onway EST CA

User Manual

2026-07-09

onway ag, Switzerland

1 Introduction

The onway EST CA implements a X.509 Certificate Authority and provides a frontend using the *Enrollment over Secure Transport* protocol (EST, RFC 7030).

Compared to other certificate management protocols, EST is relatively simple and builds upon established technologies, namely HTTPS transfers, PKCS#10 certificate requests and PKCS#7 containers. Using these foundations, certificates can be enrolled and renewed using dedicated EST clients or simple scripts using widely available command line utilities.

EST does provide some common CA operations through the CMC protocol, only (*Certificate Management over CMS*, RFC 5272). CMC, however, is a very complex protocol. Many EST clients do not support it, and unlike EST operations, scripting CMC operations using command line utilities is infeasible. The onway EST CA therefore implements some backward-compatible extensions to the EST protocol, for example to revoke certificates or approve certificate requests.

The first chapter of this manual covers the installation and configuration of one or more Certificate Authorities using the onway EST CA software. This includes user authentication and authorization and certificate policy configuration. The second chapter discusses standard and proprietary EST operations and examples how to use them using simple command line utilities.

2 Configuration

2.1 Installation

onway provides the EST CA software as a bundled Debian package for Ubuntu Server systems. The EST CA daemon, called `ester`, operates under the unprivileged user `ester`. The package includes a *systemd* init script, creates the user and appropriate directories. It is installed from a pre-configured onway package server using `apt install ester`.

2.2 Configuration Files and State

The directory for `ester` configuration files and the certificate storage are located under `/var/lib/ester`. Each (intermediate) CA served by `ester` has its own subdirectory.

Such a CA specific directory must be named after the DNS hostname the CA instance operates its EST endpoint under. Each dedicated CA requires a dedicated DNS name pointing to the host the `ester` daemon is running on. EST clients use this hostname to select the associated CA implicitly using TLS *Server Name Indication* (SNI). This implies that clients must support the TLS SNI extension.

Under each CA directory, the following files must be configured by an administrator:

File	Description
`cert.pem`	PEM encoded CA issuer certificate
`key.pem`	PEM encoded CA private key
`server.pem`	Optional PEM encoded web server certificate chain
`auth.yaml`	HTTP user authentication and authorization configuration
`policy.yaml`	Certificate Authority policy configuration

For any defined CA directory, all files are mandatory. The format of the issuer credentials and the syntax of the YAML configuration files are detailed in the subchapters below.

Some configuration options in YAML files use relative times. Relative times can be given as integers, specifying the time in seconds. Alternatively, relative times may have the unit suffix `s` for seconds, `m` for minutes, `h` for hours, `d` for days, `w` for weeks or `y` for years when given as string[v3.9.0]. Options taking relative times in this format have the *reltime* type.

The EST server stores the content and state of certificates and certificate requests in additional subdirectories, all taking PEM encoded files:

Directory	Description
`active`	Certificates currently active and usable
`renewed`	Certificates still valid but superseded
`expired`	Certificates with expired lifetime
`revoked`	Certificates revoked but with valid lifetime
`trusted`	Additional trust-chain certificates for CA issuer certificate
`pending`	Certificate requests waiting for approval
`approved`	Certificate requests having been approved
`rejected`	Certificate requests having been rejected

These subdirectories are created implicitly on `ester` startup. Files must not be modified or renamed while `ester` is running. With the exception of `trusted`, certificates in these directories are mainly for inspection, but not for modification by the administrator.

The `trusted` directory can be used to add additional Root and Intermediate CA certificates that build the trust-chain down to the CA issuer certificate; these are served through the `/cacerts` EST operation and to EST TLS clients.

2.2.1 CA Issuer Credentials "`cert.pem`" and "`key.pem`"

The `cert.pem` contains a single PEM encoded X.509 certificate that is used to issue certificates under this CA. The `key.pem` must contain a private key matching the public key contained in the issuer certificate. It must be PEM encoded and contain either a RSA or a PKCS#8 wrapped key (using `BEGIN RSA PRIVATE KEY` or `BEGIN PRIVATE KEY` PEM headers). The onway EST CA currently supports private keys for the RSA cryptosystem, as well as keys using the Ed25519 public key algorithm.

Due to the headless operation of the onway EST CA server, private keys must not be encrypted. Instead, they must not be readable by non-root users. `ester` uses an isolated process for private key operations, and the daemon user `ester` must not have access to the private key directly. Additional measures to secure the private key should be implemented on the host system, for example by using full-disk encryption or a Linux security module.

