Configuring two-way SSL using Estonian EID smartcards in Ubuntu Apache2 webserver

Dokument information	
Creation time	06.02.2019
Client	RIA
Author	Urmas Vanem, OctoX
Version	23.02/1

Version information		
Date	**Version**	**Changes/notes**
06.02.2019	19.02/1	Public version.
20.02.2019	19.02/1	Added chapter *additional configuration options*: firewall and OCSP configuration, default website removal. Changed by Urmas Vanem.
12.12.2019	19.12/1	Added recommendations for securing Apache. Changed by Urmas Vanem.
16.12.2020	20.12/1	Added requirement for end-user certificate to have correct extendedKeyUsage field and "right" certificate issuer. See chapter "(Additional) filtering of user certificate". Changed by Urmas Vanem
17.12.2020	20.12/2	Added directive SSLCADNRequestPath, see chapter "Filtering CA certificates for clients". Changed by Urmas Vanem
13.01.2021	21.01/1	Added demonstrative configuration file. Added HSTS configuration. Changed by Urmas Vanem
21.01.2021	21.01/2	SSLOCSPEnable directive replaced from on to leaf. Updated TLS 1.2 cipher recommendations. Updated TLS version usage recommendations. Variable names in Democonf and document are synchronized. Changed by Urmas Vanem
27.01.2021	21.01/3	Added "mobiil-id" filter. Changed by Urmas Vanem

26.02.2021	21.02/1	Added alternative possibility to filter intermediate certificate authorities supported for authentication using SSLCADNRequestFile directive. Changed by Urmas Vanem
27.04.2021	21.04/1	Support for aged ESTEID-SK 2011 certificates removed. Changed by Urmas Vanem
25.11.2021	21.11/1	Ubuntu version updated to Ubuntu Server 21.10. Apache version updated to 2.4.48. Added guidance for ECC certificates. Updated TLS and cipher recommendations.
21.02.2023	23.02/1	Ubuntu version updated to Ubuntu Server 22.04. Apache version updated to 2.4.55. Updates in virtualhost configuration. Changed by Urmas Vanem

Introduction

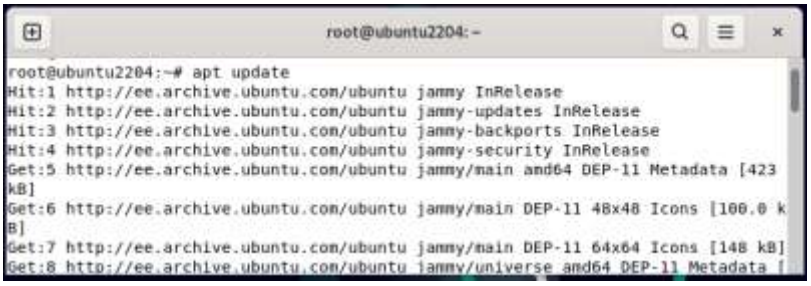
In this guide we describe:

- How to install and configure Apache2 (v. 2.4.55) webserver in Ubuntu Server 22.04!?
- How to configure one-way SSL with Apache webserver.
- How to configure two-way SSL with Apache webserver using Estonian EID cards.
- In addition, we give some security recommendations and check some useful configuration settings.

Apache2 installation and configuration

Installation

1. Renew Ubuntu package data, in terminal run „apt update“.



Picture 1 – updating packages

2. Install Apache2-e, in terminal run „apt install apache2“.

Picture 2 - Apache2 installation

As result of previous steps Apache server is installed now[1]:

[1] Today (21.02.2023) the latest version of Apache is 2.4.55. Version 2.4.52 is included with Ubuntu by default.

Picture 3 – querying Apache version

Let's upgrade Apache to version 2.4.55 now. There are many instructions for this on the internet, and we will not describe this procedure here[2].

Picture 4 – updated Apache version is 2.4.55

Apache2 web server with 4version 2.4.55 now runs in insecure http mode:

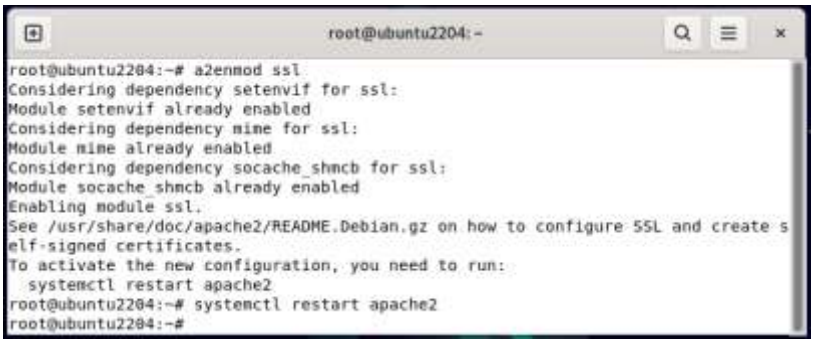
Picture 5 - Apache webserver in default configuration

Configuration

Enabling one-way SSL

Enable SSL for Apache2, in terminal run „a2enmod ssl" and restart Apache2 service.

