Configuring two-way SSL using Estonian eID smartcards in Ubuntu Nginx webserver

Dokument information	
Creation time	08.02.2019
Client	RIA
Author	Urmas Vanem, OctoX
Version	23.12/1

Version information		
Date	**Version**	Changes/Notes
08.02.2019	19.02/1	Public version.
28.02.2019	19.02/2	Notes about client certificate revocation check. Default webpage removal. Changed by Urmas Vanem.
12.12.2019	19.12/1	Added recommendations for securing NGINX. Changed by Urmas Vanem.
30.12.2020	20.12/1	Apace version is updated to 20.04.1. NGINX version is updated to 1.19.6. Configuration management is changed (sites… -> conf.d). Added OCSP configuration, recommended security settings and script to filter out certificates issued by “wrong” CAs. Changed by Urmas Vanem
13.01.2021	21.01/1	Added demonstrative configuration file. Added HSTS configuration. Changed by Urmas Vanem
25.01.2021	21.01/2	Changed HSTS configuration recommendations. Changed SSL/TLS and ciphers recommendations. Added some additional security recommendations. Changed by Urmas Vanem
28.04.2021	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. Nginx version updated to 1.21.4. Added guidance for ECC certificates. Updated TLS and cipher recommendations.

		Changed by Urmas Vanem
22.02.2023	23.02/1	Ubuntu version is updated to Ubuntu Server 22.04 and NGINX version is now 1.23.3. The virtual host configuration has also been updated. Changed by Urmas Vanem
18.12.2023	23.12/1	Removed ESTEID-SK 2015 chain. Changed by Urmas Vanem

Intro

In this guide we describe:

- How to install and configure Nginx webserver 1.23.3 on Ubuntu server 22.04?
- How to configure HTTPS (one-way SSL) on Nginx?
- How to configure two-way SSL using Estonian eID cards?
- How to configure revocation check against guaranteed OCSP or against AIA OCSP service?
- How to protect web services?

In addition, we will look at other configuration options such as how to configure HTTP -> HTTPS redirection, etc.

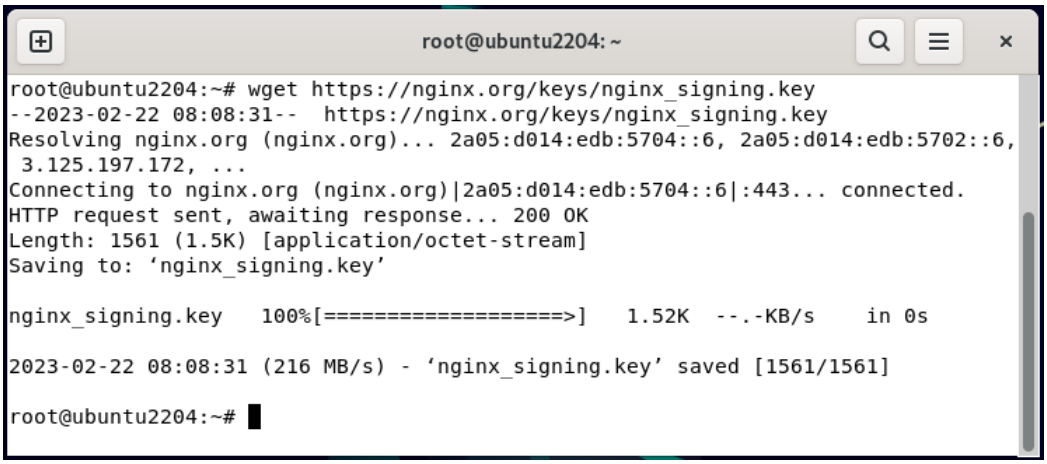
Nginx installation and configuration

Installation

By default, Ubuntu server 22.04 installs NGINX version 1.18. But for many reasons (including OCSP availability since version 1.19) we use the latest version 1.23.3 in our guide.

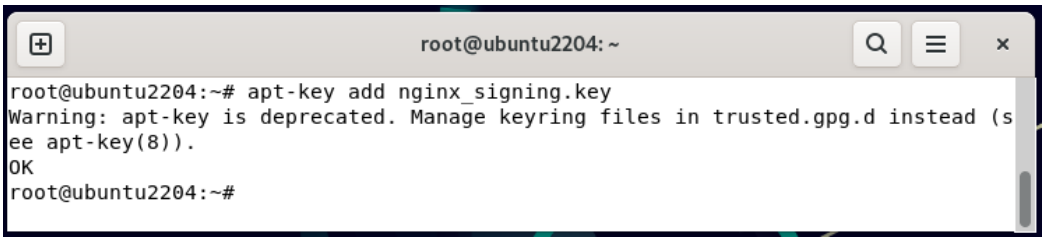
To install NGINX 1.23.3 on Ubuntu server 22.04 (in February 2023):

1. In terminal run: „wget https://nginx.org/keys/nginx_signing.key“.



Picture 1 – getting NGINX key

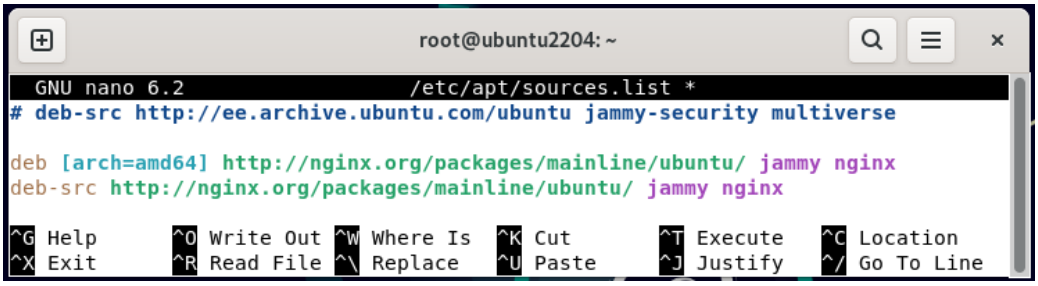
2. In the terminal run following command to add the key: „apt-key add nginx_signing.key“.