The `ester --csrgen` command can assist in setting up new CA directories. It generates a new private key if it is missing and produces a certificate request (see `--help`). Example:

```
ester --confdir /var/lib/ester --user ester \
  --csrgen ca.example.com --subject "C=CH, O=Example Org, CN=Example CA"
```

2.2.2 Web server credentials in "`server.pem`"

If no `server.pem` file is configured for a CA, the daemon automatically generates a short-lived certificates to serve as TLS web server certificate using the DNS hostname of the CA configuration. The certificate uses the same public key algorithm as the issuer certificate, directly signed by the serving EST CA.

If a `server.pem` [v3.10.1] exists, it must contain a PEM encoded RSA or PKCS#8 private key (using `BEGIN RSA PRIVATE KEY` or `BEGIN PRIVATE KEY` PEM headers) concatenated to the

PEM encoded X.509 certificate for this private key, in any order. The certificate is used as TLS web server certificate. It must contain a DNS subjectAltName for the DNS hostname of the CA configuration and the TLS Server (*1.3.6.1.5.5.7.3.1*) Extended Key Usage.

Additionally, the `server.pem` file may contain additional intermediate CA certificates, in any order, to send to TLS clients for server certificate validation.

2.2.3 User Configuration "`auth.yaml`"

The user configuration is used to authenticate CA operators using HTTP basic or TLS client certificate authentication within HTTPS. Further, it defines the authorization settings for such an operator, allowing certain EST operations or restricting them specifically.

The `auth.yaml` root section accepts the following sections:

Option	Type	Description	Mandatory
`users`	string	HTTP basic authenticated user definitions	No
`cert`	string	TLS client certificate authenticated user definitions	No

HTTP basic authenticated users

The `users` section contains further subsections named after the HTTP authentication username. Under each user section, the following options can be configured:

Option	Type	Description	Mandatory
`password`	string	Hashed user password string	Yes
`issue`	object	Certificate issue permissions	No
`list`	object	Certificate listing permissions	No
`revoke`	object	Certificate revocation permissions	No
`audit`	object	CA auditing permissions	No

For additional security, the password is provided in hashed form using the *crypt*[1] format. The hash algorithms `$1$` (MD5), `$5$` (SHA-256) and `$6$` (SHA-512) are supported. Password hashes can be generated using the `openssl passwd` utility with the `-1`, `-5` or `-6` options.

TLS client certificate authenticated users

The subsections under the `cert` [v3.10.0] section are named by Relative Distinguished Name *fnmatch* patterns. The client certificate subject selects the user section by matching all subject RDNs to pattern RDNs. Each user section takes the following options:

Option	Type	Description	Mandatory
`issuer-chain`	list[string]	User certificate issuer chain	No
`issue`	object	Certificate issue permissions	No
`list`	object	Certificate listing permissions	No
`revoke`	object	Certificate revocation permissions	No
`audit`	object	CA auditing permissions	No

[1] `https://en.wikipedia.org/wiki/Crypt_(C)`

Option	Type	Description	Mandatory
`self-renew`	object	Renewal of own certificates	No
`self-list`	object	Listing own certificates	No
`self-revoke`	object	Revoking own certificates	No

The `issuer-chain` takes a list of issuer certificate Distinguished Name subjects under which the client certificate has been issued. It must include the full chain starting at the topmost issuer certificate that is installed as trusted certificate in the CAs `trusted` directory. If the `issuer-chain` is omitted, the client certificate must have been issued directly by the serving CA itself.

User authorization configuration

The user permissions are defined in additional subsections under each user specific `users` and `cert` section. If the `issue` section is present, the user may issue certificates. It takes the options below:

Option	Type	Description	Mandatory
`lifetime`	reltime	Maximum certificate lifetime, in s	Yes
`respect-issuer-lifetime`	bool	Restrict lifetime to issuer lifetime	No
`subject`	string	*fnmatch* pattern RDNs to restrict subject to	No
`subject-altname`	string	*fnmatch* pattern to restrict subjectAltNames to	No
`blocks`	list[string]	Subnets to restrict address blocks to	No
`eku`	list[string]	ExtendedKeyUsages to restrict to	No
`ca`	object	Allow issuing intermediate CAs	No
`require-approval`	bool	User requires request approval	No

The `lifetime` option is required to issue certificates, and defines the lifetime of the issued certificate, starting at the issuing time. If explicitly requested, certificates can be issued with shorter lifetimes. By default, the maximum lifetime of a certificate is further restricted to the issuer certificate lifetime. By setting the `respect-issuer-lifetime` v4.4.0 option to `false`, certificate lifetimes may exceed the issuer certificate lifetime.

If a `subject` is defined, only certificates with the specified RDN OIDs are allowed, in exact order, and each RDN in the certificate subject must match the *fnmatch*[2] pattern specified for each RDN in the given RFC4514-compatible `subject` string. If a `subject-altname` is specified, any requested subjectAltName must match the *fnmatch* pattern; only DNS and E-Mail subjectAltNames are currently supported.