[2] https://ubiq.co/tech-blog/upgrade-apache-version-ubuntu/ for example.



Picture 6 – Enable SSL and restart service

Creating private key certificate request file

ECC

At first, we create private key based on ECC algorithm:

1. „openssl ecparam -name secp384r1 -genkey -noout -out Apache2204.key“

And then we create certificate request file based on just created private key:

2. „openssl req -new -key Apache2204.key -out Apache2204.csr -subj /C=EE/O=OctoX/CN=Apache2204.octox.demo -reqexts SAN -config <(cat /etc/ssl/openssl.cnf <(printf "[SAN]\nsubjectAltName=DNS:Apache2204.octox.demo,DNS:MYWEBSERVER.octox.demo"))“ [3]

Picture 7 – creating private key and certificate request file

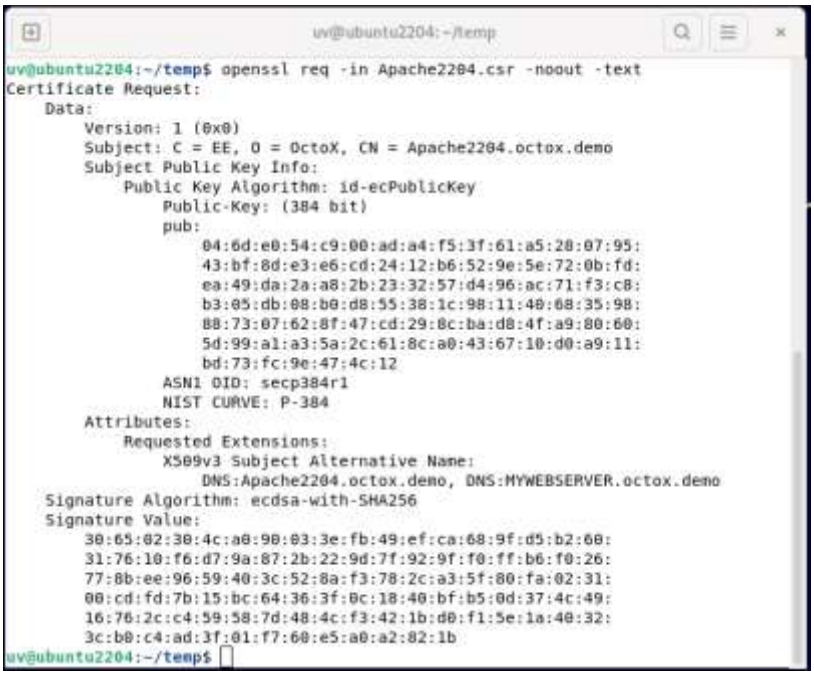
Notes about variables with yellow background:

1. Apache2204.key is private key of certificate;
2. Apache2204.csr is certificate request file for certification authority.
3. CN=Apache2204.octox.demo is *common name* for certificate.
4. DNS:Apache2204.octox.demo and DNS:MYWEBSERVER.octox.demo are SAN DNS names for certificate. These names must correspond to real website names[4]. And naturally the names must be resolvable in name services.

[3] In addition to the certificate attributes C, O and CN described on the command line, it is also possible to describe the attributes L, OU and S if desired. However, only CN can also be used.

[4] Modern browsers do not trust sites where at least one SAN DNS name is not equal to website DNS name.

Contents on certificate signing request file can be viewed by running „openssl req -in Apache2204.csr -noout -text“.

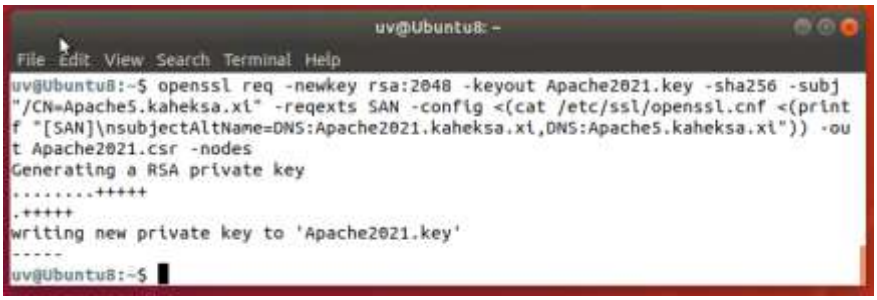


Picture 8 - certificate signing request includes request for two SAN DNS names

RSA

The following section is left here from previous version of document for case if somebody for any reason want to continue using certificates based on RSA algorithm. In following chapters of the document, we continue with certificates based on ECC algorithm.

Now we create certificate signing request and private key, in terminal run „openssl req -newkey rsa:2048 -keyout Apache2021.key -sha256 -subj "/CN=Apache5.kaheksa.xi" -reqexts SAN -config <(cat /etc/ssl/openssl.cnf <(printf "[SAN]\nsubjectAltName=DNS:Apache2021.kaheksa.xi,DNS:Apache5.kaheksa.xi")) -out Apache2021.csr -nodes”.

