Confusa architecture documentation

Henrik Austad & Thomas Zangerl

September 23, 2009

Contents

1 Authentication in Confusa

Authentication in Confusa is mostly done by calling the simplesamlphp[1] subsystem, which will take care of AuthN and deAuthN by performing the login-redirects and consuming the SAML assertions that are sent by the IdPs.

Most information about the currently logged on user is administered in the Person class, which is just a convenient data store for all information that is received from the IdPs.

Figure 1 shows the framework of authentication classes in Confusa. The state in which we want to end is having a fully "decorated" person, i.e. a person that has all attributes that we need to know about her assigned to her member variables. Assuming that we know nothing about the person and a sensitive action in Confusa, which requires such knowledge, is selected, Confusa performs the following steps:

1 The AuthHandler picks the right ConfusaAuth-subclass for the current situation. Often that is `ConfusaAuth_IdP`, which handles authN using a WebSSO SAML2 redirect in simplesamlphp to authN the user.

2 authenticateUser() in the ConfusaAuth-subclass authenticates the user and makes sure to get her attributes. In `ConfusaAuth_IdP`, the user is redirected to the initSSO page of simplesamlphp which will take care of the rest.

3 When Confusa is then opened again, because simplesamlphp relayed the user back to the site that initiated the request, the attributes are assigned to the person using the decoratePerson() method in `Confusa_Auth`.

4 Different IdPs may send different attribute names for the same attributes. For instance, the name of the user may be send as "cn" `=>` "John Doe" or "displayName" `=>` "John Doe" or "name" `=>` "John Doe". Therefore, we must keep a mapping of the different attribute keys of the NRENs and subscribers to the attribute keys we need. That mapping is defined in the admin menu and kept in the database. Upon decoration of the person the mapping is looked up to assign the right attribute values to the right attributes. For that, getMap() in AuthHandler is called. See figure 1 for schematics of the process.

5 With the right attribute keys, the attribute values can be retrieved and ConfusaAuth can decorate the person, which is now authenticated and generously providing user information to other components of Confusa.

deAuthenticating (logging out) the user is even simpler.

1 Confusa forgets all attributes of the user by unsetting the fields in Person.

2 For IdPs not supporting SingleLogout (SLO), Confusa calls a simplesamlphp logout procedure that calls registered logout hooks, i.e. attributes that were sent by the IdPs for logging out.