Picture 2 – adding key

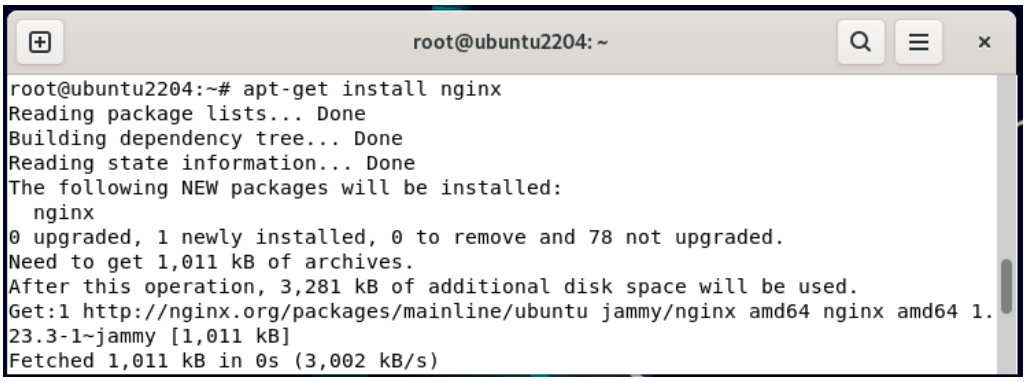
3. Add lines below to the end of file /etc/apt/sources.list (open file with command "nano /etc/apt/sources.list"):

deb https://nginx.org/packages/mainline/ubuntu/ jammy nginx
deb-src https://nginx.org/packages/mainline/ubuntu/ jammy nginx



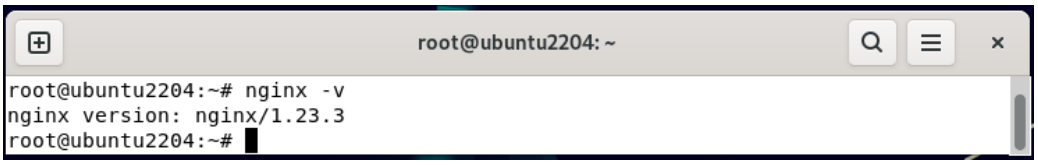
Picture 3 – updating source files

4. If needed, remove other versions of nginx by running terminal command: "apt-get remove nginx-common".
5. In terminal run: „apt-get update".
6. In terminal run: „apt-get install nginx".



Picture 4 – NGINX 1.23.3 installation

From installation windows we can see Ubuntu version 1.23.3 is installed now. We can also check NGINX version with terminal command "nginx -v":



Picture 5 - NGINX version check

Because we usually do not have plans to use default configuration, it is recommended to disable it by running terminal command „mv /etc/nginx/conf.d/default.conf /etc/nginx/conf.d/default.conf.disabled“.

Configuration

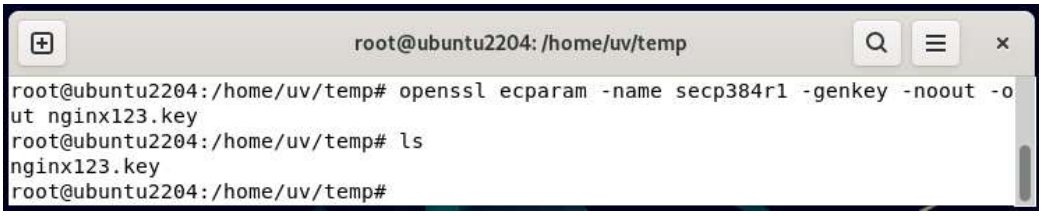
Enabling one-way SSL

Creating private key and certificate request file

ECC

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

1. „openssl ecparam -name secp384r1 -genkey -noout -out nginx123.key“

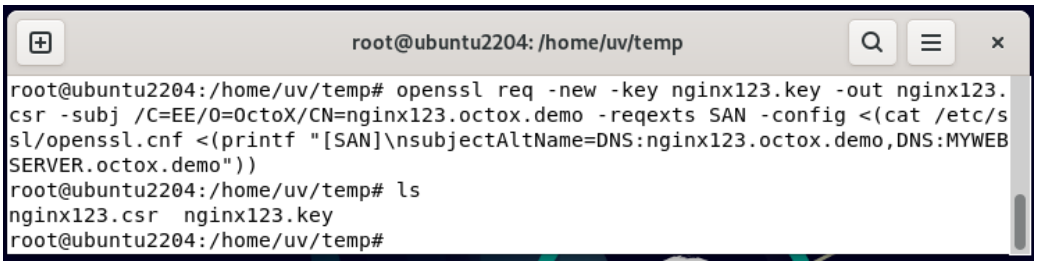


Picture 6 – creating private key

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

2. „openssl req -new -key nginx123.key -out nginx123.csr -subj /C=EE/O=OctoX/CN=nginx123.octox.demo -reqexts SAN -config <(cat /etc/ssl/openssl.cnf <(printf "[SAN]\nsubjectAltName=DNS:nginx123.octox.demo,DNS:MYWEBSERVER.octox.demo"))“[1]

[1] In addition to certificate informational parameters C, O and CN included in command line, it is also possible to add attributes L, OU and S. You can also use only CN here.