If `blocks` is specified, the RFC 3779 address block to issue in certificates must be part of one of the specified CIDR subnets. If the `blocks` section is omitted, the blocks are instead validated against the issuer certificate address blocks.

If an `eku` list is defined, Extended Key Usage extension OIDs are allowed in issued certificates if they are explicitly listed, only. Extended Key Usage OIDs must be requested in the

[2] `https://www.man7.org/linux/man-pages/man3/fnmatch.3.html`

certificate request, whereas Key Usage extensions are ignored. Adding Extended Key Usage OIDs to certificates implicitly adds a Key Usage extension with appropriate values[v4.0.0]. The following Extended Key Usage values are supported, adding the listed Key Usage bits:

`eku` value	Description	OID	Implicit *Key Usage*
`server`	TLS Server	*1.3.6.1.5.5.7.3.1*	*DigitalSignature*, *KeyEncipherment* (RSA)
`client`	TLS Client	*1.3.6.1.5.5.7.3.2*	*DigitalSignature*
`code` [v4.3.0]	Code Signing	*1.3.6.1.5.5.7.3.3*	*DigitalSignature*

If the `ca` subsection is present, the user may issue intermediate CA certificates with the CA basic constraint extension set. It takes these properties:

Option	Type	Description	Mandatory
`pathlen`	integer	Certificate CA path length constraint	No

If the optional `pathlen` is defined, the path length is restricted to the given number of further sub-CAs. A `0` value indicates that the issued intermediate CA may not issue further sub-CAs. If the EST CA issuer certificate has itself a path length property set, the path length must conform to it. Path lengths can be requested in the certificate request CA basic constraint extension. If missing, the path length is determined by the given maximum `pathlen` setting and/or the issuer certificate path length property.

If the `require-approval` [v3.10.0] property is present and set to `true`, acceptable certificate requests require approval by a user having the property `require-approval` not set. Both the requesting and the approving user must have the permissions to issue the offending certificate.

If the `list` section is present, the user may list certificates in the `active` or `renewed` state. It takes the options below:

Option	Type	Description	Mandatory
`subject`	string	Subject *fnmatch* pattern RDNs to restrict listing certificates to (see `issue` section `subject`)	No

If the `revoke` section is present, the user may revoke certificates. It takes the following options:

Option	Type	Description	Mandatory
`subject`	string	Subject *fnmatch* pattern RDNs to restrict certificate revocation to (see `issue` section `subject`)	No

If the `audit` section is present, the user can operate on the CA audit log. It takes the following options:

Option	Type	Description	Mandatory
`view`	bool	Viewing the audit log is allowed	No

The `self-list` v3.12.0, `self-revoke` v3.12.0 and `self-renew` v3.12.0 options are valid for TLS client certificate authenticated users, only. They allow listing, revoking and renewing certificates associated to the certificate used for TLS client authentication. The `self-list` and `self-revoke` sections allow the user to list and revoke certificates with the same certificate subject Distinguished Name as present in the TLS client certificate. Both the `self-list` and `self-revoke` sections currently take no further options.

The `self-renew` section allows the user to renew certificates, without requiring potential subjectAltName challenges or certificate approval. A renewed certificate may contain only properties that already exist in the TLS client certificate.

Option	Type	Description	Mandatory
`lifetime`	reltime	Renewed certificate lifetime, in s	Yes

The `lifetime` option is required in the `self-renew` section, and defines the lifetime of the issued certificate, starting at the issuing time.

All uses of *fnmatch* support extended matches with `FNM_EXTMATCH`. Example `auth.yaml` configuration:

```
users:
  tester:
    password: $1$BGhg9N8U$ZjwIi57fidJ4aJdmwCbN51
    issue:
      lifetime: 1209600
      subject: "C=CH, O=Example Org, CN=*"
      subject-altname: "+(client.*.example.com|*@example.com)"
      blocks:
      - 172.16.0.0/16
      eku:
      - client
      ca:
        pathlen: 0
      require-approval: true
cert:
  "C=CH, O=Example Org, OU=Admin, CN=*":
    issuer-chain:
    - C=CH, O=Example Org, CN=Root
    - C=CH, O=Example Org, OU=Admin, CN=Admin IM
    issue:
      lifetime: 1209600
    list:
      subject: "C=CH, O=Example Org, CN=*"
    revoke:
      subject: "C=CH, O=Example Org, CN=*"
    audit:
      view: True
  "C=CH, O=Example Org, OU=Employee, CN=*":
    self-list: {}
    self-revoke: {}
    self-renew:
      lifetime: 1y
```

2.2.4 Certificate Authority Policy "`policy.yaml`"

The Certificate Authority policy configuration defines global settings that are enforced for a single CA, independent of the per-user permissions defined in `auth.yaml`.

