Date: Dec 19 13:04:46 2001 GMT
    Serial Number: 1A
        Revocation Date: Dec 19 13:23:24 2001 GMT
    Signature Algorithm: md5WithRSAEncryption
        6e:9e:7d:7c:04:97:a5:8c:57:c2:53:be:af:d7:91:8f:0d:d6:
        14:82:58:77:12:07:3b:17:8e:5f:e2:9a:dc:29:8c:36:ff:37:
        ac:7e:a1:79:02:0b:14:f9:b5:b0:5b:0b:f3:72:7f:75:48:01:
        25:02:c6:2f:d2:c2:a5:3c:b6:3a:7a:e1:40:bc:7c:93:7b:86:
        34:a1:e0:24:40:16:61:68:f9:c0:47:ec:34:9f:be:be:f5:a8:
        95:b4:b5:16:60:11:a3:76:27:ac:de:a7:9c:cb:0f:09:dd:77:
        66:88:cd:02:b2:e0:7c:70:05:10:5e:cb:38:fd:ac:49:81:e4:
        65:e6
```

## A.5. Test Case

For testing we used self-signed certificates, signed on `valid.nr.no`. The OCSP responder has access to the data base of this CA. The RP (web shop) has access to the public key of the CA. The Apache HTTPD server of the RP has to check whether the customer certificate is valid by using the VA (validation authority).

In our tests we use the OCSP responder from openssl 0.9.7, which is installed on `valid.nr.no`, port 9091. The customer "Valid Bay" (which is the RP in this example) uses Apache/mod_ssl. The page is accessed on `https://perceptron.nr.no:8443/`.

In the following we demonstrate the steps when web shop customer Per Hansen accesses Valid Bay.

1. "Valid Bay" configures the file `/opt/conf/httpd.conf` with the public key of Per Hansen's issuer, and the `SSLVA*` parameters for `mod_ssl`, which have been defined for the validation authority module. See the following example:

```
SSLCACertificateFile /opt/CERTS/cacert.pem

#   OCSP Validation Authority:
#   Validation Authority
SSLUseVA 1
SSLVAServiceType 1
SSLVAPath '/'
SSLVAHost valid.nr.no
SSLVAPort 9091

#   Client Authentication (Type):
#   Client certificate verification type and depth.  Types are
#   none, optional, require and optional_no_ca.  Depth is a
#   number which specifies how deeply to verify the certificate
#   issuer chain before deciding the certificate is not valid.
SSLVerifyClient require
SSLVerifyDepth  10
```

2. A request for Per Hansen's certificate is generated, the request is signed with the private key of the CA, and a key pair for Per Hansen is generated. The certificate is exported in PKCS12 format, which is imported into the web browser of Per Hansen. The steps to achieve this are the following

```
$ /opt/bin/openssl req -config /opt/nr-ca.cnf \
     -new -keyout per_hansen.key -out per_hansen.req -days 365
$ /opt/bin/openssl ca -config /opt/nr-ca.cnf \
     -keyfile /opt/CA/private/cakey.pem \
     -cert /opt/CA/cacert.pem -in per_hansen.req -out per_hansen.crt
$ /opt/bin/openssl pkcs12 -export -in per_hansen.crt \
     -inkey per_hansen.key -out per_hansen.p12
$ scp per_hansen.p12 oka@naos.nr.no:
```

   We also checked that Per Hansen cannot access Valid Bay on `http://perceptron:8443/` without a valid certificate. An excerpt from the log files is shown in the following:

```
[28/Feb/2002 15:26:20 02844] [error] SSL handshake failed (server perceptron.nr.no:8443,
           client 127.0.0.1) (OpenSSL library error follows)
[28/Feb/2002 15:26:20 02844] [error] OpenSSL: error:140890C7:SSL
           routines:SSL3_GET_CLIENT_CERTIFICATE:
           peer did not return a certificate
           [Hint: No CAs known to server for verification?]
```

3. Per Hansen imports `per_hansen.p12` into his Mozilla 0.9.8 browser (alternatively Internet Explorer is possible):

```
Edit -> Preferences -> Privacy & Security -> Certificates
```

4. Per Hansen accesses Valid Bay `https://perceptron:8443/` using the certificate. Then the log file `/opt/logs/ssl_engine_log` on `perceptron.nr.no` looks as follows:

```
[28/Feb/2002 15:26:20 02844] [error] SSL handshake failed (server perceptron.nr.no:8443,
            client 127.0.0.1) (OpenSSL library error follows)
[28/Feb/2002 15:26:20 02844] [error] OpenSSL: error:140890C7:SSL
            routines:SSL3_GET_CLIENT_CERTIFICATE:peer
            did not return a certificate [Hint: No CAs known to server for verification?]
[28/Feb/2002 15:30:31 02846] [info]  Connection to child 3 established
            (server perceptron.nr.no:8443, client 127.0.0.1)
[28/Feb/2002 15:30:31 02846] [info]  Seeding PRNG with 1160 bytes of entropy
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: Checking certificate
            with serial 0 (0x0) checking CRL from issuer
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no with
            subject /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority:
            OCSP_response_get1_basic(vsresp) at line 1824
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: OCSP_response arrived at line 1729
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: OCSP_response arrived at line 1735
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: good
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: OCSP_response: Got retval 1 at line 1835
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority for certificate sais: Result=1
[28/Feb/2002 15:30:32 02846] [warn]  ValidationAuthority: Returning from VS_ContactVA_OCSP with 1