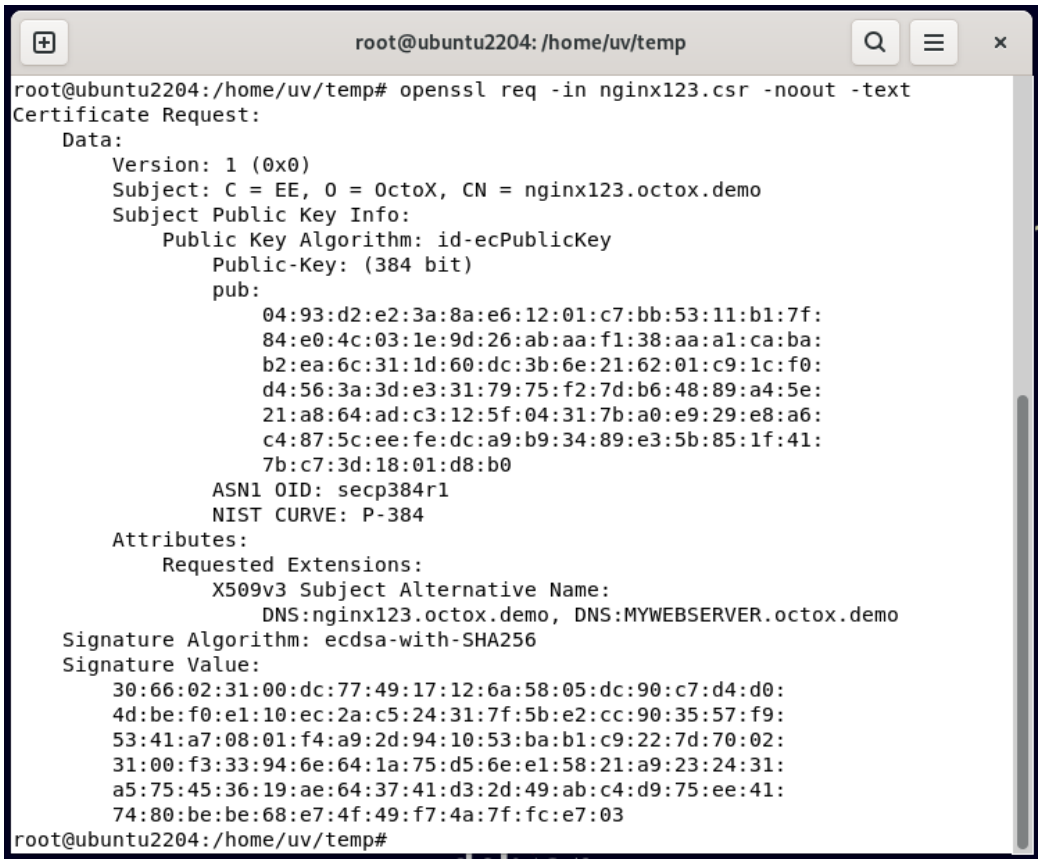
Picture 7 - creating certificate request file

Notes about variables with yellow background:

1. nginx123.key is private key of certificate;
2. nginx123.csr is certificate request file for certification authority;
3. CN=nginx123.octox.demo is *common name* for certificate;
4. DNS:nginx123.octox.demo and DNS:MYWEBSERVER.octox.demo are SAN DNS names for certificate. These names must correspond to real website names[2]. And naturally the names must be resolvable in name services.

Contents on certificate signing request file can be viewed by running „openssl req -in nginx123.csr -noout -text“.

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



Picture 8 – certificate request file includes SAN DNS names

RSA

The following section is left here from the previous version of the document in case somebody for any reason wants to continue using certificates based on RSA algorithm. In the 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 NGINX20PRIV.key -sha256 -subj "/CN=Nginx20.kaheksa.xi" -reqexts SAN -config <(cat /etc/ssl/openssl.cnf <(printf "[SAN]\nsubjectAltName=DNS: Nginx20.kaheksa.xi,DNS: Nginx22.kaheksa.xi ")) -out NGINX20.csr -nodes“.

```
root@Ubuntu2020:/home/uv# openssl req -newkey rsa:2048 -keyout NGINX20PRIV.key -
sha256 -subj "/CN=Nginx20.kaheksa.xi" -reqexts SAN -config <(cat /etc/ssl/openss
l.cnf <(printf "[SAN]\nsubjectAltName=DNS: Nginx20.kaheksa.xi,DNS: Nginx22.kahek
sa.xi ")) -out NGINX20.csr -nodes
Generating a RSA private key
.....................................................................
.....................................................+++++
.................+++++
writing new private key to 'NGINX20PRIV.key'
-----
```

Picture 9 – generating private kay and CSR

Notes about variables with yellow background:

1. NGINX20PRIV.key is certificate private key;
2. NGINX20.csr is certificate service request;
3. Nginx20.kaheksa.xi is a subject name for certificate;
4. Nginx20.kaheksa.xi and Nginx22.kaheksa.xi are certificate SAN DNS names. These names must correspond to real website names[3]. And naturally the names must be resolvable in name services.