Under the configuration root section, the following subsections can be defined:

Option	Type	Description	Mandatory
`issue`	object	Certificate issuing policy	No
`crl`	object	CRL issuing policy	No
`expire`	object	Certificate expiration notification	No
`static`	object	Static resources to serve over HTTP	No

Multiple policy configuration options use the type *scriptable*, allowing the definition of user scripts to execute. A *scriptable* can be a *string* to pass as argument to the system shell interpreter using `/bin/sh -c`. Alternatively, a *scriptable* can be of a *object* type[v3.10.0], taking the following options:

Option	Type	Description	Mandatory
`script`	string	Script to pass to interpreter	Yes
`interpreter`	string	Custom script interpreter	No

The `interpreter` option takes an executable with absolute path or found in the system search `PATH`. The executable receives the defined `script` as additional argument to interpret. If omitted, the `interpreter` defaults to `/bin/sh -c`.

Certificate issuing policy

The `issue` policy takes the following parameters:

Option	Type	Description	Mandatory
`hash`	string	Hash function to use in certificate signatures, one of: `sha1` / `sha256` / `sha384` / `sha512`	No
`keys`	object	Certificate public key restrictions	No
`approval`	scriptable	Certificate request approval script	No
`verdict`	scriptable	Certificate request verdict script	No
`challenge`	scriptable	subjectAltName challenge script	No

If the CA issuer uses an RSA key, the `hash` algorithm is used in signatures for both issued certificates and CRLs. It defaults to `sha256`. For CA issuers using Ed25519 private keys, the option must not be used, as that crypto system always uses the Identity Hash to create signatures.

If the `keys` section exists, it restricts the subject public keys that are allowed to include in certificates. It takes further subsections named `rsa` to allow RSA keys and `eddsa` to allow Ed25519 keys. The `size` integer option within `rsa` or `eddsa` section defines the minimum key length, in bits, that a certificates public key must at least have.

If an `approval` [v3.10.0] script is specified, it is executed to request approval for issuing a certificate by a user that has the `require-approval` authorization setting set. The script is intended to notify administrators about a new certificate request requiring approval. It receives the following environment variables:

Variable	Description
`ESTER_CSR`	Path of the PEM encoded certificate request
`ESTER_SUBJECT`	Stringified subject Distinguished Name
`ESTER_USER`	Username requesting the given certificate

If a `verdict` [v3.10.0] script is specified, it is executed when a verdict has been set for a certificate request requiring approval. It allows, for example, to notify administrators that it has been acted upon a approval request triggered by an `approval` script. The script receives the following environment variables:

Variable	Description
`ESTER_CSR`	Path of the PEM encoded certificate request
`ESTER_SUBJECT`	Stringified subject Distinguished Name
`ESTER_USER`	Username that approved/rejected the request
`ESTER_VERDICT`	Certificate request verdict: `accepted` or `rejected`

If a `challenge` [v3.10.0] script is specified, it is executed to challenge subjectAltNames requested for any certificate. The script receives a list of subjectAltNames and a list of challenge tokens. To allow issuing the requested certificate, the requesting user must include all given tokens, in any order, concatenated as PKCS#10 *challengePassword* in the certificate request to retry the enrolment. The script is expected to challenge all subjectAltNames, for example by sending the token associated to an RFC822 subjectAltname via e-mail to the user. The script output on `stdout` is included in the enrolment error message and may contain instructions how the user must proceed to retry the enrolment. The script receives the following environment variables:

Variable	Description
`ESTER_SUBJECT`	Stringified subject Distinguished Name
`ESTER_USER`	Username requesting the given certificate
`ESTER_ALTNAMES`	Space separated stringified subjectAltNames
`ESTER_TOKENS`	Space separated tokens for each subjectAltName

Challenge tokens are valid for the same day, for the same public key in the certificate request, for the exactly same subjectAltName and for the same user enrolling the certificate. The `challenge` script is executed only if the *challengePassword* is omitted from a certificate request.

CRL issuing policy

The `crl` policy, if present, enables periodic issuing of Certificate Revocation Lists. It takes the following parameters:

Option	Type	Description	Mandatory
`interval`	reltime	Interval for issuing new CRLs, in s	Yes
`lifetime`	reltime	CRL lifetime, in s	Yes
`deploy`	scriptable	CRL deployment script	No
`uris`	list[string]	CRL distribution points to include in certificates	No

The `interval` option defines the interval CRLs are issued in. The lifetime of the CRL (that is, the *nextUpdate* field relative to the *thisUpdate* field) of the CRL may be longer, and is specified using the `lifetime` parameter.

If the `deploy` option is specified, the script is invoked for any newly issued CRL to deploy the PEM encoded CRL file specified in the environment variable `ESTER_CRL`. Usually the CRL is served by an external (web) server, and the deployed CRL must be available to certificate validators by the specified `uris`, which are embedded into all issued certificates.