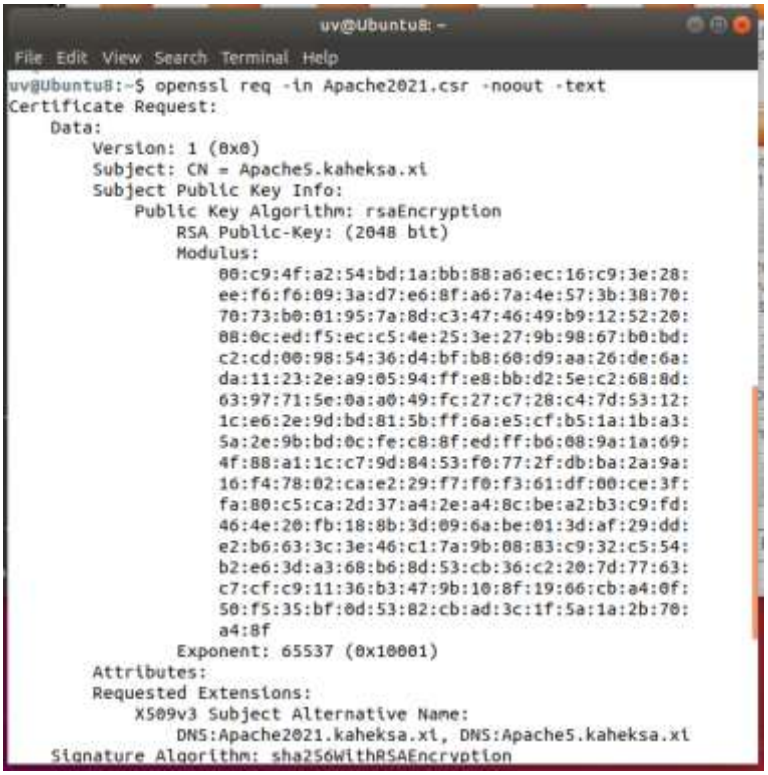


Picture 9 – generating private key and certificate signing request

Notes about variables with yellow background:

1. Apache2021.key is certificate private key.
2. Apache2021.csr is certificate service request.
3. Apache5.kaheksa.xi is a subject name for certificate.
4. Apache2021.kaheksa.xi and Apache5.kaheksa.xi are certificate SAN DNS names. These names must correspond to real website names[5]. And naturally the names must be resolvable in name services.

Contents on certificate signing request file can be viewed by running „openssl req -in Apache2021.csr -noout -text“ in terminal.



Picture 10 – certificate signing request includes request for two SAN DNS names

Certificate request

Certificate signing request file Apache2204.csr should be sent to certificate signer (in our demo environment it is just one test CA). As a response we get signed certificate in Base-64 encoded format that should look like the following picture:

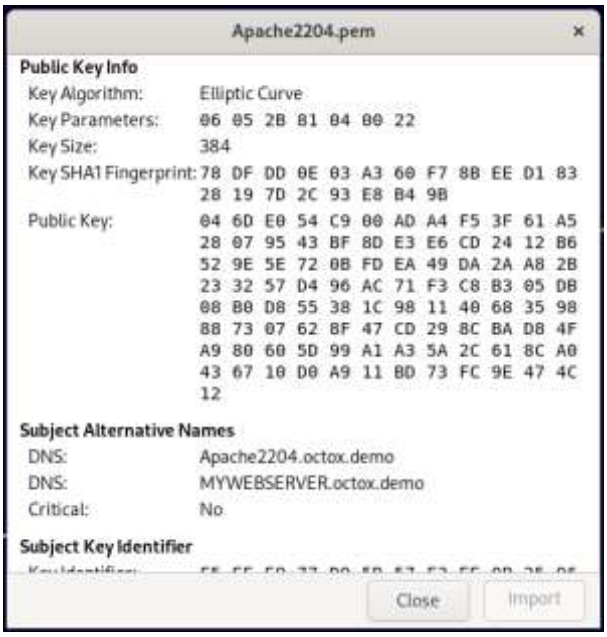
[5] Modern browsers do not trust sites where at least one SAN DNS name is not equal to website DNS name.

Picture 11 – signed certificate in text redactor

In Ubuntu the certificate looks like in the following picture:

Picture 12 – ECC certificate in Ubuntu

The certificate also includes alternative SAN DNS names:



Picture 13 – algorithm and SAN DNS names

As we can see, the certificate issuer is a CA named „OctoX Demo 73“. Now we need to create a certificate file in which both the future web server's TLS certificate and its chain of issuers are located. To do this, we add the issuer's certificate in pem format to the certificate file of the web server in pem format and save the file named Apache2204.pem[6].

Picture 14 - webserver certificarte and chain/issuer/root certificates in Ubuntu

[6] In the previous version of the guide, we also used the SSLCertificateChainFile directive where we described the issuer(s) of the web server's certificate in pem format. However, this is now considered redundant and the SSLCertificateFile directive is sufficient to describe the certificate and it's chain. See also https://httpd.apache.org/docs/2.4/mod/mod_ssl.html#sslcertificatechainfile for more information.