Contents on certificate signing request file can be viewed by running „openssl req -in NGINX20.csr -noout -text“ in terminal.

```
root@Ubuntu2020:/home/uv# openssl req -in NGINX20.csr -noout -text
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = Nginx20.kaheksa.xi
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:f1:62:c3:ed:1d:0b:ea:cb:7c:22:17:41:1e:e3:
                    c1:02:a6:7b:f3:72:13:ae:8d:72:72:6f:09:77:d6:
                    51:84:4b:2a:f6:7b:65:9d:9f:f3:2a:0c:16:e5:26:
                    47:70:aa:3e:c8:4c:50:62:5b:6c:2a:49:ea:51:01:
                    60:5c:94:2c:d6:1d:78:70:eb:41:88:6c:09:c8:2f:
                    e4:d5:bb:2f:fb:ec:2f:9d:0c:42:66:b5:de:91:e3:
                    60:62:ff:94:11:21:aa:de:bb:52:bd:20:a6:ff:b4:
                    c3:92:0a:5b:b5:fc:2f:88:bc:44:3e:b4:5b:a4:ec:
                    de:49:16:b6:c0:13:ed:d0:e2:ee:d0:58:bc:cb:36:
                    32:c9:1b:6d:8f:79:db:83:22:fd:fe:a7:9a:b2:cd:
                    26:b1:d7:52:c4:0c:40:6d:0e:49:b5:18:07:c2:3c:
                    c0:c9:70:5d:06:da:0a:e6:01:1a:a4:78:19:aa:a7:
                    38:1c:9d:36:07:4d:db:d2:b5:7b:50:f1:4b:d0:c7:
                    5d:90:86:92:2d:a6:ea:d7:d2:09:8f:51:e8:b6:52:
                    07:b1:1e:5e:ca:65:f3:d4:69:52:f1:d9:47:02:24:
                    98:42:70:83:bc:49:13:c1:92:51:f7:ca:b2:fa:f6:
                    a7:08:13:c1:74:23:d6:58:ab:27:d5:e5:02:20:3f:
                    11:3b
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Subject Alternative Name:
                DNS:Nginx20.kaheksa.xi, DNS:Nginx22.kaheksa.xi
    Signature Algorithm: sha256WithRSAEncryption
         5c:ad:79:7d:be:e1:e0:02:a5:26:11:73:76:b0:77:63:5a:47:
```

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

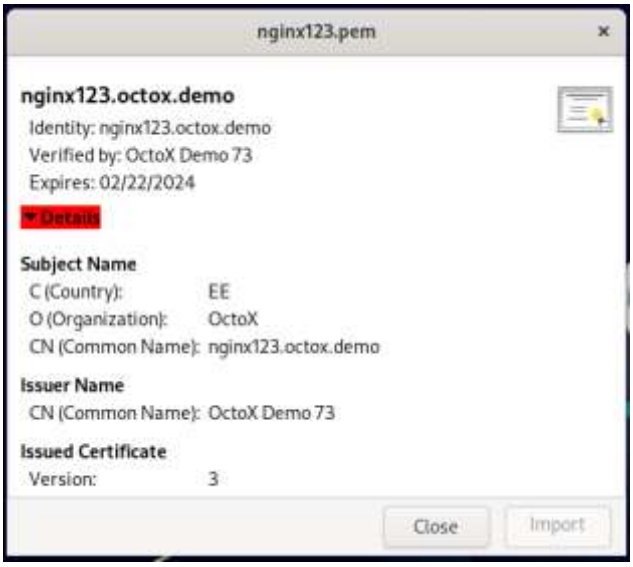
Certificate request

Certificate signing request file nginx123.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 Base64 encoded format.

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

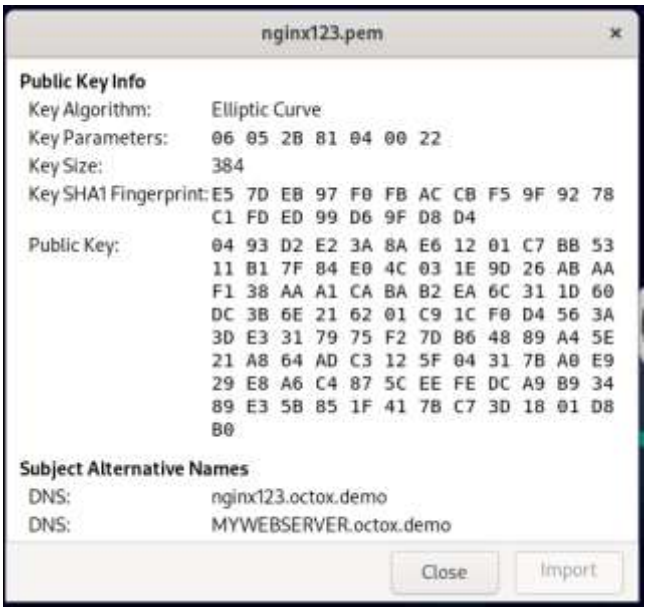
Picture 11 - certificate in pem format in text redactor