[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority:
            Checking certificate with serial 30 (0x1E) checking CRL from
            issuer /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no with
            subject /C=NO/ST=Oslo/O=NR CA/OU=Per Hansen/CN=Per Hansen/Email=oka@nr.no
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: OCSP_response_get1_basic(vsresp) at line 1824
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: OCSP_response arrived at line 1729
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: OCSP_response arrived at line 1735
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: unknown
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority: OCSP_response: Got retval 1 at line 1835
[28/Feb/2002 15:30:32 02846] [info]  ValidationAuthority for certificate sais: Result=1
[28/Feb/2002 15:30:32 02846] [warn]  ValidationAuthority: Returning from VS_ContactVA_OCSP with 1

[28/Feb/2002 15:30:32 02846] [info]  Connection: Client IP: 127.0.0.1,
            Protocol: TLSv1, Cipher: RC4-MD5 (128/128 bits)
[28/Feb/2002 15:30:32 02846] [info]  Initial (No.1) HTTPS request
            received for child 3 (server perceptron.nr.no:8443)
[28/Feb/2002 15:30:36 02846] [info]  Subsequent (No.2) HTTPS request
            received for child 3 (server perceptron.nr.no:8443)
[28/Feb/2002 15:30:52 02846] [info]  Connection to child 3 closed
            with standard shutdown (server perceptron.nr.no:8443, client 127.0.0.1)
```

5. Per Hansen's issuer revokes the certificate using the following call on `valid.nr.no`:

   ```
   $ /opt/bin/openssl ca -config /opt/nr-ca.cnf -revoke per_hansen.crt
   ```

   Note, the OCSP responer of openssl must be restartet every time when a certificate's status is changed, e.g., is revoked.

   ```
   $ /opt/bin/openssl ocsp \
        -CA /opt/CA/cacert.pem \
        -index /opt/CA/index.txt \
        -rsigner /opt/CERTS/cert-ocsp-responder-crt.pem \
        -rkey /opt/CERTS/cert-ocsp-responder-privkey.pem \
        -nmin 1 \
        -validity_period 1 \
        -status_age 1 \
        -port 9091
   ```

6. Per Hansen accesses Valid Bay `https://perceptron.nr.no:8443/` again (now with the certificate revoked). The log file `/opt/logs/ssl_engine_log` shows the following:

```
[28/Feb/2002 16:01:33 02845] [info]  Connection to child 2 established
            (server perceptron.nr.no:8443, client 127.0.0.1)
[28/Feb/2002 16:01:33 02845] [info]  Seeding PRNG with 1160 bytes of entropy
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority:
            Checking certificate with serial 0 (0x0) checking CRL from issuer
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no with subject
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: OCSP_response_get1_basic(vsresp) at line 1824
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: OCSP_response arrived at line 1729
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: OCSP_response arrived at line 1735
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: good
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: OCSP_response: Got retval 1 at line 1835
```

```
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority for certificate sais: Result=1
[28/Feb/2002 16:01:38 02845] [warn]  ValidationAuthority: Returning from VS_ContactVA_OCSP with 1
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: Checking certificate
            with serial 30 (0x1E) checking CRL from issuer
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no with subject
            /C=NO/ST=Oslo/O=NR CA/OU=Per Hansen/CN=Per Hansen/Email=oka@nr.no
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: OCSP_response_get1_basic(vsresp) at line 1824
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: OCSP_response arrived at line 1729
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: OCSP_response arrived at line 1735
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: revoked
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority: OCSP_response: Got retval 0 at line 1835
[28/Feb/2002 16:01:38 02845] [info]  ValidationAuthority for certificate sais: Result=0
[28/Feb/2002 16:01:38 02845] [warn]  ValidationAuthority: Returning from VS_ContactVA_OCSP with 0

[28/Feb/2002 16:01:38 02845] [info]  Certificate with serial 30 (0x1E) revoked per CRL from issuer
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no
[28/Feb/2002 16:01:38 02845] [error] Certificate Verification: Error (0): ok
[28/Feb/2002 16:01:38 02845] [error] SSL handshake failed (server perceptron.nr.no:8443, client 127.0.0.1)
            (OpenSSL library error follows)
[28/Feb/2002 16:01:38 02845] [error] OpenSSL: error:140890B2:SSL
            routines:SSL3_GET_CLIENT_CERTIFICATE:no certificate returned
[28/Feb/2002 16:01:38 02844] [info]  Connection to child 1 established (server perceptron.nr.no:8443, client 127.0.0.1)
[28/Feb/2002 16:01:38 02844] [info]  Seeding PRNG with 1160 bytes of entropy
[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority: Checking certificate
            with serial 0 (0x0) checking CRL from issuer
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no with subject
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no
[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority: OCSP_response_get1_basic(vsresp) at line 1824
[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority: OCSP_response arrived at line 1729
[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority: OCSP_response arrived at line 1735
[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority: good
[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority: OCSP_response: Got retval 1 at line 1835
[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority for certificate sais: Result=1
[28/Feb/2002 16:01:39 02844] [warn]  ValidationAuthority: Returning from VS_ContactVA_OCSP with 1