[1]http://rnd.feide.no/simplesamlphp

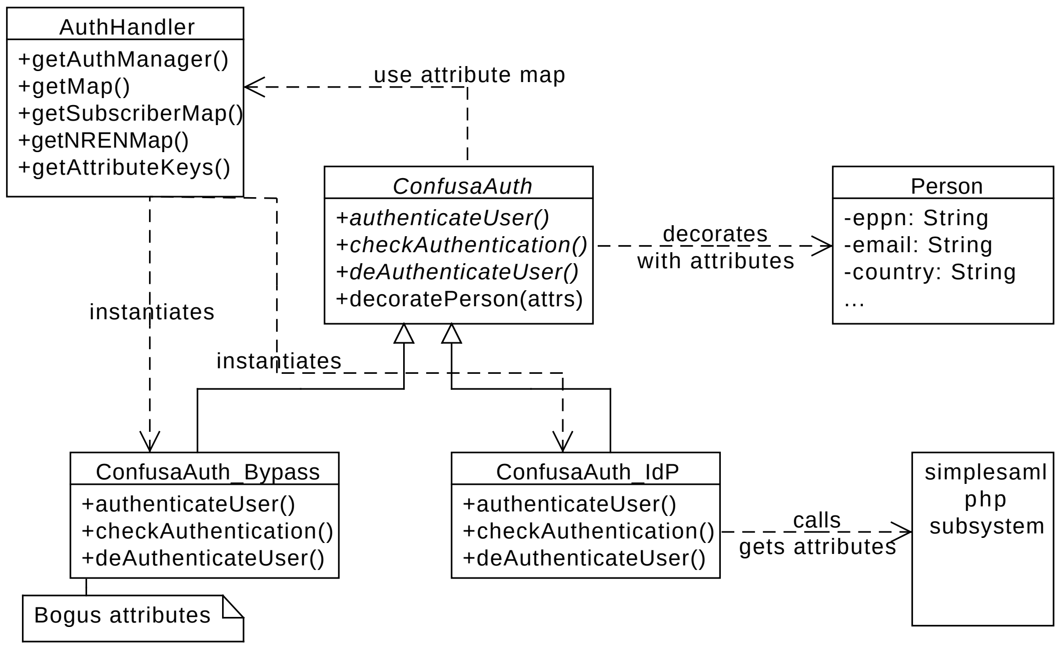


Figure 1: The Confusa Auth classes

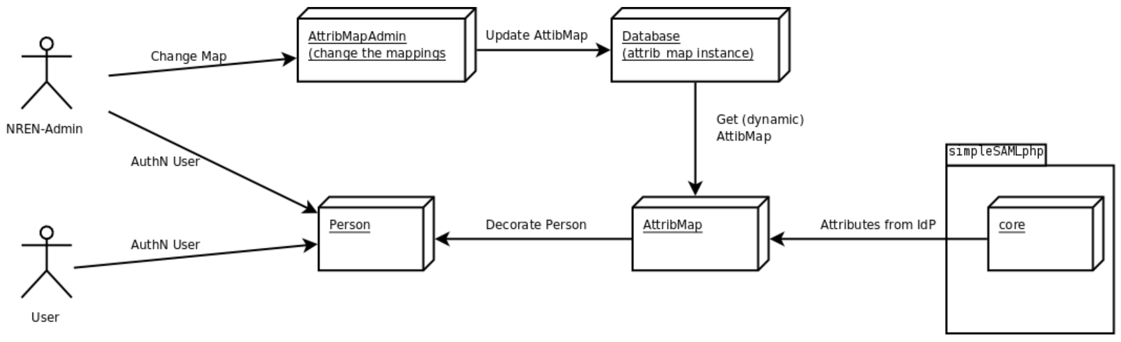


Figure 2: Confusa attribute mapping process

3 For those IdPs that support (SLO), Confusa just redirects to simplesamlphp's initSLO page. Otherwise, Confusa has to find a logout address in the attributes sent by the IdP

2 Certificate request signing in Confusa

2.1 Confusa modes

Confusa can sign certificates in two modes

- Using its own certificate and key (*standalone*)
- Using the HTTP POST API of Comodo (*online*)

Depending on the mode the behaviour is slightly different. For standalone-mode, an `auth_key`, which is a pubkey-hash, is the main identifier of the certificate, while in online-mode, an `order-number` serves as Confusa's ID for the certificate.

2.2 Certificate signing in online-mode

The flow of certificate signing in online mode is shown in figure 2.1. The following steps are followed by Confusa:

1 The user uploads a CSR using her browser

2 An `auth_key` is computed, which is the first 16 characters of the pubkey-hash.

3 `Process_CSR` passes the CSR to `csr_lib.php` to check for well-formedness and known-problems (is the pubkey already known? is the RSA algorithm used? does the key-length match the configuration? was the vulnerable Debian openssl software used for generating the CSR?).

4 If everything is okay in steps 1 to 3, then Confusa inserts the CSR into the DB where it will be stored for some time (default: 10 minutes).