The generated file must be installed in the /etc/ssl/certs folder. In addition, we need to install the private key of the web server certificate in the /etc/ssl/private folder.

Picture 15 – copying RCA certificate to certificates container

Now we have correctly installed all certificates and private key needed by Apache2 for one-way SSL.

Creating virtual website

For SSL configuration demonstration we create separate virtual website. At first, we create home folder for our website /var/www/Apache2204.

Picture 16 – creating website home folder

Then, for testing purposes, we put a simple webpage named index.html in the folder. In our example file /var/www/html/index.html is copied to our new folder. Then minor modifications are made in its' heading or title to understand later it is our website.

Then we prepare virtual site configuration file, in terminal run „nano /etc/apache2/sites-available/Apache2204.conf".

Picture 17 – creating new virtual website configuration file

Now we paste the following configuration in it:

```
# Beginning of file

<Virtualhost Apache2204.octox.demo:80>

# HTTP -> HTTPS redirection
        Servername Apache2204.octox.demo
        redirect / https://Apache2204.octox.demo
</Virtualhost>

<VirtualHost Apache2204.octox.demo:443>
```

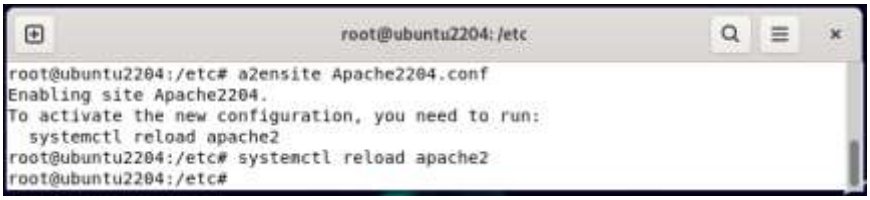
```
# General info
        ServerName Apache2204.octox.demo:443
        DocumentRoot /var/www/Apache2204

# SSL configuration
        SSLEngine on
        SSLCertificateFile /etc/ssl/certs/Apache2204.pem
        SSLCertificateKeyFile /etc/ssl/private/Apache2204.key

# Error collection configuration
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</Virtualhost>

# End of file
```

The new configuration should be activated by running „a2ensite Apache2204.conf“ in terminal. Then restart Apache2 service.



Picture 18 – activating new virtual website and restarting Apache2

Now we configured our new website to use one-way SSL. Also, all HTTP requests to our site http://Apache2204.octox.demo are redirected to HTTPS site https://Apache2204.octox.demo.

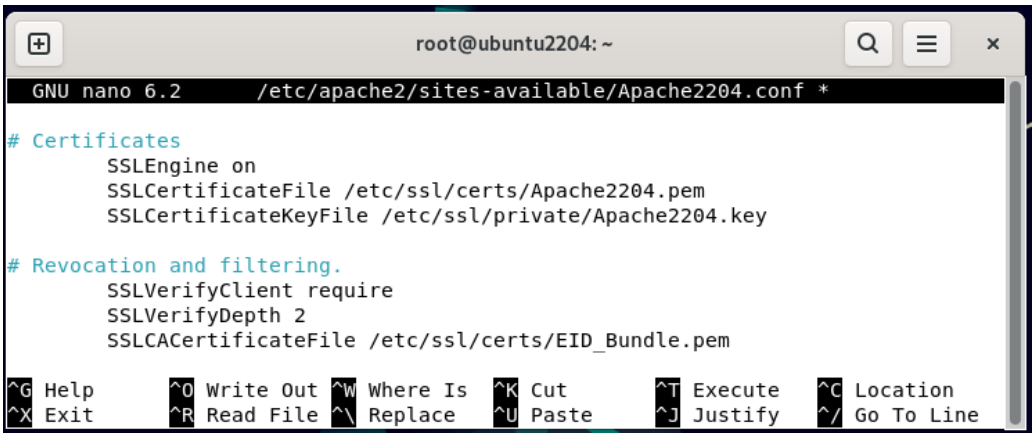
Picture 19 - Apache web server is working and using one-way SSL!

Note. There can be many similar virtual websites with different names in same Apache2 server with single IP address.

Requiring two-way SSL

If we want to require strong Estonian eID client certificate-based authentication, we must update our configuration by adding following lines to our site configuration file Apache2204.conf:

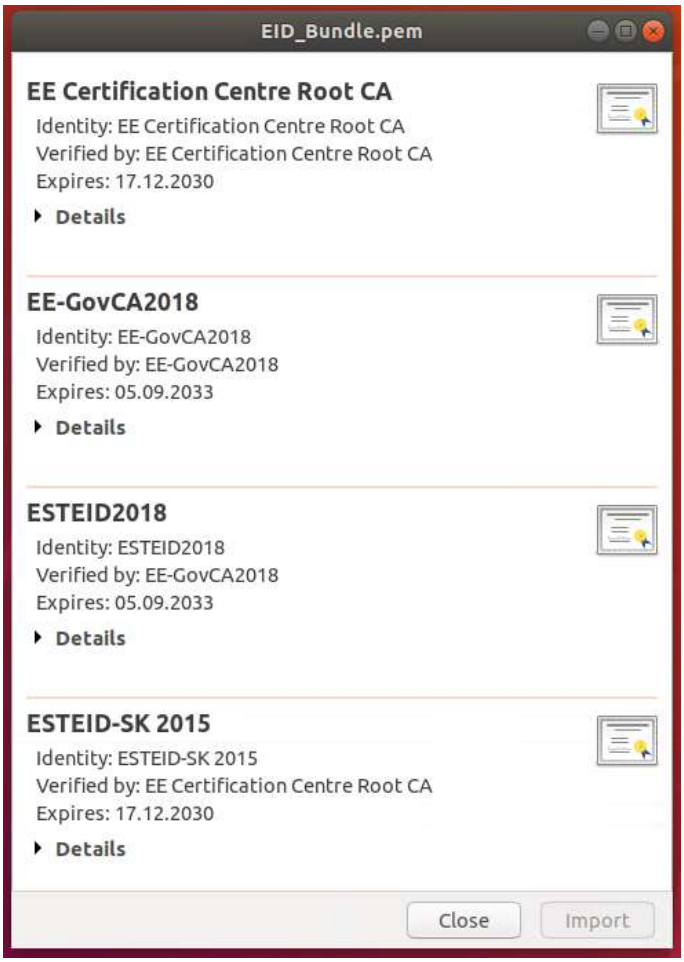
- SSLVerifyClient require
- SSLVerifyDepth 2
- SSLCACertificateFile /etc/ssl/certs/EID_Bundle.pem



Picture 20 – updated configuration file, SSL section

Now we create new text file named EID_Bundle.pem[7], which includes all necessary Estonian EID root- and intermediate certificates (EE-GovCA2018, ESTEID2018**,** EE Certification Centre Root CA, ESTEID-SK 2015) in Base-64 encrypted format. With this file we can initially filter out all client certificates supported by Apache2 web services. File opened in Ubuntu lists all four certificates we want to support:

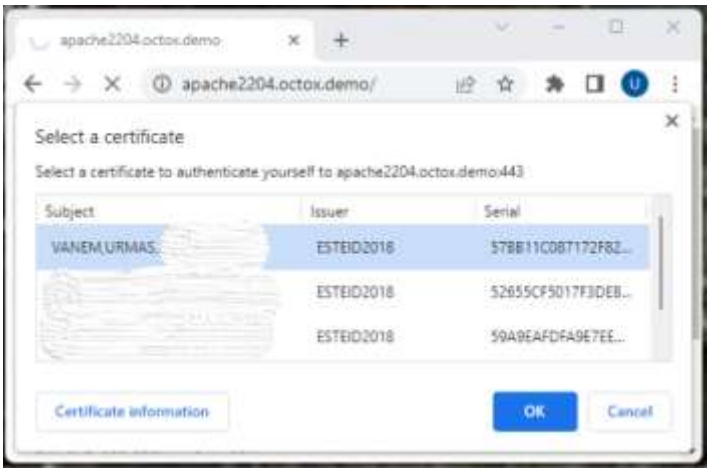
[7] Downloadable: https://installer.id.ee/media/id2019/EID_Bundle.pem



Picture 21 - supported certificates in one file

Copy file EID_Bundle.pem to folder /etc/ssl/certs and restart Apache2 webserver, in terminal run „systemctl reload apache2“.

After accessing website Apache2204.octox.demo now, client certificate is required.



Picture 22 – client certificate request

After confirming the certificate and entering PIN we can access the website! Two-way SSL works.

Demonstrative Apache configuration file based on settings in this document, including some additional configuration options, is downloadable from address https://installer.id.ee/media/id2019/Apache_2.4.55_EID_Demo.conf.

Additional configuration options

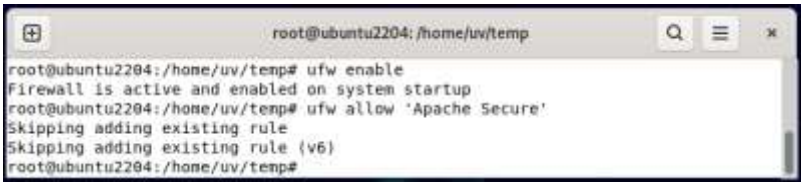
The purpose of this document is not to give exact guidance's how to optimize or protect websites. The main purpose is to show how is possible to configure two-way SSL and use Estonian EID cards for authentication. However, in following section we pay attention on some options which can be useful.

Firewall rules, on demand

By default, firewall is switched off on Ubuntu. To enable firewall, run "ufw enable" in terminal. When firewall is on, we probably want to create some rules to access websites on server. For Apache there are three options:

1. Apache - enables port 80
2. Apache Full – enables ports 80 and 443
3. Apache Secure – enables port 443

For creating firewall rule, run in terminal „ufw allow 'RULE'". For example, enabling HTTPS traffic only, run „ufw allow 'Apache Secure'".