[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority: Checking certificate
            with serial 30 (0x1E) checking CRL from issuer
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no with subject
            /C=NO/ST=Oslo/O=NR CA/OU=Per Hansen/CN=Per Hansen/Email=oka@nr.no
[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority: OCSP_response_get1_basic(vsresp) at line 1824
[28/Feb/2002 16:01:39 02844] [info]  ValidationAuthority: OCSP_response arrived at line 1729
[28/
[28/Feb/2002 16:01:39 02846] [info]  ValidationAuthority for certificate sais: Result=1
[28/Feb/2002 16:01:39 02846] [warn]  ValidationAuthority: Returning from VS_ContactVA_OCSP with 1

[28/Feb/2002 16:01:39 02846] [info]  ValidationAuthority: Checking certificate
            with serial 30 (0x1E) checking CRL from issuer
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no with subject
            /C=NO/ST=Oslo/O=NR CA/OU=Per Hansen/CN=Per Hansen/Email=oka@nr.no
[28/Feb/2002 16:01:39 02846] [info]  ValidationAuthority: OCSP_response_get1_basic(vsresp) at line 1824
[28/Feb/2002 16:01:39 02846] [info]  ValidationAuthority: OCSP_response arrived at line 1729
[28/Feb/2002 16:01:39 02846] [info]  ValidationAuthority: OCSP_response arrived at line 1735
[28/Feb/2002 16:01:39 02846] [info]  ValidationAuthority: revoked
[28/Feb/2002 16:01:39 02846] [info]  ValidationAuthority: OCSP_response: Got retval 0 at line 1835
[28/Feb/2002 16:01:39 02846] [info]  ValidationAuthority for certificate sais: Result=0
[28/Feb/2002 16:01:39 02846] [warn]  ValidationAuthority: Returning from VS_ContactVA_OCSP with 0

[28/Feb/2002 16:01:39 02846] [info]  Certificate with serial 30 (0x1E) revoked per CRL from issuer
            /C=NO/ST=Oslo/L=Oslo/O=NR CA/OU=IMEDIA/CN=Director/Email=d@nr.no
[28/Feb/2002 16:01:39 02846] [error] Certificate Verification: Error (0): ok
[28/Feb/2002 16:01:39 02846] [error] SSL handshake failed
            (server perceptron.nr.no:8443, client 127.0.0.1) (OpenSSL library error follows)
[28/Feb/2002 16:01:39 02846] [error] OpenSSL: error:140890B2:SSL
            routines:SSL3_GET_CLIENT_CERTIFICATE:no certificate returned
```

7. Per Hansen is refused access to Valid Bay. This is shown in the Mozilla browser with the following message:

   Could not establish an encrypted connection because your certificate was rejected by perceptron. Error Code: -12224

# B. Implementation

We introduced ValidationAuthority functionality in mod_ssl 2.8.6. We also refer to the build procedure stated in Appendix A.1.3.

## B.1. mod_ssl/pkg.sslmod/libssl.module

We set -DVALIDATION_AUTHORITY in SSL_CFLAGS to activate the introduced code on compile time.

## B.2. mod_ssl/pkg.sslmod/mod_ssl.c

Configuration parameters introduced in the ssl_config_cmds struct:

```
#ifdef VALIDATION_AUTHORITY
    AP_SRV_CMD(UseVA, TAKE1,
       "Switch Validation Authority off=0, on=1, both=2")
    AP_SRV_CMD(VAServiceType, TAKE1,
       "Choose Validation Authority Service Type: 1=OCSP, 2=WVP")
    AP_SRV_CMD(VAPath, TAKE1,
       "Relative location path, such as '/' or '/cert.cgi'")
    AP_SRV_CMD(VAHost, TAKE1,
       "Host name of Validation Authority responder")
    AP_SRV_CMD(VAPort, TAKE1,
       "Port number of Validation Authority responder")
#endif
```

## B.3. mod_ssl/pkg.sslmod/mod_ssl.h

```
#define VALIDATION_AUTHORITY 1
```

## B.4. mod_ssl/pkg.sslmod/ssl_engine_kernel.c

```
...