Certificate expiration notification

The `expire` [v3.9.0] policy, if configured, can issue certificate expiration warnings using a script. It takes the following parameters:

Option	Type	Description	Mandatory
`notify`	scriptable	Certificate expiration script	Yes
`warn-at`	list[reltime]	Time of warning before certificate expires	No

The `notify` option specifies a script to invoke whenever a certificate, which has not been either `renewed` or `revoked`, expires. The script receives the following environment variables:

Variable	Description
`ESTER_CERT`	Path of the affected PEM encoded certificate
`ESTER_SUBJECT`	Stringified subject Distinguished Name
`ESTER_SERIAL`	Raw hex encoded certificate serial
`ESTER_EXPIRES`	Expiration date in ISO8601 format
`ESTER_WARN`	Number of additional warnings that will follow

`ESTER_WARN` is `0` if the certificate has expired. If the `warn-at` list is defined, it takes a list of up to eight relative time specifications when additional certificate expiration warnings shall be issued using the `notify` script before a certificate finally expires.

Static HTTP resource serving

The `static` policy section defines directories with static files to serve over the builtin HTTPS server for this CA. This is useful to serve additional content over the same origin as the EST server itself, for example Javascript client code that interacts with the EST CA. It takes sections names for *URI path* prefixes, starting with a slash, each taking the following options:

Option	Type	Description	Mandatory
`path`	string	Local path to serve files from	Yes
`indexes`	list[string]	Index file names to consider for serving	No

The `path` option defines the local path to find static files to serve under the sections *URI prefix*. The path of served files must resolve to a path under the configured `path` when following symlinks. The `indexes` list defines candidate files in a directory to serve if a URL requests a static file resolving to that directory.

Example for a `policy.yaml` CA policy:

```
issue:
  hash: sha384
  keys:
    rsa:
      size: 2048
  approval: |
    echo "New CSR for $ESTER_SUBJECT." | \
      mail -s "Approval request by $ESTER_USER" ca@example.org
  verdict: |
    echo "CSR for $ESTER_SUBJECT $ESTER_VERDICT." | \
      mail -s "Request by $ESTER_USER handled" ca@example.org
  challenge:
    interpreter: bash -e -c
    script: |
      read -r -a T <<< "$ESTER_TOKENS"
      read -r -a N <<< "$ESTER_ALTNAMES"
      test "${#T[@]}" -eq 1
      test "${#N[@]}" -eq 1
      echo "Your token: ${T[0]}" | \
        mail -s "Challenge token for $ESTER_USER" ${N[0]}
      echo "Please include challenge token sent to ${N[0]}."
crl:
  interval: 86400
  lifetime: 259200
  uris:
  - http://www.example.com/crl.der
  - ftp://ftp.example.org/crl.der
  deploy: >-
    openssl crl -in $ESTER_CRL -outform der |
    ssh crl@www.example.com "cat > /var/www/crl.der"
expire:
  notify: |
    set -e
    if [ $ESTER_WARN = 0 ]
    then
      echo "$ESTER_SUBJECT $ESTER_SERIAL has expired." | \
        mail -s "Certificate expired!" ca@example.com
    else
      echo "$ESTER_SUBJECT $ESTER_SERIAL expires on $ESTER_EXPIRES." | \
        mail -s "Certificate expiring" ca@example.com
    fi
  warn-at:
  - 2w
```

```
  - 3d
static:
  /:
    path: /usr/share/est-webapp
    indexes:
    - index.html
```

3 EST Operations

The onway EST CA implements an EST endpoint compatible to RFC7030 and RFC8951. It does not support all operations defined in EST, but implements some additional proprietary operations:

Operation	Standard	Authentication	onway EST CA support
`/cacerts`	RFC7030	None	Supported
`/simpleenroll`	RFC7030	Both	Supported with extensions
`/simplereenroll`	RFC7030	TLS Client Cert	Supported
`/fullcmc`	RFC5273	Any	Not supported
`/serverkeygen`	RFC7030	Any	Not supported
`/csrattrs`	RFC7030	None	Supported
`/certs`	Proprietary	Both	Supported
`/simplerevoke`	Proprietary	Both	Supported
`/pending`	Proprietary	Both	Supported
`/approved`	Proprietary	Both	Supported
`/rejected`	Proprietary	Both	Supported
`/auditlog`	Proprietary	Both	Supported

The *both* authentication type means that clients may either use HTTP basic authentication with username and password or a TLS client certificate, depending on the authentication configuration.

In all responses to TLS client certificate authenticated requests, the EST server includes a `EST-Client-Subject` [v3.12.0] HTTP header with a string representation of the subject Distinguished Name of the client certificate. This allows a client to identify itself in a restricted environment, for example in a web browser.