Certificate in Ubuntu viewer:



Picture 12 - certificate in Ubuntu

The certificate also includes alternative SAN DNS names:



Picture 13 – algorithm and SAN DNS names

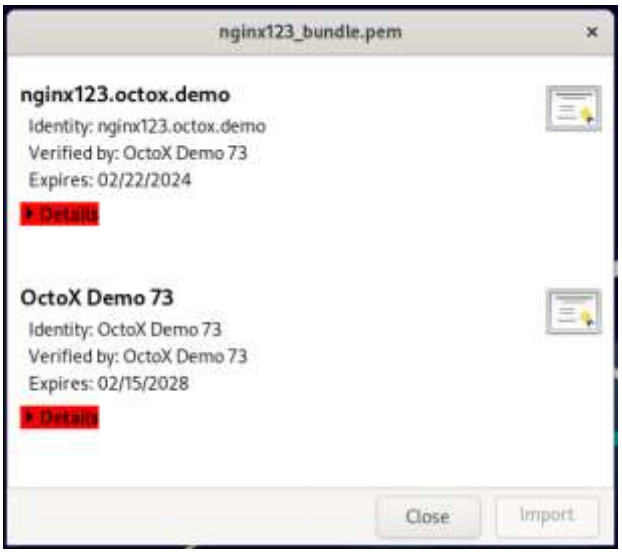
Save new certificate to user work folder in Ubuntu as nginx123.pem.

As we can see, certificate issuer is „ OctoX Demo 73" CA. Now we must get our issuer CA certificate in Base-64 encoded format and save it to user home folder in Ubuntu as OD73.pem.

Nginx does not have option for describing certificates chain, all necessary certificates must be consolidated into one file. First certificate in the file must respond to our certificate with web server private key. So, let's put together certificates nginx123.pem and OctoX_D7.pem and save the file as nginx123_Bundle.pem. We can use any text redactor to do it or run in terminal „cat nginx123.pem OctoX_D7.pem > nginx123_Bundle.pem".

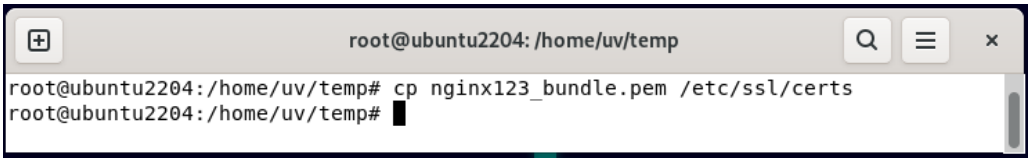
Picture 14 – consolidating certificates to NGINX123_Bundle.pem

Opened in Ubuntu the file looks like on the following picture:



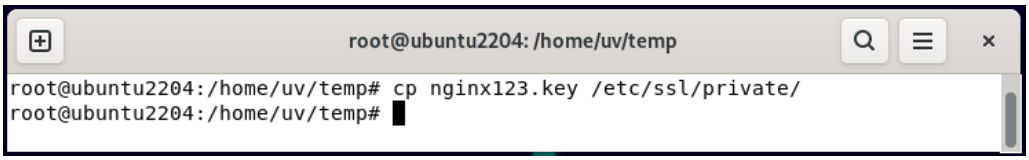
Picture 15 – web server certificate and its' root are consolidated into single file

Let's put the bundled file into correct location by running "cp nginx123_Bundle.pem /etc/ssl/certs" in terminal.



Picture 16 - copying bundled file to certificates container

In addition, we must correctly install private key. Private key must be located in folder /etc/ssl/private. We can do it by running „cp nginx123.key /etc/ssl/private" in terminal.

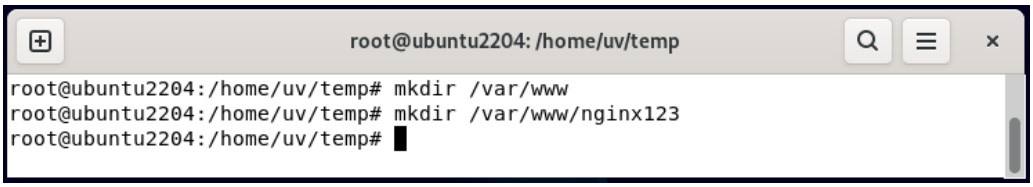


Picture 17 – copying private key

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

Creating virtual website

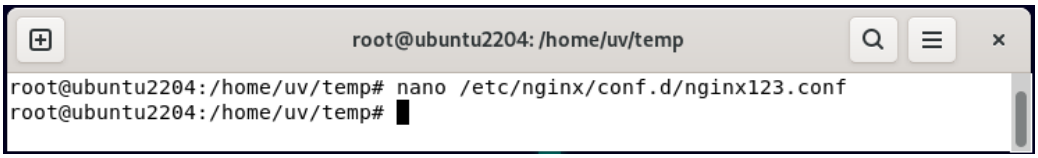
For SSL configuration demonstration we create separate virtual website. At first, we create home folder for content for our website /var/www/nginx123.



Picture 18 – creating root folder for website

For testing purposes, we put a simple and recognizable webpage named index.html into folder /var/www/nginx123.