#if VALIDATION_AUTHORITY
#include <openssl/ocsp.h>
#endif /* VALIDATION_AUTHORITY */

    /*
     * Additionally perform CRL-based revocation checks
     */
    if (ok) {
#if VALIDATION_AUTHORITY
if (sc->vsUseVA == 1) {
ok = ssl_callback_SSLVerify_VS(ok, ctx, s);
if (!ok) {
errnum = X509_STORE_CTX_get_error(ctx);
}
}
else {
#endif
        ok = ssl_callback_SSLVerify_CRL(ok, ctx, s);
        if (!ok)
            errnum = X509_STORE_CTX_get_error(ctx);
#if VALIDATION_AUTHORITY
}
#endif
    }

...

#if VALIDATION_AUTHORITY
/* The following routines uses OCSP.
 * Note: Certificate extension X509V3 Authority Key Identifier is needed
 */

int VS_magic_hexval(unsigned char c) {
    unsigned char cc;
    if (!isalnum(c)) return (-1);
    if (isdigit(c)) return c - '0';
    cc = tolower(c);
    return (cc - 'a') +10;
}

int VS_magic_convert(unsigned char* instring, unsigned char* outstring) {
  /* returns length */
    unsigned char *pin = instring;
```

```
    unsigned char *pout = outstring;
    int count = 0;

    if (!instring) return (-1);

    while (*pin) {
unsigned char sum = 0;
int val;
count += 1;
sum += ( VS_magic_hexval(*(pin++)) << 4);
if (!*pin) break;
sum += ( VS_magic_hexval(*(pin++)));
*(pout++) = (unsigned char) sum;
if (!*pin) break;
if (VS_magic_hexval(*pin) == (-1) ) pin += 1;
    }
    return count;
}

/*  generates an OCSP Certid of the given input values */
OCSP_CERTID *OCSP_cert_valid_new(const EVP_MD *dgst,
 X509_NAME *issuerName,
 unsigned char *issuerKey,
 ASN1_INTEGER *serialNumber)
{
  int nid;
  unsigned int i;
  unsigned char issuerKeyHash[200];
  int issuerKeyHashLen;

  X509_ALGOR *alg;
  OCSP_CERTID *cid = NULL;
  unsigned char md[EVP_MAX_MD_SIZE];

if (!(cid = OCSP_CERTID_new())) goto err;

alg = cid->hashAlgorithm;
if (alg->algorithm != NULL) ASN1_OBJECT_free(alg->algorithm);
if ((nid = EVP_MD_type(dgst)) == NID_undef)
       {
OCSPerr(OCSP_F_CERT_ID_NEW,OCSP_R_UNKNOWN_NID);
goto err;
}
if (!(alg->algorithm=OBJ_nid2obj(nid))) goto err;
if ((alg->parameter=ASN1_TYPE_new()) == NULL) goto err;
alg->parameter->type=V_ASN1_NULL;

if (!X509_NAME_digest(issuerName, dgst, md, &i)) goto digerr;
if (!(ASN1_OCTET_STRING_set(cid->issuerNameHash, md, i))) goto err;

/* Calculate the issuerKey hash, excluding tag and length */
   issuerKeyHashLen = VS_magic_convert(issuerKey, issuerKeyHash);

     if (!(ASN1_OCTET_STRING_set(cid->issuerKeyHash, issuerKeyHash, issuerKeyHashLen))) goto err;

if (serialNumber) {
  ASN1_INTEGER_free(cid->serialNumber);
  if (!(cid->serialNumber = ASN1_INTEGER_dup(serialNumber))) goto err;
}
return cid;
digerr:
OCSPerr(OCSP_F_CERT_ID_NEW,OCSP_R_DIGEST_ERR);
err:
if (cid) OCSP_CERTID_free(cid);
return NULL;
} /* OCSP_cert_valid_new */

static int add_ocsp_vsigncert (OCSP_REQUEST *req, X509 *vcert, server_rec *s, OCSP_CERTID **vsvid)
{
OCSP_CERTID *vid;
EVP_MD *dgst;

X509_NAME *vcertAuthName = NULL;
X509_EXTENSION *vauthExtension = NULL;
ASN1_INTEGER *vcertSerial = NULL;
ASN1_OBJECT *vauthObject;
ASN1_OCTET_STRING *vauthString = NULL;

void *extr_str = NULL;
char *value = NULL;
unsigned char *p;
CONF_VALUE *val;
X509V3_EXT_METHOD *method;
STACK_OF(CONF_VALUE) *nval = NULL;
CONF_VALUE *keyhashval = NULL;

int num, idx;

if (!req) {
  ssl_log(s, SSL_LOG_WARN,
  "ValidationAuthority: Program Error: no ocsp request ...\n");
  return 0;
}
```