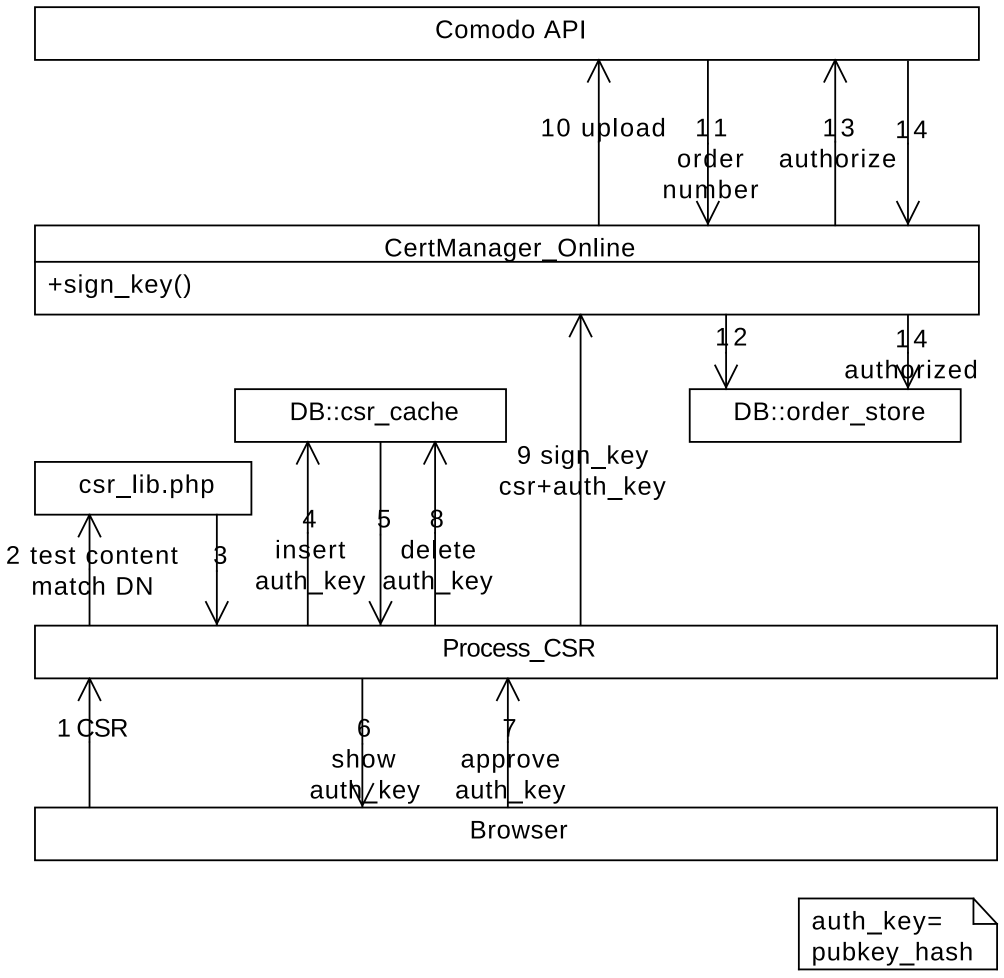
5 Within that timeframe, the user has the chance to approve the CSR for signing. If the user approves the request, it will be deleted from the DB and passed to `CertManager_Online`

6 `CertManager_Online` takes the request and uploads it using a HTTP POST request to the Comodo API. The Comodo API returns an order-number.

7 `CertManager_Online` inserts the `auth_key`, order-number and other certificate releated information into the `order_store` table in the database

8 In the next step, the Comodo-API is contacted to authorize the signing request.

9 If the request was successfully authorized, this is reflected in the DB. Now, the certficate is processed by Comodo and will appear in the download-list later on.


Comodo API
10 upload
11 order number
13 authorize
14
CertManager_Online
+sign_key()
12
14 authorized
DB::csr_cache
DB::order_store
csr_lib.php
9 sign_key csr+auth_key
2 test content match DN
3
4 insert auth_key
5
8 delete auth_key
Process_CSR
1 CSR
6 show auth_key
7 approve auth_key
Browser
auth_key= pubkey_hash