Then we prepare new virtual site configuration file, in terminal run „nano /etc/nginx/conf.d/nginx123.conf".



Picture 19 – creating new virtual website configuration file

Now we paste the following configuration in it[4]:

```
# Start
server {
      listen 80;
      listen [::]:80;
           server_name nginx123.octox.demo;
           return 301 https://nginx123.octox.demo;
}
server{
           # SSL configuration
           listen 443 ssl;
           listen [::]:443 ssl;
           root /var/www/nginx123;
           index index.html;
           server_name nginx123.octox.demo;

           # Certificates
           ssl_certificate /etc/ssl/certs/nginx123_Bundle.pem;
           ssl_certificate_key /etc/ssl/private/nginx123.key;
```

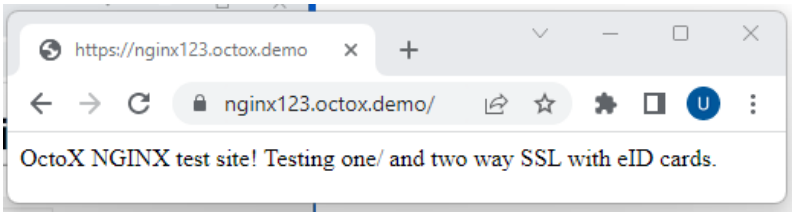
[4] HTTP sections in configuration file are not necessary and are here just like an example.

```
        location / {
        try_files $uri $uri/ =404;
        }
}
# End
```

We can check correctness of our syntax by running terminal command “nginx -t”. If there are no problems with our configuration, we can activate our web service by starting nginx: “systemctl start nginx”. If nginx already runs, we can apply changes by restarting nginx with terminal command “systemctl reload nginx”.

Result

Now we have configured our new website to use one-way SSL and all HTTP requests to our site are redirected to HTTPS. Address http://nginx123.octox.demo will be automatically redirected to https://nginx123.octox.demo.



Picture 20 - Nginx web server is working and using one-way SSL, customized index.html!

Requiring two-way SSL

If we want to require Estonian eID client certificate-based authentication, we must update our configuration by adding following lines to our site configuration file nginx123.conf:

- ssl_client_certificate /etc/ssl/certs/EID_Bundle.pem;
- ssl_verify_client on;
- ssl_verify_depth 2;

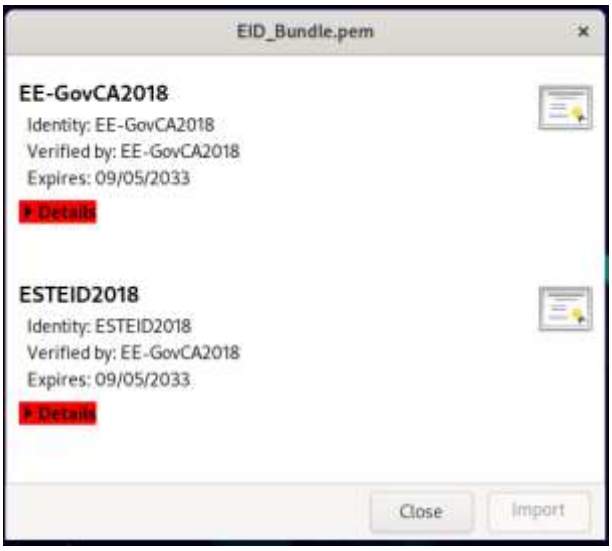
```
# Certificates
ssl_certificate /etc/ssl/certs/nginx123_bundle.pem;
ssl_certificate_key /etc/ssl/private/nginx123.key;
ssl_client_certificate /etc/ssl/certs/EID_Bundle.pem;
ssl_verify_client on;
ssl_verify_depth 2;
```

Picture 21 - updated configuration file, Certificates/SSL section

Now we create new text file named EID_Bundle.pem[5], which includes active Estonian eID root- and intermediate certificates (EE-GovCA2018, ESTEID2018) in Base 64 encrypted format. With this file we

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

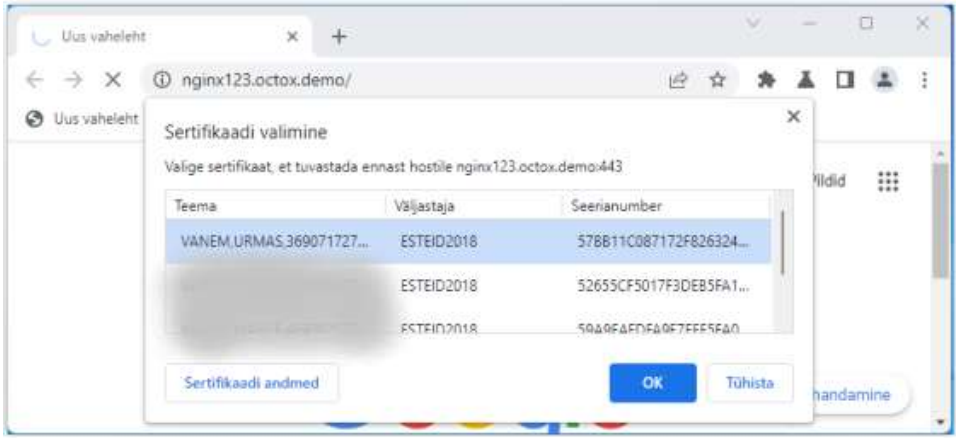
can filter out all client certificates supported by Nginx web services (in other words, on client side only these client certificates are available, which are issued by CA-s/chains listed in our file). In Ubuntu viewer the file looks like shown on the following picture:



Picture 22 – root- and intermediate certificates in one file

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

After accessing website nginx123.octox.demo now, client certificate is required:



Picture 23 - client certificate request

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

Demonstrative Nginx configuration file based on settings in this document, including some additional configuration options, is downloadable from address https://installer.id.ee/media/id2019/NGINX_1.23.3_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 the following section we pay attention to some options which can be useful.