Note

The onway EST CA always uses an EST *Explicit Trust Anchor* for TLS server certificate validation. The TLS server certificate is issued automatically under the CA issuer certificate, hence a TLS client must manually establish trust in the issuer certificate, for example using the `/cacerts` operation.

Note

The onway EST CA follows the clarification recommendations as outlined in RFC8951 for Base64 encoding of requests and responses. It ignores any `Content-Transfer-Encoding` header in requests and always assumes Base64 encoding. In responses, it omits the `Content-Transfer-Encoding` and implicitly uses Base64 encoding for any PKCS#7 response.

3.1 `/cacerts` Operation

The `/cacerts` operation allows unauthenticated clients to retrieve the CA certificate chain. It uses a `GET` request, and the response is a PKCS#7 container with the configured issuer CA

certificate, followed by any issuing Intermediate/Root CA provided in the CA configurations `trusted` directory. The *Root CA Key Update* mechanism is currently not supported.

Example to query CA certificates:

```
wget https://est.example.com/.well-known/est/cacerts -O - \
   --no-check-certificate --content-on-error | base64 -d | \
   openssl pkcs7 -inform der -print_certs
```

The above command will skip TLS server certificate validation and fetches the full CA certificate chain. This EST *Explicit Trust Anchor* is untrusted, and verifying the certificate is up to the user. To establish trust in the EST CA, select the root certificate and store it in a file called `root.pem` before executing any following example.

```
wget https://est.example.com/.well-known/est/cacerts -O - \
   --ca-certificate root.pem --content-on-error | base64 -d | \
   openssl pkcs7 -inform der -print_certs
```

The second version will reuse that trust anchor to verify TLS certificates. All the following commands will make use of that `root.pem` certificate to establish trust in the EST server.

3.2 `/simpleenroll` Operation

The `/simpleenroll` operation enrolls a certificate. It expects a HTTP `POST` request with a PKCS#10 certificate request, and on success responds with a PKCS#7 container including the issued certificate.

In the onway EST CA, `/simpleenroll` is always authenticated using HTTP basic auth or with TLS client certificates against the configured users. Deferred certificate approval with enrolment retries is supported. If a certificate request requires approval, a `/simpleenroll` request is accepted with a `202` HTTP status code, allowing the client to retry the enrolment after the timeout specified in the HTTP `Retry-After` response header.

The certificate request may contain the following X.509 extensions, which are, when validated successfully, applied to the issued X.509 certificate:

- subjectAltNames
- RFC3779 address blocks (containing subnets or ranges[v3.9.0])
- ExtendedKeyUsages
- CA basic constraints with optional path length

Note

Certificates having exactly the same subject, subjectAltName, RFC3779 address blocks, ExtendedKeyUsages and CA basic constraints as an existing valid certificate, replace that existing certificate. This means that such an existing certificate is moved to the *renewed* state. The certificate stays valid until it expires, but it will not trigger any expiration warning.

The type and strength of the public key may be restricted by the CA policy. Currently supported are RSA and Ed25519 public keys.

The lifetime is defined by the user authorization configuration, always starting at the issuing time. Shorter lifetimes can be requested by using proprietary HTTP headers in the `/simpleenroll` request:

- `EST-Not-Before`: Requested X.509 *notBefore* date
- `EST-Not-After`: Requested X.509 *notAfter* date

The `EST-Not-Before` date must not precede the issuing date, and the `EST-Not-After` date must not exceed the maximum lifetime, relative to the issuing date. Dates must be in the exact same format as a HTTP `Date` header, that is, as specified in RFC7231 7.1.1.1. under *Preferred format*.

The determined lifetime must be within the lifetime of the issuer certificate[v4.0.0]. This ensures that a certificate chain obtained during the enrolment process stays valid as a whole for the issued certificate.

The following example generates a new RSA private key and enrolls a certificate using a generated PKCS#10 request:

```
openssl genrsa > rsa.pem
wget https://est.example.com/.well-known/est/simpleenroll -O - \
   --ca-certificate root.pem --content-on-error \
   --http-user tester --http-password test \
   --header="EST-Not-After: $( \
      LC_TIME=C date -u --date=tomorrow \
         +'%a, %d %b %Y %H:%M:%S GMT' \
   )" \
   --header="Content-Type: application/pkcs10" \
   --post-data="$( \
      openssl req -new -key rsa.pem \
      -subj '/C=CH/O=Example Org/CN=Test' \
      -addext subjectAltName=DNS:client.test.example.com \
      -addext sbgp-ipAddrBlock=IPv4:172.16.1.0/24 \
      -addext extendedKeyUsage=clientAuth \
      -addext basicConstraints=critical,CA:TRUE,pathlen:0 \
      -outform der | base64 \
   )" | base64 -d | openssl pkcs7 -inform der -print_certs
```

The `date` command uses the syntax from the GNU `coreutils` package. For other systems, the syntax may differ.