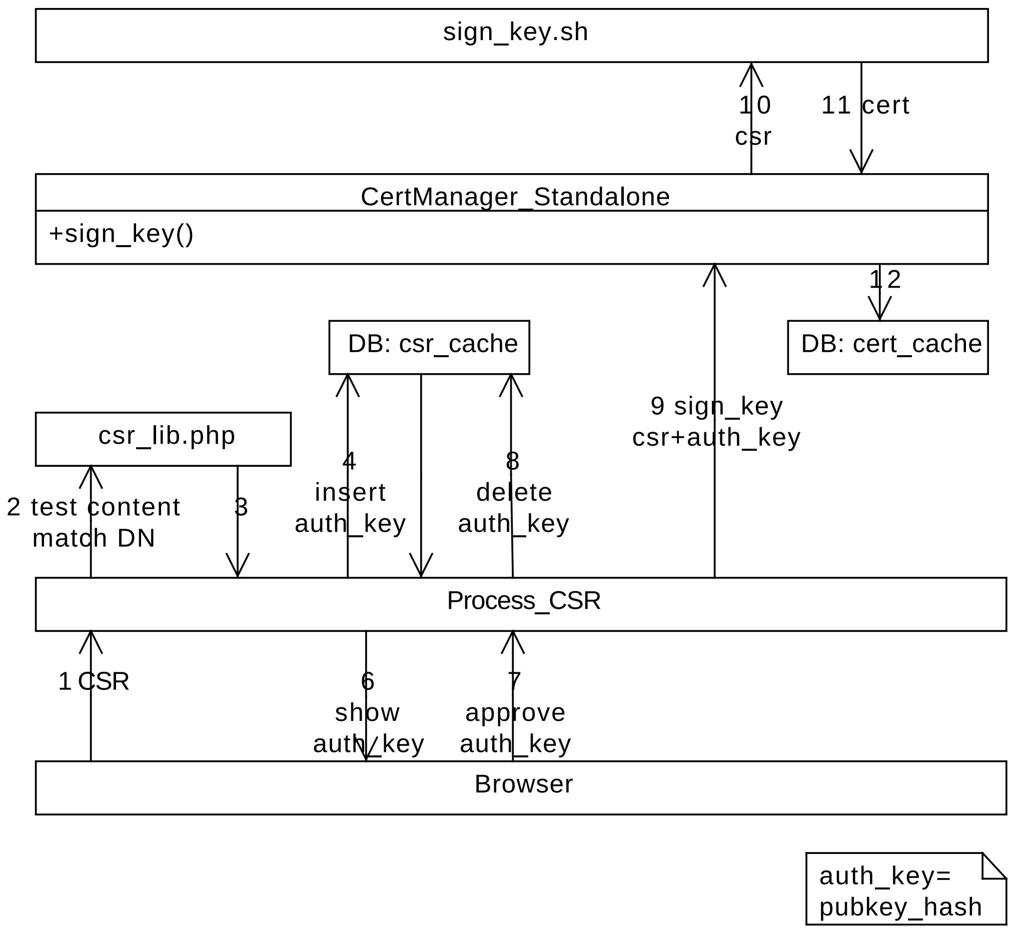


Figure 3: CSR signing flow in Confusa's standalone configuration

2.3 Certificate signing in standalone-mode

Figure 2.1 shows the standalone-signing mode, which is quite similar to the online-signing mode. For standalone-signing to work, CA-certificates must be present in the `cert_handle/` directory of Confusa and certain files such as ca.db.serial must be writeable by the webserver-user. Certificate signing in standalone works as follows (steps 1 and 2 are the same as in online mode):

3 Is the same as in online mode, but additionally it is checked if the subject-DN of the CSR matches the attributes that were received from the identity provider. It is only thus that Confusa can ensure that the user only issues certificates to herself.

4 If everything is okay in steps 1 to 3, then Confusa inserts the CSR into the DB where it will be stored for some time (default: 10 minutes).

5 Within that timeframe, the user has the chance to approve the CSR for signing. If the user approves the request, it will be deleted from the DB and passed to `CertManager_Standalone`

6 `CertManager_Standalone` passes the CSR to a shell script within the `cert_handle/` directory of Confusa, which will call openssl and thus sign the CSR with the installed CA certificates. The signed certificate is returned to `CertManager_Standalone`.

7 `CertManager_Standalone` inserts the certificate into the `cert_cache` table in the DB, where it can be picked up for download later on.

2.4 Certificate signing for browser requests

Note: Browser requests currently work only in online mode.

For browser requests, certificate signing works a bit different. First, browsers are a quite inhomogenous kind: they don't have an uniform crypto API, nor do they all support the semi-official `<keygen>` tag. Different versions of IE generate certificate requests in `PKCS#10` format, Firefox's crypto-API in CMRF-format and the `<keygen>` tag outputs SPKAC (Signed Public Key And Challenge), a Netscape format that is, however, supported by OpenSSL.

We use

- XEnroll.dll for Internet Explorer on Windows XP and Server 2003
- CertEnroll.dll for Internet Explorer on Windows Vista, Server 2008 and Windows 7
- `<keygen>` for all other browsers

Initially we used Firefox's Crypto-API for Firefox, but because of the bad support of the CRMF-standard that was dropped in favour of universal use of the `<keygen>` tag. JavaScript is needed for all browsers, even though `<keygen>` could work without, to keep the programmatic part as simple as possible and - more important - to have the same user flow for all browsers. And if some crypto API will appear for other browsers as well, it will be easier to hook into that.

Figures 2.4 and 2.4 illustrate processing and deployment of certificates in the browser. When processing a browser request,

1 The browser generates a CSR, in `PKCS#10` or SPKAC and a private key. The private key goes into the browser's keystore. The CSR is submitted in a POST message to `Process_CSR`.

2 `Process_CSR` calls signBrowserCSR, along with the user-agent that made the request in `CertManager_Online`.

3 The request is sent to Comodo using the HTTP POST API, a order number is returned.

4 The order-number is passed through all components down to the JavaScript, which will periodically poll for the status of the order using HTTP GET messages.