```
if (!vcert) {
  ssl_log(s, SSL_LOG_WARN,
  "ValidationAuthority: No vsigncert specified.\n");
  return 0;
}

dgst = (EVP_MD *) EVP_sha1();
vcertSerial    = X509_get_serialNumber(vcert);
vcertAuthName  = X509_get_issuer_name(vcert);

num = X509_get_ext_count(vcert);
idx = X509_get_ext_by_NID(vcert, NID_authority_key_identifier, 0);
vauthExtension = X509_get_ext(vcert, idx);
vauthObject    = X509_EXTENSION_get_object(vauthExtension);
vauthString    = X509_EXTENSION_get_data(vauthExtension);

if (!(method = X509V3_EXT_get (vauthExtension))) {
  ssl_log(s, SSL_LOG_WARN,
  "ValidationAuthority: No X509V3 Extension for Authority Key Identifier found.\n");
  return 0;
}

p = vauthExtension->value->data;

if (method->it) {
  extr_str = ASN1_item_d2i(NULL, &p, vauthExtension->value->length, ASN1_ITEM_ptr(method->it));
}
else {
  extr_str = method->d2i(NULL, &p, vauthExtension->value->length);
}

if (method->i2v) {
  if (!(nval = method->i2v(method, extr_str, NULL))) {
    ssl_log(s, SSL_LOG_WARN,
    "ValidationAuthority: i2v error on line %i\n", __LINE__);
    return 0;
  }
  keyhashval = sk_CONF_VALUE_value(nval, 0);
}

vid = OCSP_cert_valid_new(dgst, vcertAuthName, keyhashval->value, vcertSerial);
*vsvid = vid;

if(!OCSP_request_add0_id(req, vid)) {
  ssl_log(s, SSL_LOG_WARN,
  "ValidationAuthority: Error Creating OCSP request\n");
  return 0;
}

return 1;
} /* add_ocsp_vsigncert */

static int VS_OCSP_response (OCSP_BASICRESP *bs, OCSP_CERTID *id, long nsec, int maxage, server_rec *s) {

  const int VSIGN_OCSP_RESPONSE_GOOD = 1;
  const int VSIGN_OCSP_RESPONSE_FAIL = 0;

  char *name;
  int i;
  int status, reason;

  ASN1_GENERALIZEDTIME *rev, *thisupd, *nextupd;

  ssl_log(s, SSL_LOG_INFO, "ValidationAuthority: OCSP_response arrived at line %i", __LINE__);

  if (!OCSP_resp_find_status(bs, id, &status, &reason, &rev, &thisupd, &nextupd)) {
    ssl_log(s, SSL_LOG_WARN, "ValidationAuthority: No status found.");
  }

  ssl_log(s, SSL_LOG_INFO, "ValidationAuthority: OCSP_response arrived at line %i", __LINE__);
  if (!OCSP_check_validity(thisupd, nextupd, nsec, maxage)) {
    ssl_log(s, SSL_LOG_WARN, "ValidationAuthority: Status times invalid");
  }

  ssl_log(s, SSL_LOG_INFO, "ValidationAuthority: %s", OCSP_cert_status_str(status));

  if (status != V_OCSP_CERTSTATUS_REVOKED) return VSIGN_OCSP_RESPONSE_GOOD;

  return VSIGN_OCSP_RESPONSE_FAIL;
} /* VS_OCSP_response */

int VS_ContactVA_OCSP(X509 *cert, server_rec *s) {
  SSLSrvConfigRec *sc;
  char *path   = NULL;
  char *vshost = NULL;
  char *vsport = NULL;
  long nsec = (5 * 60);
  long maxage = -1;

  BIO  *vsbio = NULL;

  OCSP_CERTID     *vsvid  = NULL;
```

```
  OCSP_RESPONSE   *vsresp = NULL;
  OCSP_REQUEST    *vsreq  = NULL;

  OCSP_BASICRESP  *bs = NULL;

  int retval = 0;
  int i;

  sc      = mySrvConfig(s);
  path    = sc->vsVAPath;
  vshost  = sc->vsVAHost;
  vsport  = sc->vsVAPort;

  /* ========================================================= */
  /* Connect to host */
  vsbio = BIO_new_connect(vshost);
  if (!vsbio) {
    ssl_log(s, SSL_LOG_WARN,
    "ValidationAuthority: cannot connect to host %s", vshost);
    goto vs_free_and_return;
  }
  BIO_set_conn_port(vsbio,vsport);

  /* we do not use ssl yet, code therefore omitted ... */

  if (BIO_do_connect(vsbio) <= 0) {
    ssl_log(s, SSL_LOG_WARN,
    "ValidationAuthority: cannot connect to host %s port %s", vshost,vsport);
    goto vs_free_and_return;
  }

  /* ========================================================= */
  /* Generate request ... */
  vsreq = OCSP_REQUEST_new();
  if (!vsreq) {
    ssl_log(s, SSL_LOG_WARN,
    "ValidationAuthority: Error generating OCSP request for %s port %s", vshost,vsport);
    goto vs_free_and_return;
  }
  add_ocsp_vsigncert (vsreq, cert, s, &vsvid);

  /* ========================================================= */
  /* send request */
  vsresp = OCSP_sendreq_bio(vsbio,path,vsreq);
  BIO_free_all(vsbio);
  vsbio = NULL;
  if (!vsresp) {
    ssl_log(s, SSL_LOG_WARN,
    "ValidationAuthority: Error querying OCSP responder on host %s port %s", vshost, vsport);
    goto vs_free_and_return;
  }