3.3 `/simplereenroll` Operation

The `/simplereenroll` [v3.12.0] operation renews or rekeys an existing certificate. It requires the client to authenticate using a TLS client certificate subject for re-enrolment, and the user must have the `self-renew` authorization.

The operation expects a HTTP `POST` request with a PKCS#10 certificate request. The certificate request must contain only properties that already exist in the current user certificate presented for authentication (namely the certificate subject, any subjectAltNames and RFC3779 address blocks, any ExtendedKeyUsage and any CA basic constraint having an identical path length).

The `/simplereenroll` operation requires no subjectAltName challenging nor certificate approval, hence a response is never deferred using a `202` HTTP status code. Lifetime customization using the `EST-Not-Before` and `EST-Not-After` HTTP headers is currently not supported. Otherwise, the behaviour of the `/simplereenroll` operation is identical to the

`/simpleenroll` operation. On success, the response contains a PKCS#7 container including the issued certificate.

The following example authenticates with a TLS client certificate to renew it:

```
wget https://est.example.com/.well-known/est/simplereenroll -O - \
   --ca-certificate root.pem --content-on-error \
   --certificate cert.pem --private-key rsa.pem \
   --header="Content-Type: application/pkcs10" \
   --post-data="$( \
      openssl x509 -x509toreq -key rsa.pem -in cert.pem \
         -copy_extensions copy | openssl req -outform der | base64 \
   )" | base64 -d | openssl pkcs7 -inform der -print_certs
```

3.4 `/csrattrs` Operation

The `/csrattrs` EST operation provides information about attributes and X.509 extensions to expect or support in certificate requests. It does not require any authentication, and provides the information using the `Content-Type: application/csrattrs` returned for HTTP `GET` requests.

Currently no information about expected signature algorithms or supported public key systems is included. The response contains the list of X.509 extension types that can be requested by specifying them in certificate requests. As the request is unauthenticated, this does not reflect the user specific permissions for individual extensions.

3.5 `/certs` Operation

The `/certs` EST operation is a proprietary extension to the EST protocol implemented by the onway EST CA. It allows an authenticated user to list certificates in the *active* and *renewed* states using a HTTP `GET` request.

The request may include a `EST-Subject` HTTP header to filter certificates to return by the given subject Distinguished Name. The header value is a string representation of the full Distinguished Name, as defined in RFC4514. The user must have the `list` authorization to list certificates, which may further restrict the subjects that are allowed to be listed. Alternatively[v3.12.0], a TLS client certificate authenticated user having the `self-list` authorization may list all certificates having a subject Distinguished Name equal to that of the TLS client certificate.

The response is similar to the `/cacerts` operation, that is, a PKCS#7 container with a list of certificates. It uses a `Content-Type: application/pkcs7-mime; smime-type=certs-only`.

The following example produces a list of currently valid certificates:

```
wget https://est.example.com/.well-known/est/certs -O - \
   --ca-certificate root.pem --content-on-error \
   --http-user tester --http-password test \
   --header="EST-Subject: C=CH, O=Example Org, CN=Test" | \
   base64 -d | openssl pkcs7 -inform der -print_certs
```

3.6 `/simplerevoke` Operation

The `/simplerevoke` EST operation is a proprietary extension to the EST protocol implemented by the onway EST CA. It allows an authenticated user to revoke certificates. The user must have the `revoke` authorization, which may further restrict the subjects to allow certificate revocation for. Alternatively[v3.12.0], a TLS client certificate authenticated user having the `self-revoke` authorization may revoke any certificate having a subject Distinguished Name equal to that of the TLS client certificate.

A `/simplerevoke` operation expects a HTTP `PUT` request. One or more certificates to revoke are provided in their full encoding, wrapped into a PKCS#7 container, very similar to a `/cacerts` response. It uses the `Content-Type: application/pkcs7-mime`.

If multiple certificates are provided for revocation, and the revocation of one of the certificates fails, the revocation state of the other certificates is undefined. The `/certs` operation can be used to check if a certificate is still in the list of active certificates. Revoking already revoked or expired certificates is skipped silently to have idempotent semantics for the `PUT` request.

The response format is not explicitly defined, but usually includes a textual operation result.

Example to revoke a certificate in `cert.pem`:

```
wget https://est.example.com/.well-known/est/simplerevoke -O - \
   --ca-certificate root.pem --content-on-error \
   --http-user tester --http-password test \
   --header="Content-Type: application/pkcs7-mime" \
   --method PUT --body-data="$(
      openssl crl2pkcs7 -nocrl -certfile cert.pem -outform der | \
      base64 \
   )"
```

3.7 `/auditlog` Operation

The `/auditlog` EST operation is a proprietary extension to the EST protocol implemented by the onway EST CA. It allows an authenticated user to retrieve the audit logfile of the CA. The user must have the `audit view` authorization to view the log.