5 Whenever `Process_CSR` is polled for the status, it will itself query the backend, which will query the Comodo API for the status.

6 Once processing was successful, a "done" status is pushed down to the JavaScript, which will stop polling and instead use DOM manipulations to show an install-link on the page.

If the user clicks the install-link on the page, or in the download-section, the following happens:

1 A HTTP GET message is sent to `Process_CSR`.

2 `Process_CSR` queries `CertManager_Online` for a certificate deployment script for the `order_number`, passing the user agent as a second argument.

3 `CertManager_Online` queries the Comodo API for the certificate in the right format, that means dependant on the browser either the full-chain or only the certificate itself and either a certificate or executable JavaScript code for certificate installation.

4 Once it has the certificate, it either returns it, or wraps it in `<script>` tags so it can be executed.

5 The certificate in the right format gets passed down to `Process_CSR`. Dependant on the user agent, it will be passed to `download_certificate` in `file_download.php` or passed down to the JavaScript for execution.

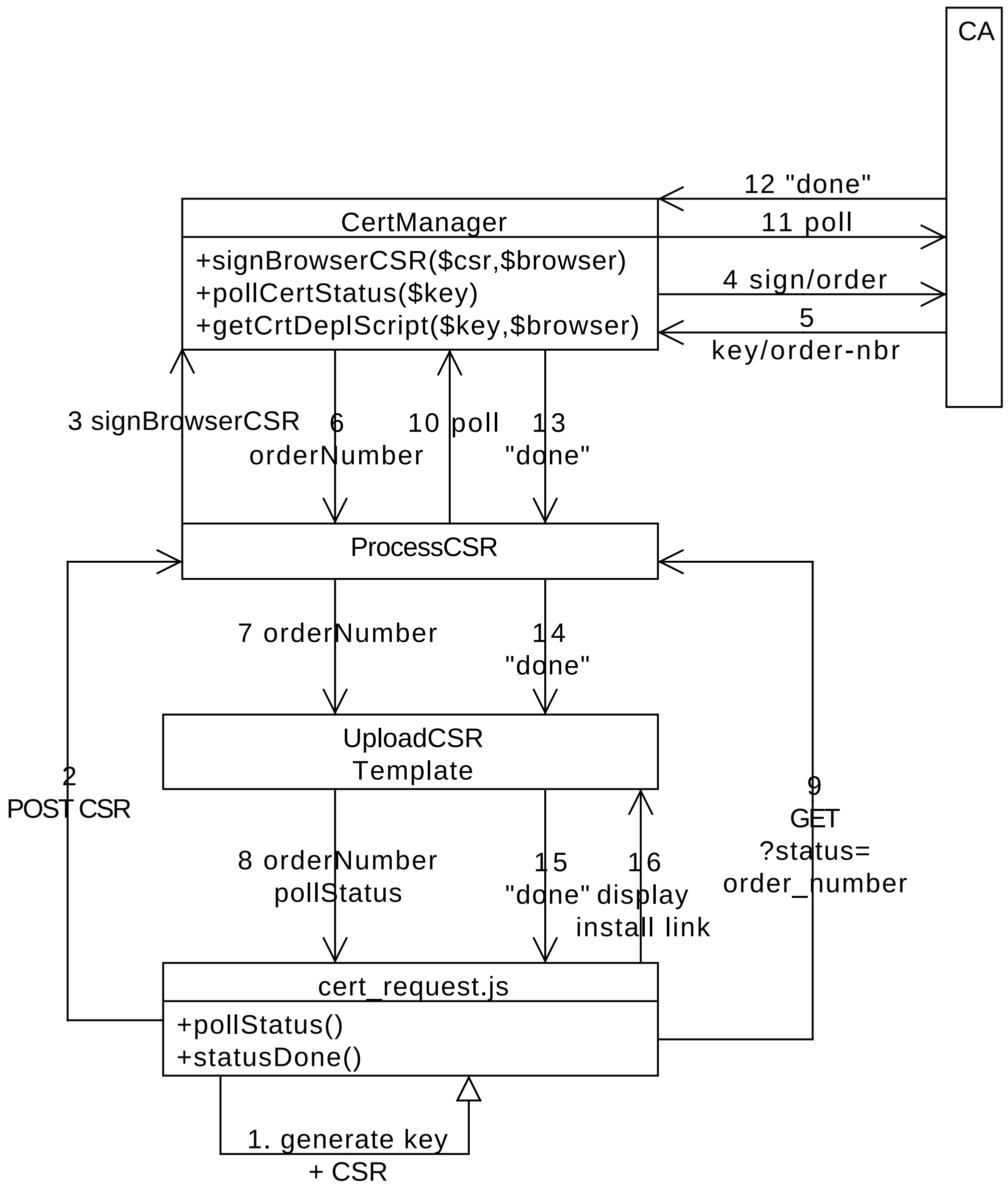


Figure 4: Processing of browser certificate requests

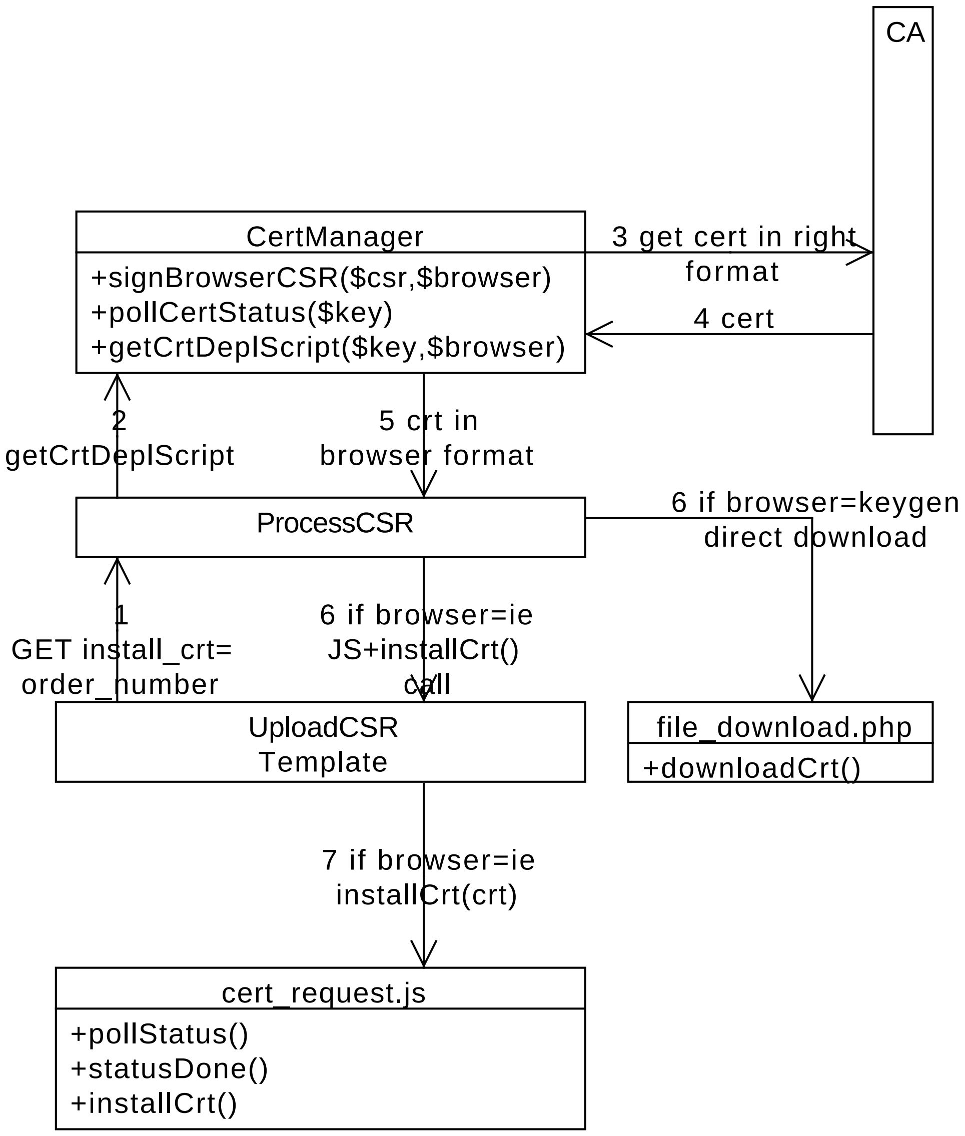

CA
CertManager
+signBrowserCSR($csr,$browser)
+pollCertStatus($key)
+getCrtDeplScript($key,$browser)
3 get cert in right format
4 cert
2 getCrtDeplScript
5 crt in browser format
ProcessCSR
6 if browser=keygen direct download
1 GET install_crt= order_number
6 if browser=ie JS+installCrt() call
UploadCSR Template
file_download.php
+downloadCrt()
7 if browser=ie installCrt(crt)
cert_request.js
+pollStatus()
+statusDone()
+installCrt()

Figure 5: Deployment of certificates in browsers

3 The Framework

All visible Confusa pages should be embedded in the `Framework` class, subclassing `ContentPage`. The communication between the `Framework` and the `ContentPage` is detailed in figure 3.