Picture 23 – enabling firewall and creating Apache https rules

Running „ufw status“ in terminal shows us firewall status and active rules:

Picture 24 – firewall is active and HTTPS is enabled

OCSP[8]

Quaranteed OCSP service

By default, configuration described above allows access to Apache2 website for all users with certificates valid in action time. Certificate revocation status is not checked! If we want to check client certificate revocation status using SK guaranteed OCSP service, we first need to make agreement with them (SK). After agreement SK allows client access to guaranteed OCSP service (with address http://ocsp.sk.ee) based on client IP address.

After getting access to the service, we must add following lines to our Apache2 virtual site SSL configuration section:

- SSLOCSPEnable leaf # – enable OCSP certificate revocation check for client certificate.
- SSLOCSPDefaultResponder http://ocsp.sk.ee # - determine OCSP service location.
- SSLOCSPOverrideResponder on # - use defined OCSP responder even if OCSP responder is determined in client certificate.
- SSLOCSPResponderCertificateFile /etc/ssl/certs/EID_OCSP.pem – define OCSP signing certificate in Base-64 encoded format. Today (21.02.2023) the certificate is SK OCSP RESPONDER 2011.[9] (And of course the certificate in base-64 format must be copied to the location mentioned in configuration file (example above).)

Now our SSL configuration looks like presented on following picture:

[8] Client certificate revocation check is also doable with configuring certificate revocation lists (CRL). In this document we do not deal with this because we think configuring OCSP is more reasonable.

[9] Actually the certificate is also trusted by default, because it is issued by CA „EE Certification Centre Root CA“.

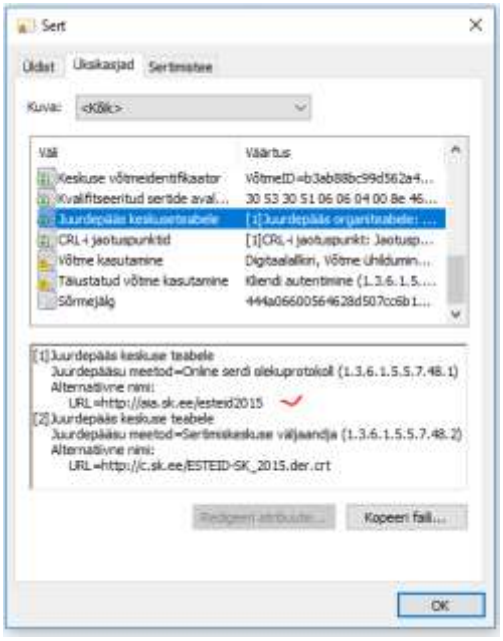
Picture 25 - OCSP part is added to SSL configuration section

Reload Apache2 service with command „systemctl reload Apache2“ in terminal. Now all client certificates are checked against SK guaranteed OCSP service in authentication phase.

AIA-OCSP service

In addition to guaranteed (payable) OCSP service, SK offers also similar AIA-OCSP service for free. AIA-OCSP is basically simple OCSP service with lower availability demands and less functionality. There are different AIA-OCSP service paths for different user certificate chains:

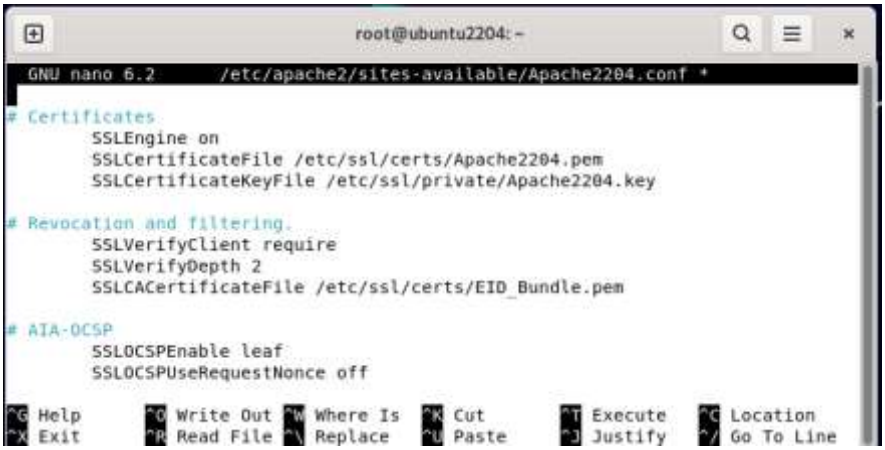
1) Certificates issued by “ESTEID-SK 2015” CA: http://aia.sk.ee/esteid2015
2) Certificates issued by “ESTEID2018” CA: http://aia.sk.ee/esteid2018



Picture 26 - ESTEID-SK 2015 AIA-OCSP path in certificate

To enable client certificate revocation check against AIA-OCSP service, we need to add following lines to our Apache2 virtual website SSL configuration section:

- SSLOCSPEnable leaf # – enable OCSP revocation check for client certificates.
- SSLOCSPUseRequestNonce off # - turn off OCSP service *nonce* requirement.
- # SSLOCSPDefaultResponder # - possibility to describe default OCSP responder on server side. Certificates issued from both chains can be verified against both OCSP paths described above. But certificates already include AIA OCSP path, therefore this directive is not important in our configuration.