The `/auditlog` operation uses a HTTP `GET` request, and returns a plaintext logfile with a `Content-Type: text/plain`. The log is human readable, but does not provide a stable format. The audit logfile is always provided in full, and is identical to the `audit.log` file found in the CA directory.

3.8 `/pending` Operations

The `/pending`[v3.10.0] EST operations are a proprietary extension to the EST protocol. They allow an authenticated user to list certificate requests awaiting approval in the *pending* state using a HTTP `GET` request, or to delete such a certificate request using the HTTP `DELETE` method. A user can see or manage certificate requests that they are also allowed to approve, only.

The response for the HTTP `GET` operation contains zero or more concatenated PEM encoded PKCS#10 certificate requests with the `Content-Type: application/x-pem-file`. The

HTTP `DELETE` method expects a `Content-Type: application/pkcs10` request with a base64-encoded certificate request, similar to the `/simpleenroll` operation, and produces an unspecified response.

The following example fetches a list of *pending* certificate requests and stores the first to a file. Then it deletes that first certificate request:

```
wget https://est.example.com/.well-known/est/pending -O - \
   --ca-certificate root.pem --content-on-error \
   --http-user tester --http-password test | \
   openssl req > pending.pem
wget https://est.example.com/.well-known/est/pending -O - \
   --ca-certificate root.pem --content-on-error \
   --http-user tester --http-password test \
   --header="Content-Type: application/pkcs10" --method=DELETE \
   --body-data="$(openssl req -in pending.pem -outform der | base64)"
```

Deleting a *pending* certificate request is useful to clean up state if a user aborts enrolment before the request was approved or rejected. If a user tries to enrol the same request that has been deleted previously, it is again put into *pending* state. A request that has been *rejected*, though, is refused directly when an enrolment is retried.

3.9 `/approved` Operations

The `/approved` [v3.10.0] EST operations are a proprietary extension to the EST protocol. They allow an authenticated user to list certificate requests awaiting enrolment in the *approved* state using a HTTP `GET` request, to delete such a certificate request using the HTTP `DELETE` method, or to HTTP `PUT` a certificate request from the *pending* to the *approved* state. A user can see or manage certificate requests that they are also allowed to approve, only.

The response for the HTTP `GET` operation contains zero or more concatenated PEM encoded PKCS#10 certificate requests with the `Content-Type: application/x-pem-file`. Both the HTTP `PUT` and `DELETE` methods expect a `Content-Type: application/pkcs10` request with a base64-encoded certificate request, similar to the `/simpleenroll` operation, and produce an unspecified response.

The following example approves a *pending* certificate request:

```
wget https://est.example.com/.well-known/est/approved -O - \
   --ca-certificate root.pem --content-on-error \
   --http-user tester --http-password test \
   --header="Content-Type: application/pkcs10" --method=PUT \
   --body-data="$(openssl req -in pending.pem -outform der | base64)"
```

Putting a certificate request into the *approved* state does not immediately enrol the certificate. Instead, the user is required to retry the enrolment over `/simpleenroll` with the exactly same certificate request. Once the enrolment completes, the certificate request is implicitly deleted from the *approved* state. Deleting an *approved* certificate request explicitly is useful to clean up state if a user aborts enrolment after it has been approved.

3.10 `/rejected` Operations

The `/rejected` [v3.10.0] EST operations are a proprietary extension to the EST protocol. They allow an authenticated user to list certificate requests in the *rejected* state using a HTTP `GET` request, to delete such a certificate request using the HTTP `DELETE` method, or to HTTP `PUT` a certificate request from the *pending* to the *rejected* state. A user can see or manage certificate requests that they are also allowed to approve, only.

The response for the HTTP `GET` operation contains zero or more concatenated PEM encoded PKCS#10 certificate requests with the `Content-Type: application/x-pem-file`. Both the HTTP `PUT` and `DELETE` methods expect a `Content-Type: application/pkcs10` request with a base64-encoded certificate request, similar to the `/simpleenroll` operation, and produce an unspecified response.

The following example rejects a *pending* certificate request:

```
wget https://est.example.com/.well-known/est/rejected -O - \
   --ca-certificate root.pem --content-on-error \
   --http-user tester --http-password test --method=PUT \
   --body-data="$(openssl req -in pending.pem -outform der | base64)"
```

A certificate request in the *rejected* state is directly rejected if a user retries the enrolment for it. Deleting a certificate request can be useful to reconsider its enrolment traversing again through the *pending* state.

4 Contact Information

onway ag
Stauffacherstrasse 16
CH-8004 Zürich
Switzerland
Tel: +41 55 214 18 30
info@onway.ch
www.onway.ch