  /* Now extract the result */

  i = OCSP_response_status(vsresp);

  if (i != OCSP_RESPONSE_STATUS_SUCCESSFUL) {
    ssl_log(s, SSL_LOG_WARN,
    "ValidationAuthority: Bad OCSP response status");
    goto vs_free_and_return;
  }

  bs = OCSP_response_get1_basic(vsresp);

  ssl_log(s, SSL_LOG_INFO,
  "ValidationAuthority: OCSP_response_get1_basic(vsresp) at line %i", __LINE__);

  /* retval: 0 on failure
             1 on success */

  retval = VS_OCSP_response (bs, vsvid, nsec, maxage, s);

  ssl_log(s, SSL_LOG_INFO,
  "ValidationAuthority: OCSP_response: Got retval %d at line %i", retval, __LINE__);

  /* ========================================================= */
 vs_free_and_return:
  //... free all other pending variables ...

  ssl_log(s, SSL_LOG_INFO,
  "ValidationAuthority for certificate sais: Result=%d",
  retval);
  ssl_log(s, SSL_LOG_WARN,
  "ValidationAuthority: Returning from VS_ContactVA_OCSP with %d\n",retval);
  return retval;
} /* VS_ContactVA_OCSP */


/* The following routine sends the entire certificate to a server */
int VS_ContactVA_CERT(X509 *cert, server_rec *s) {
  SSLSrvConfigRec *sc;
  char *path   = NULL;
  char *vshost = NULL;
```

```
  char *vsport = NULL;
  BIO  *vsbio  = NULL;
  char tmpbuf[1024];
  int  tmpbuflen;
  int  retval = 0;

  sc      = mySrvConfig(s);
  path    = sc->vsVAPath;
  vshost  = sc->vsVAHost;
  vsport  = sc->vsVAPort;

  /* ------------------------------------------------------------ */
  /* Connect to host */
  vsbio = BIO_new_connect(vshost);
  if (!vsbio) {
    ssl_log(s, SSL_LOG_WARN,
    "ValidationAuthority: cannot connect to host %s", vshost);
    goto vs_free_and_return;
  }
  BIO_set_conn_port(vsbio,vsport);

  /* we do not use ssl yet, code therefore omitted ... */

  if (BIO_do_connect(vsbio) <= 0) {
    ssl_log(s, SSL_LOG_WARN,
    "ValidationAuthority: cannot connect to host %s port %s", vshost,vsport);
    goto vs_free_and_return;
  }

  /* ------------------------------------------------------------ */
  /* send certificate ... */
  PEM_write_bio_X509(vsbio,cert);
  tmpbuflen = BIO_read(vsbio, tmpbuf, 1024);

  if (tmpbuf[0]=='1') retval = 1;
  else retval = 0;

  /* ------------------------------------------------------------ */
 vs_free_and_return:

  ssl_log(s, SSL_LOG_INFO,
  "ValidationAuthority for certificate sais: Result=%d Answer=%c",
  retval,tmpbuf[0]);

  return retval;
} /* VS_ContactVA_CERT */

int ssl_callback_SSLVerify_VS(
    int ok, X509_STORE_CTX *ctx, server_rec *s)
{
    SSLSrvConfigRec *sc;
    X509_NAME *subject;
    X509_NAME *issuer;
    X509 *xs;
    long serial;
    int i, n, rc;
    char *cp;
    char *cp2;
    X509 *theIssuer;

    /* the following variables are new ... */
    long v_serial;
    int  v_result;

    /*
     * Determine certificate ingredients in advance
     */
    sc      = mySrvConfig(s);
    xs      = X509_STORE_CTX_get_current_cert(ctx);
    subject = X509_get_subject_name(xs);
    issuer  = X509_get_issuer_name(xs);

    theIssuer = NULL;

    v_serial = ASN1_INTEGER_get(X509_get_serialNumber(xs));
    cp  = X509_NAME_oneline(issuer, NULL, 0);
    cp2 = X509_NAME_oneline(subject, NULL, 0);
    ssl_log(s, SSL_LOG_INFO,
    "ValidationAuthority: Checking certificate with serial %ld (0x%lX) "
    "checking CRL from issuer %s with subject %s",
    v_serial, v_serial, cp,cp2);

    switch (sc->vsServiceType) {
    case 1:
    v_result = VS_ContactVA_OCSP(xs,s);
    break;
    case 2:
    v_result = VS_ContactVA_CERT(xs, s);
    break;
    default:
    v_result = 0;
    break;
    }
```

```
    if (!v_result) {
      ssl_log(s, SSL_LOG_INFO,
      "Certificate with serial %ld (0x%lX) "
      "revoked per CRL from issuer %s",
      v_serial, v_serial, cp);
      free(cp);
      free(cp2);
      return FALSE;
    }

    free(cp);
    free(cp2);

    return ok;
}
#endif
/* endif VALIDATION_AUTHORITY */
```

## B.5. mod_ssl/pkg.sslmod/ssl_engine_config.c

We set some default parameters for the SSLSrvConfigRec *sc pointer.