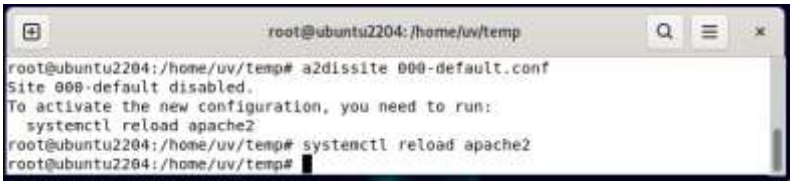


Picture 27 - AIA-OCSP configuration is added to SSL section

Reload Apache2 service with command „systemctl reload Apache2“ in terminal. Now all client certificates are checked against SK AIA-OCSP service determined in certificate in authentication phase.

Default website removal

After Apache2 installation we have also default website installed. To remove default website run command „a2dissite 000-default.conf“ in terminal.



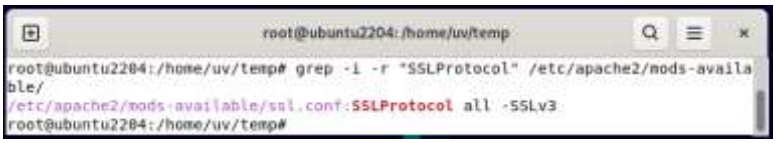
Picture 28 – default website removal

Restart Apache2 service with command „systemctl reload Apache2“ in terminal to apply new configuration.

Recommended security settings for Apache

SSL/TLS

Apache version 2.4.55 is using all SSL/TLS protocols with version higher than SSL3 by default:



Picture 29 - Apache version and SSL/TLS default configuration

The lowest TLS version we want to use in our days is 1.2. Definitely it is very bad idea to support TLS versions lower than 1.2! For a while, we can also use the newest version TLS 1.3.

TLS 1.2 is widely supported and works fine and secure with correct configuration, but today it can be good idea to enable only TLS version 1.3. It is faster, more secure by default and needs less configuration. So, if you have no specific requirements to support TLS version 1.2, you should support TLS 1.3 protocol only! In standard situation, TLS 1.2 should be enabled only if really needed and if it is enabled, it is mandatory to allow only secure ciphers and extensions!

To configure Apache to support only TLS protocol version 1.3, we must add following line to Apache configuration file: „SSLPROTOCOL -all +TLSv1.3“.

```
SSLPROTOCOL -all +TLSv1.3
```

Picture 30 – enabling only TLS version 1.3 in configuration file

To support TLS versions 1.2 and 1.3, add “+TLSv1.2” to configuration line.

If we want to make the change on server level, we must modify parameter SSLPROTOCOL in the file /etc/apache2/mods-available/ssl.conf.

More information about recommendations of using TLS versions can be found in document Kruptoalgoritmid-ning-nende-tugi-teekides-ja-infosusteemides-2021.pdf.

Cipher suites

All TLS 1.3 all cipher suites available today are considered to be safe, so in security point of view we can go on with no additional configuration.

It is different with TLS 1.2. There are many different TLS cipher suites available with Apache version 2.4.55.[10] We can list available cipher suites in Apache with command „openssl ciphers -v“.

By default, only two rules are defined against ciphers:

1) HIGH – some ciphers with key length 128 bits and all stronger are enabled;
2) !aNULL – ciphers not supporting authentication are disabled.

[10] We handle only TLS 1.2 ciphers in this chapter, because lower protocols should be disabled and everything is OK with TLS version 1.3 today.

```
SSLCipherSuite HIGH:!aNULL
```

Picture 31 – server-based configuration in file /etc/apache2/mods-available/ssl.conf

If we want to configure available cipher suites for example for security reason, we can use command line SSLCIPHERSUITE in Apache configuration file. Here we can use predefined aliases or exact cipher suite descriptions.

It is impossible to give exact recommendation for configuring cipher suites because different environments have different requirements. Requirements and possibilities are changing in time. Only recommendation we can give here is to remove non-secure ciphers from list.

Example:

- Using following command line in configuration file: 'SSLCIPHERSUITE "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384"'– only described ciphers are allowed.

You can also configure cipher suites on server level by modifying parameter SSLCIPHERSUITE in file/etc/apache2/mods-available/ssl.conf.

More information about recommendations of using different ciphers can be found in document Kruptoalgoritmid-ning-nende-tugi-teekides-ja-infosusteemides-2021.pdf.

SSLHONORCIPHERORDER

Important parameter related to SSLCIPHERSUITE is also SSLHONORCIPHERORDER. If this parameter has value ON, server list of cipher suites is always preferred! By default, it is not defined and default value is "off".

(Additional) filtering of user certificate

Important! To avoid access to our web service with incorrect certificates we must add following requirement:

1) Certificate must have correct extendedKeyUsage field;
2) Certificate issuer must be "ESTEID2018" or "ESTEID-SK 2015";
3) Certificates issued by "ESTEID-SK 2015" CA must **not** include 'O="ESTEID (MOBIIL-ID)"' in certificate DN.