Firewall rules, on demand

To create firewall rules, run the command "ufw allow 'DESIRABLE RULE'" on the terminal. to allow https traffic only, run "ufw allow 443/tcp".

Picture 24 - creating https rule

If firewall is active (ufw enable) running command „ufw status" in terminal shows us active rules.

Picture 25 – firewall is active and HTTPS is enabled

Client certificate revocation check with OCSP[6]

Guaranteed OCSP service

In NGINX, by default all users with time valid and trusted client authentication certificates can access web services. Revocation check is not enabled by default. But it is strongly recommended to perform revocation checking of user certificates to avoid access for users with revoked certificates. To use guaranteed OCSP service offered by "SK ID solutions AS", we must first make contract with them. After agreement we can access OCSP service on address http://ocsp.sk.ee based on certificate or IP.

If we have access to the service, we must add following lines to our NGINX SSL configuration:

- ssl_ocsp leaf;
- ssl_ocsp_cache off;

[6] Client certificate revocation check is also available by using certificate revocation lists (CRLs), but we prefer OCSP solution and do not describe it here!

- resolver 194.126.115.18; [7]
- ssl_ocsp_responder http://ocsp.sk.ee;

```
# Certificates
ssl_certificate /etc/ssl/certs/nginx123_bundle.pem;
ssl_certificate_key /etc/ssl/private/nginx123.key;
ssl_client_certificate /etc/ssl/certs/EID_Bundle.pem;
ssl_verify_client on;
ssl_verify_depth 2;
ssl_ocsp leaf;
ssl_ocsp_cache off;
resolver 194.126.115.18;
ssl_ocsp_responder http://ocsp.sk.ee;
```

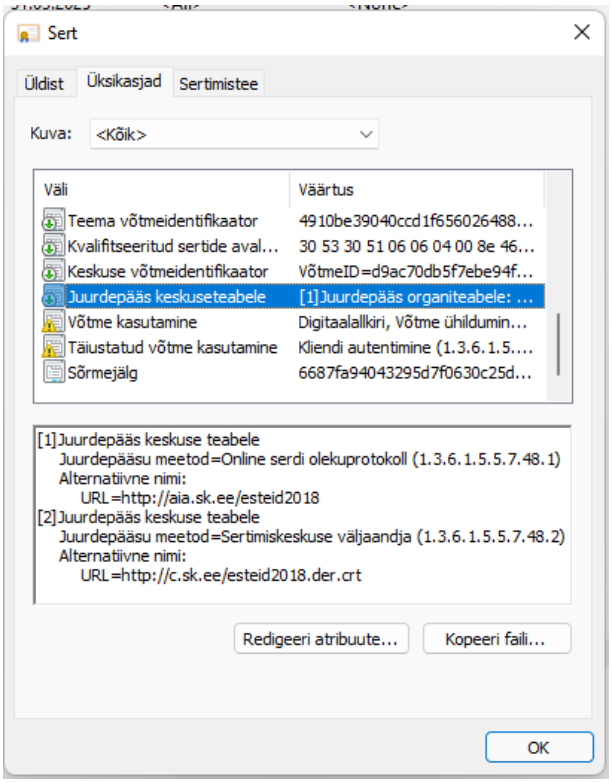
Picture 26 - updated SSL configuration

To apply updated configuration nginx service must be restarted. Afterwards revocation status for all client certificates will be checked against guaranteed OCSP service. When user certificate status is not "GOOD", access to website will be disabled. And I repeat myself here – to use guaranteed OCSP service, agreement with "SK ID Solutions AS" must be confirmed.

AIA OCSP service

In addition to guaranteed (paid) OCSP service with contract, "SK ID Solutions AS" offers also simpler but free access AIA OSCP service. AIA OCSP path is already written into user certificates: ESTEID2018 CA chain certificates: http://aia.sk.ee/esteid2018

[7] „resolver" seems to be optional if direcive ssl_ocsp_responder is defined. We can change IP address value for „resolver" with any DNS server able to solve public DNS names. Probably the best option here is to use your own standard DNS server here, if you want to use it.



Picture 27 – ESTEID2018 AIA OCSP path in certificate

To force client certificate revocation control against AIA OCSP service, we must add following directives to our configuration:

ssl_ocsp leaf;
ssl_ocsp_cache off;
resolver 194.126.115.18;[8]

```
# Certificates
ssl_certificate /etc/ssl/certs/nginx123_bundle.pem;
ssl_certificate_key /etc/ssl/private/nginx123.key;
ssl_client_certificate /etc/ssl/certs/EID_Bundle.pem;
ssl_verify_client on;
ssl_verify_depth 2;
ssl_ocsp leaf;
resolver 194.16.115.18;
```

Picture 28 – configuration with AIA-OCSP service

With configuration above path to OCSP service is read from client certificate.