The framework is roughly responsible for the following things:

- Ensure a decorated person object is in place
- Initiate user authentication if the content-page is protected (restricted to authN users)
- Call `pre_process`, process and `post_process` on the ContentPage
- Render fixed component of Confusa itself (NREN-logo, menu, include NREN-CSS)
- Display of notification messages

The ContentPage is the superclass for all pages of Confusa showing actual content. The actual content is stored in a Smarty[2] template. The ContentPage is called by Framework in three phases:

- `pre_process`: Things to be done before the content of the page (template) is shown. E.g. processing of POST parameters, translation of strings on the page.
- `process`: Things to be done when the content of the page is shown, e.g. assigning template variables.
- `post_process`: Things to be done after the page has been displayed. Cleanup...

[2] http://www.smarty.net/

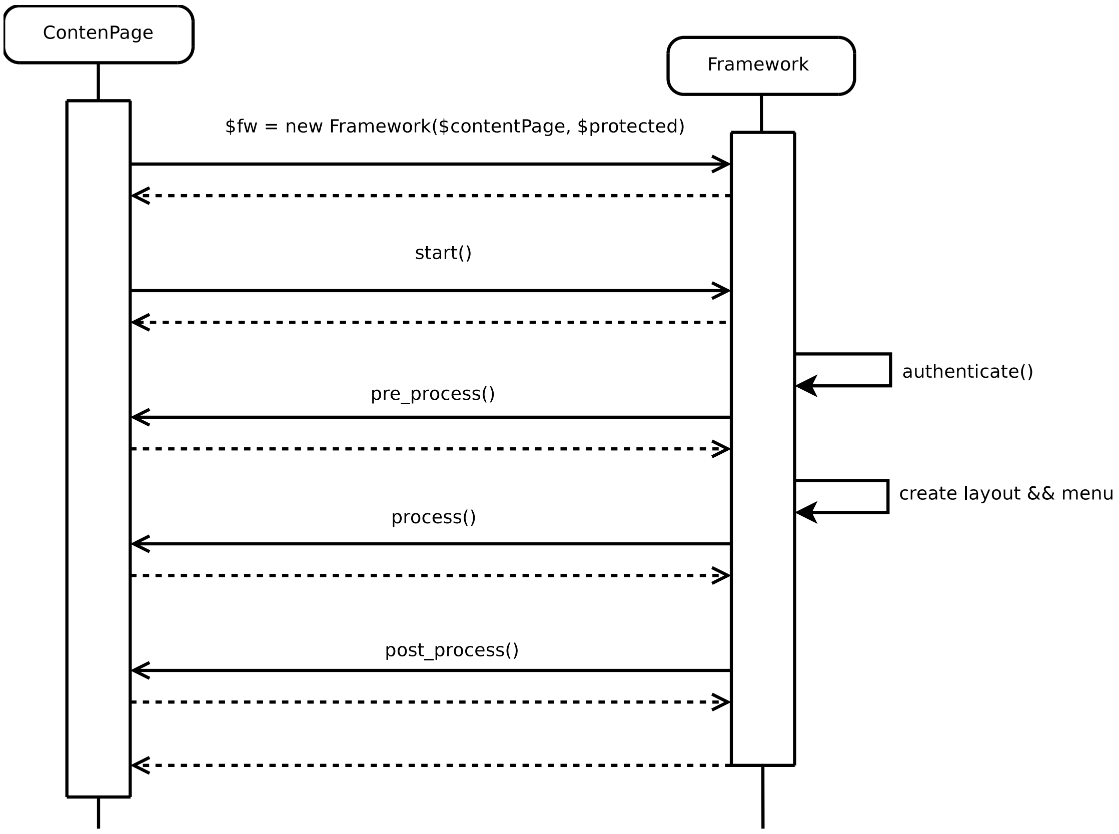


Figure 6: Sequence diagram of the interaction between the Framework and the ContentPage