So, let's add following lines to our Apache configuration:

```
<Location "/">
Require expr ( \
        %{SSL_CLIENT_I_DN_CN} == "ESTEID2018" \
and     "TLS Web Client Authentication, E-mail Protection" in PeerExtList('extendedKeyUsage') \
        or %{SSL_CLIENT_I_DN_CN} == "ESTEID-SK 2015" \
and     "TLS Web Client Authentication, E-mail Protection" in PeerExtList('extendedKeyUsage') \
and %{SSL_Client_S_DN_O} != "ESTEID (MOBIIL-ID)" \
)
</Location>
```

The configuration above can be added to virtual host or to Apache main configuration. After enabling the configuration only certificates with correct extendedKeyUsage field issued form enabled chains not containing “bad” information can access our services.

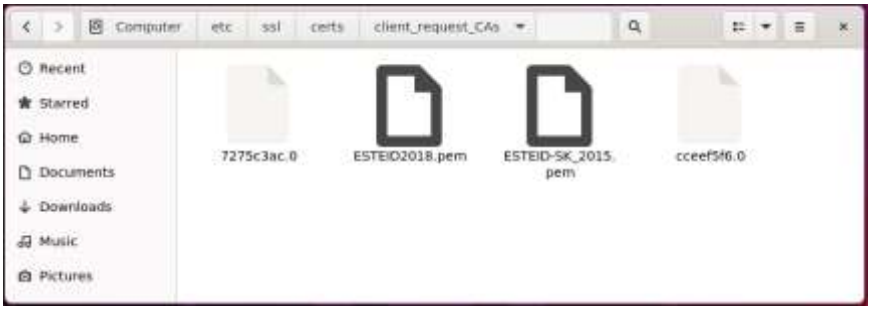
Notes!:

- If you have additional possibilities to filter network traffic, then implementing secure configuration on that level too! SK ID Solutions (hereafter SK) published information about recommended F5 configuration in chapter “Only accept certificates with trusted key usage” in the following article: https://github.com/SK-EID/smart-id-documentation/wiki/Secure-Implementation-Guide
- SK recommendations for secure smart card authentication are published here in chapter “Defence: implement ID-card authentication securely”: https://github.com/SK-EID/smart-id-documentation/wiki/Secure-Implementation-Guide
- Especially good method for filtering good and bad certificates is using OID’s in end user certificates. Unfortunately, we didn’t find method to filter certificate using OID’s on server level. So, if you have possibility to do this in web application level, please do so. Open certificate, read OID in the certificate and if it is not in trusted list, reject the authentication request! All currently known OID’s are listed in chapter “Only accept certificates with trusted issuance policy” in the following article published by SK: https://github.com/SK-EID/smart-id-documentation/wiki/Secure-Implementation-Guide

Filtering CA certificates for clients

By default, when user tries to authenticate him- or herself with certificate to Apache web server, all user authentication certificates are listed for to, even not supported ones. To avoid listing of not supported certificates we can create supported certificate authorities list on server side. For that:

1) Create folder for certificates: mkdir /etc/ssl/certs/client_request_CAs;
2) Put “ESTEID2018” and “ESTEID-SK 2015“ certificates in Base64 format to folder created above;
3) Create certificates hash by running: “openssl rehash /etc/ssl/certs/client_request_CAs”, there are 2 new files/links in the folder now, file names are corresponding to certificate hash:



Picture 32 - intermediate certificates with hashes

4) Add directive “SSLCADNRequestPath /etc/ssl/certs/client_request_CAs” into Apache config file SSL section and save new configuration;

5) Restart Apache server: “systemctl reload apache2”.

Now Apache sends information about supported certificates to client by issuing CA-s and only certificates issued by “ESTEID2018” or “ESTEID-SK 2015” are listed to user in authentication attempt.

Alternative directive for certificate filtering

We can replace directive SSLCADNRequestPath with directive SSLCADNRequestFile. In this case we must create file containing all supported intermediate certification authorities in pem format and describe it in our configuration file with SSLCADNRequestFile directive. As example, “„SSLCADNRequestFile /etc/ssl/certs/SupportedCAs.pem“ enables all intermediate authorities described in SupportedCAs.pem file.

Enabling HTTP Strict Transport Security (HSTS)

Enable mod-headers, run “a2enmod headers” in terminal:

Picture 33 – enabling Apache headers, in our example for HSTS

Add following line to Apache configuration: „Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"“.

```
# Enable HSTS.
        Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"
```

Picture 34 – activating HSTS for 30 days

To apply changes restart Apache service.

Additional possibilities

In addition to TLS and cipher suite configuration there are many other things we can do to secure our server:

- Keep operating system up to date.
- Keep Apache up to date.
- Run Apache under non-root user rights.
- Disable presenting server information.
- Remove unnecessary modules.
- Disable HTTP requests.
- Add and configure *Mod Security*.

- Add and configure *Mod Evasive*.
- Disable directory listing.
- Enable logging.
- …

Please take the list above as short demo recommendations list. Of course, it makes sense to follow the recommendations, but there can be much more you can do to secure your server:

https://www.google.com/search?q=how+to+secure+apache+server.