```
void *ssl_config_server_create(pool *p, server_rec *s)
{
    SSLSrvConfigRec *sc;

    ssl_config_global_create();

    sc = ap_palloc(p, sizeof(SSLSrvConfigRec));
    sc->bEnabled               = UNSET;
    sc->szCACertificatePath    = NULL;
    sc->szCACertificateFile    = NULL;
    sc->szCertificateChain     = NULL;
    sc->szLogFile              = NULL;
    sc->szCipherSuite          = NULL;
    sc->nLogLevel              = SSL_LOG_NONE;
    sc->nVerifyDepth           = UNSET;
    sc->nVerifyClient          = SSL_CVERIFY_UNSET;
    sc->nSessionCacheTimeout   = UNSET;
    sc->nPassPhraseDialogType  = SSL_PPTYPE_UNSET;
    sc->szPassPhraseDialogPath = NULL;
    sc->nProtocol              = SSL_PROTOCOL_ALL;
    sc->fileLogFile            = NULL;
    sc->pSSLCtx                = NULL;
    sc->szCARevocationPath     = NULL;
    sc->szCARevocationFile     = NULL;
    sc->pRevocationStore       = NULL;

#ifdef VALIDATION_AUTHORITY
    sc->vsUseVA                = 1;
    sc->vsServiceType          = 1; /* 0=off, 1=OCSP, 2=WVP */
    sc->vsVAPath               = NULL;
    sc->vsVAHost               = NULL;
    sc->vsVAPort               = NULL;
#endif

}
```

We merge parameters.

```
/*
 *  Merge per-server SSL configurations
 */
void *ssl_config_server_merge(pool *p, void *basev, void *addv)
{
    SSLSrvConfigRec *base = (SSLSrvConfigRec *)basev;
    SSLSrvConfigRec *add  = (SSLSrvConfigRec *)addv;
    SSLSrvConfigRec *new  = (SSLSrvConfigRec *)ap_palloc(p, sizeof(SSLSrvConfigRec));
    int i;

    cfgMergeBool(bEnabled);
    cfgMergeString(szCACertificatePath);
    cfgMergeString(szCACertificateFile);
    cfgMergeString(szCertificateChain);
    cfgMergeString(szLogFile);
    cfgMergeString(szCipherSuite);
    cfgMerge(nLogLevel, SSL_LOG_NONE);
    cfgMergeInt(nVerifyDepth);
    cfgMerge(nVerifyClient, SSL_CVERIFY_UNSET);
    cfgMergeInt(nSessionCacheTimeout);
    cfgMerge(nPassPhraseDialogType, SSL_PPTYPE_UNSET);
    cfgMergeString(szPassPhraseDialogPath);
    cfgMerge(nProtocol, SSL_PROTOCOL_ALL);
    cfgMerge(fileLogFile, NULL);
    cfgMerge(pSSLCtx, NULL);
    cfgMerge(szCARevocationPath, NULL);
```

```
    cfgMerge(szCARevocationFile, NULL);
    cfgMerge(pRevocationStore, NULL);

    for (i = 0; i < SSL_AIDX_MAX; i++) {
        cfgMergeString(szPublicCertFile[i]);
        cfgMergeString(szPrivateKeyFile[i]);
        cfgMerge(pPublicCert[i], NULL);
        cfgMerge(pPrivateKey[i], NULL);
    }

#ifdef VALIDATION_AUTHORITY
    cfgMergeInt(vsUseVA);
    cfgMergeInt(vsServiceType);
    cfgMergeString(vsVAPath);
    cfgMergeString(vsVAHost);
    cfgMergeString(vsVAPort);
#endif

#ifdef SSL_VENDOR
    cfgMergeCtx(ctx);
    ap_hook_use("ap::mod_ssl::vendor::config_server_merge",
                AP_HOOK_SIG5(void,ptr,ptr,ptr,ptr), AP_HOOK_MODE_ALL,
                p, base, add, new);
#endif

#ifdef SSL_EXPERIMENTAL_PROXY
    cfgMergeInt(nProxyVerifyDepth);
    cfgMergeString(szProxyCACertificatePath);
    cfgMergeString(szProxyCACertificateFile);
    cfgMergeString(szProxyClientCertificateFile);
    cfgMergeString(szProxyClientCertificatePath);
    cfgMergeString(szProxyCipherSuite);
    cfgMerge(nProxyProtocol, (SSL_PROTOCOL_ALL & ~SSL_PROTOCOL_TLSV1));
    cfgMergeBool(bProxyVerify);
    cfgMerge(pSSLProxyCtx, NULL);
#endif

    return new;
}
```

Functions for parsing the introduced parameters:

```
#ifdef VALIDATION_AUTHORITY
const char  *ssl_cmd_SSLUseVA(
    cmd_parms *cmd, SSLDirConfigRec *dc, char *arg)
{
    SSLSrvConfigRec *sc = mySrvConfig(cmd->server);
    sc->vsUseVA = atoi(arg);
    if (sc->vsUseVA < 0)
        return "vsUseVA: Invalid argument";
    if (sc->vsUseVA > 1)
        return "vsUseVA: Invalid argument";
    return NULL;
}

const char  *ssl_cmd_SSLVAServiceType(
    cmd_parms *cmd, SSLDirConfigRec *dc, char *arg)
{
    SSLSrvConfigRec *sc = mySrvConfig(cmd->server);
    sc->vsServiceType = atoi(arg);
    if (sc->vsServiceType < 0)
        return "vsServiceType: Invalid argument";
    if (sc->vsServiceType > 2)
        return "vsServiceType: Invalid argument";
    return NULL;
}

const char  *ssl_cmd_SSLVAPath(
    cmd_parms *cmd, SSLDirConfigRec *dc, char *arg)
{
    SSLSrvConfigRec *sc = mySrvConfig(cmd->server);
    sc->vsVAPath = arg;   /* do we need strcpy here? */
    return NULL;
}

const char  *ssl_cmd_SSLVAHost(
    cmd_parms *cmd, SSLDirConfigRec *dc, char *arg)
{
    SSLSrvConfigRec *sc = mySrvConfig(cmd->server);
    sc->vsVAHost = arg;   /* do we need strcpy here? */
    return NULL;
}

const char  *ssl_cmd_SSLVAPort(
    cmd_parms *cmd, SSLDirConfigRec *dc, char *arg)
{
    SSLSrvConfigRec *sc = mySrvConfig(cmd->server);
    sc->vsVAPort = arg;   /* do we need strcpy here? */
    return NULL;
}
#endif
```

## References

[1] Eric Rescorla. *SSL and TLS*. Addison-Wesley, Upper Saddle River NJ, 2001. See also `http://www.rtfm.com/sslbook`.

[2] PKIX Working Group. Internet X.509 Public Key Infrastructure, Certificate and CRL Profile. *IETF*, 2002. Available from http://www.ietf.org/internet-drafts/draft-ietf-pkix-ipki-new-rfc2527-01.txt.

[3] Håkon Liberg, Per Myrseth, and Jon Ølnes. Validsign, an open meta pki service. Pki cs paper 5/2001, confidential, PKI Consulting Services AS, 2002.

[4] Heinz Johner, Selei Fujiwara, Amelia Sm Yeung, Anthony Stephanou, and Jim Whitemore. Deploying a public key infrastructure. Redbook sg24-5512-00, IBM International Technical Support Organization, 2000. `http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245512.pdf`.

[5] Håkon Liberg. Kort om pki. Pki cs notat 3/2000, `http://www.pki.no/kort_om_pki_2000.pdf`, PKI Consulting Services AS, 2000.

[6] Jon Ølnes and Håkon Liberg. Samtrafikk mellom sertifikattjenester. Pki cs notat 1/2000, `http://www.pki.no/Samtrafikk_2000.pdf`, PKI Consulting Services AS, 2000.

[7] Jon Ølnes. Digitale signaturer, sertifikater, tillit og ttp-tjenester. Pki cs notat 1/2001, `http://www.pki.no/Digitale_signaturer_2001.pdf`, PKI Consulting Services AS, 2001.

[8] Peter A. Henning. *Taschenbuch Multimedia*. Fachbuchverlag Leipzig, 2 edition, 2001.

[9] Eric Rescorla. An Introduction to OpenSSL Programming, Part I of II. *Linux Journal*, September, 2001. also available on `www.linuxjournal.com`.

[10] Eric Rescorla. An Introduction to OpenSSL Programming, Part II of II. *Linux Journal*, December, 2001. only available on `www.linuxjournal.com`.

[11] M. Meyers, A. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol. *IETF*, RFC 2560, 1999.

[12] Michael Myers, Rich Ankney, Carlisle Adams, Stephen Farrell, and Carlin Covey. Online Certificate Status Protocol, version 2. internet draft, IETF, 2001. Available from `http://www.ietf.org/internet-drafts/draft-ietf-pkix-ocspv2-02.txt`.

[13] A. Malpani, R. Housley, and T. Freeman. Simple Certificate Validation Protocol (SCVP). internet draft, IETF, 2002. Available from `http://www.ietf.org/internet-drafts/draft-ietf-pkix-scvp-08.txt`.

[14] Warwick Ford, Phillip Hallam-Baker, Barbara Fox, Blair Dillaway, Brian LaMacchia, and Jeremy Epstein. XML Key Management Specification (XKMS). W3c note, W3C, 2001. Available from `http://www.w3.org/TR/xkms/`.

[15] Peter Gutmann. X.509 Style Guide. web document, University of Auckland, 2000. Available from `http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt`.

[16] Karl Fogel. *Open Source Development With CVS*. The Coriolis Group, 2000. Available from `http://cvsbook.red-bean.com/`.