[8] Resolver – replace the IP address here with any address on DNS server you like. I believe usually it is good to use your ordinary DNS server here.

Recommended security settings for NGINX

SSL/TLS

NGINX version 1.23.3 running on Ubuntu can support old TLS versions like TLS 1.0 or TLS 1.1 by default. Old unsecure TLS protocols with version number lower than TLS 1.2 should definitely no longer be used. TLS 1.2 should be the lowest version to use! For a while, we can also use the newest version of 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 cipher suits and extensions!

To configure Nginx to support only TLS protocol version 1.3, we must add following line to Nginx configuration file: “ssl_protocols TLSv1.3;”

```
ssl_protocols TLSv1.3;
```

Picture 29 – enable only TLS version 1.3 in NGINX 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 ssl_protocols in the file /etc/nginx/nginx.conf.

More information about the recommendations for the use of the TLS protocol can be found in the cryptographic algorithms life cycle report ordered by RIA at https://www.id.ee/en/article/cryptographic-algorithms-life-cycle-reports-2/.

Cipher suites

All TLS 1.3 cipher suites available today are considered to be safe, so from a 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 Nginx version 1.23.3.[9] We can list available cipher suites in Ubuntu with terminal command „openssl ciphers -v“.

If we want to configure cipher suites, we can use command line ssl_ciphers in Nginx configuration file. Here we can use predefined aliases or exact cipher suite descriptions.

It is impossible to give an exact recommendation for configuring cipher suites because different environments have different requirements. Also, requirements and possibilities change in time. The only recommendation we can give here is to remove non-secure cipher suites from the list by describing safe cipher suites in the configuration file.

[9] 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.

Example:

- Using following command line in configuration file: 'ssl_ciphers 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 ssl_ciphers in file /etc/nginx/nginx.conf.

More information about the recommendations for the use of the cipher suites can be found in the cryptographic algorithms life cycle report ordered by RIA at https://www.id.ee/en/article/cryptographic-algorithms-life-cycle-reports-2/.

ssl_prefer_server_ciphers

Preferring server ciphers over client ciphers can be enabled with directive ssl_prefer_server_ciphers. Set its' value to "on" in configuration file.

Additional filtering of client certificates

Important! To guarantee access to our web service with only "right" certificates, it is strongly recommended to add additional requirements:

1) Certificate must include correct OID;
2) Certificate issuer must be ESTEID2018.

Today we do not know how to check existence of correct OID in certificate. So, we recommend to do it in web application level.

To implement second recommendation above, we can check end-user certificate issuer and reject connection if there is wrong one. To implement it lets add following conditions to our configuration (server section):

```
#Determine IMCA and cancel, if not trusted
        set $ocspr "";
    if ($ssl_client_i_dn = "CN=ESTEID2018,organizationIdentifier=NTREE-10747013,O=SK ID Solutions AS,C=EE") {
        set $ocspr "http://aia.sk.ee/esteid2018";
    }
    if ($ocspr = "") {
        return 403;
    }
```

After adding the conditions above to our configuration, any session will be canceled if user certificate is not issued from chain we describe there.

```
# Start
server {
        listen 80;
        listen [::]:80;
        server_name nginx123.octox.demo;
        return 301 https://nginx123.octox.demo;
}
server{
        # SSL configuration
        listen 443 ssl;
        listen [::]:443 ssl;
        root /var/www/nginx123;
        index index.html;
        server_name nginx123.octox.demo;

        # Certificates
        ssl_certificate /etc/ssl/certs/nginx123_bundle.pem;
        ssl_certificate_key /etc/ssl/private/nginx123.key;
        ssl_client_certificate /etc/ssl/certs/EID_Bundle.pem;
        ssl_verify_client on;
        ssl_verify_depth 2;
        ssl_ocsp leaf;
        ssl_ocsp_cache off;
        resolver 194.126.115.18;
        ssl_protocols TLSv1.3;
        #ssl_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384;
        ssl_prefer_server_ciphers on;

        # Other recommended security and optimization settings
        add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload" always;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 1h;
        ssl_session_tickets on;

        #Determine IMCA and cancel, if not trusted
        set $ocspr "";
        if ($ssl_client_i_dn = "CN=ESTEID2018,organizationIdentifier=NTREE-10747013,O=SK ID Solutions AS,C=EE") {
        set $ocspr "http://aia.sk.ee/esteid2018";
        }
        if ($ocspr = "") {
        return 403;
        }

        location / {
        try_files $uri $uri/ =404;
}
}
```

Picture 30 - server configuration

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 recommended 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

Enabling HTTP Strict Transport Security (HSTS)

To enable HSTS for Nginx website, add line "add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload" always;" to server configuration.

```
# Other recommended security and optimization settings.
        ssl_prefer_server_ciphers on;
        add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload" always;
        ssl_session_cache   shared:SSL:10m;
        ssl_session_timeout 1h;
        ssl_session_tickets on;
```

Picture 31 - activating HSTS

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 Nginx up to date.
- Disable presenting server information.
- Disable HTTP requests.
- Install and Configure Naxsi.
- Monitoring with Monit.
- Configure X-XSS Protection.
- Configure X-Frame-Options.
- Configure X-Content-Type-Options.
- Configure Content Security Policy (CSP).
- …

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+nginx+server.