BFH (Bern University of Applied Sciences), CH-2501 Biel, Switzerland

UniCert Architecture Specification

Version 0.2

Philémon von Bergen

December 2, 2014

Revision History

Revision	Date	Author(s)	Description
0.1	September 10, 2014	Philémon von Bergen	Initial draft.
0.2	December 2, 2014	Philémon von Bergen	Review after user interface suppression

Contents

1 Introduction **4**

2 Components and Processes **5**
- 2.1 Components . . . 5
- 2.2 Authentication Process . . . 5
- 2.3 Certificate Issuance Process . . . 6

3 Architecture and Interface **8**
- 3.1 unicert-authentication . . . 8
  - 3.1.1 Interface . . . 8
  - 3.1.2 UserData and IdentityFunction . . . 9
  - 3.1.3 Error description . . . 12
- 3.2 unicert-issuer . . . 13
  - 3.2.1 CryptographicSetup . . . 13
  - 3.2.2 Certificate class . . . 14

4 UniCert Particularities **18**
- 4.1 X.509 Certificate Extensions . . . 18
- 4.2 JNDI Propreties for *unicert-issuer* . . . 18

1 Introduction

This document presents the architectural specification of UniCert. UniCert is a certification authority, that issues digital certificates used to authenticate users, to sign and/or encrypt messages. UniCert provides an interface where the user can authenticate themselves and request predefined parameters that must appear in the certificate they will ask UniCert to issue. UniCert uses UniBoard to publish all issued certificates. A general description of what UniCert does and how it works is available in document [1].

This document presents the detailed certificate issuing process and focuses on architectural and technical specifications. In a Chapter 2, the components and processes of authentication and issuance of the certificate will be described in more details. Chapter 3 describes the architecture of some important classes. Finally, Chapter 4 describes some important UniCert particularities.

2 Components and Processes

To give an overview of how UniCert works, we give a high-level picture of the architecture and the processes that compose UniCert.

2.1 Components

UniCert is composed of two components. The first one, the authentication component called *unicert-authentication*, has following functions:

- authenticate the user
- received certificate request data and send them to issuer component
- return the issued certificate (received from the issuer component) to the user.

The application using UniCert is responsible for generating the key pair and choosing other values to certify. UniCert provides only a test user interface for these tasks.

The second component, the issuer component called *unicert-issuer*, has following functions:

- issue the certificate based on the data received
- post the certificate on UniBoard
- return the certificate to the authentication component

2.2 Authentication Process

Currently, the authentication component supports two identity providers to authenticate the user, namely SwitchAAI and Google. Even they do not use the same technologies, the flow of the authentication process is similar. Their concrete implementation differ however slightly. The authentication process shows in Figure 2.1 is described below.

To begin the procedure, the user asks the application using UniCert to authenticate themselves. The application calls the *AuthenticationServlet* of *unicert-authentication* providing the identifier of the identity provider, the return url that will by called by *unicert-authentication* at the end of the authentication process to go back to the application and optionally an additional parameter. *AuthenticationServlet* saves the return url and the additional parameter in the session and makes a redirect on selected identity provider's authentication page where the user can process to the login. If the login process is successful, the identity provider returns user attributes to *SwitchAAICallbackServlet* (in case of SwitchAAI) or *OAuth2CallbackServlet* (in case of Google). Theses servlets save the user attributes in the session and call the *CallbackServlet*. This servlet call the return url saved in the session, though some information back to the application (session id, e-mail of user,

unique id of user, and the additional parameter passed when requesting the authentication page).

At the end of the authentication process, the application has an authenticated session on *unicert-authentication* for this user.

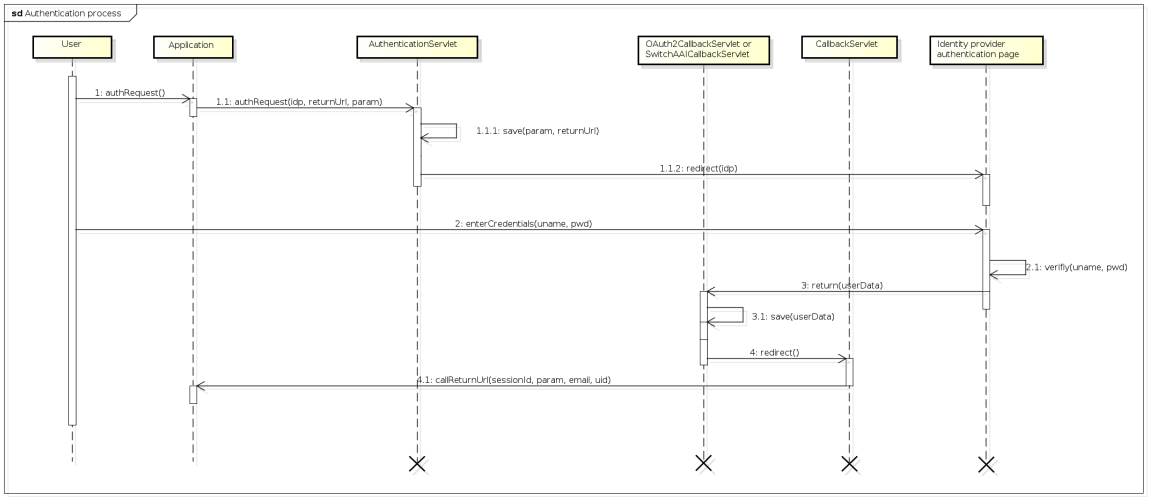

Figure 2.1: Authentication process

2.3 Certificate Issuance Process

The rest of the process is issuance of the certificate. This works in the same way independently of the identity provider used. The process is shown in Figure 2.2. The application using UniCert must provide following information to request a certificate:

- type of cryptographic values (currently supported: RSA and discrete logarithm)
- cryptographic parameters: size of key for RSA, values p, q, g for discrete logarithm
- a private key (generated locally and not sent to the server)
- a public key corresponding to the private key
- a password to encrypt the private key (not send to the server)
- application identifier of the application which the certificate is issued for
- role in the application the certificate is issued for
- the identity function that must be applied on the identity information
- a cryptographic proof or signature demonstrating knowledge of the private key

The generation of the private key is responsibility of the application or the user. The public key corresponding to the generated private key must be computed together with a signature (or a proof in case of discrete logarithm setup) to prove the knowledge of the private key. The data listed above (except the private key and the password) must then be sent

to the *CertificateRequestServlet*. In this servlet, the user's personal information (also called *user data*) that were received during the authentication process (see Section 2.2) are used in combination with the provided identity function to generate the *identity data* that will be included in the certificate. This process is realized by the *unicert-authentication* component. The data received and the identity data are then transmitted to the *unicert-issuer* component. This second component verifies the data received and issues the certificate as requested, publishes it on UniBoardand returns it to the *unicert-authentication* component which sends it back to the application.

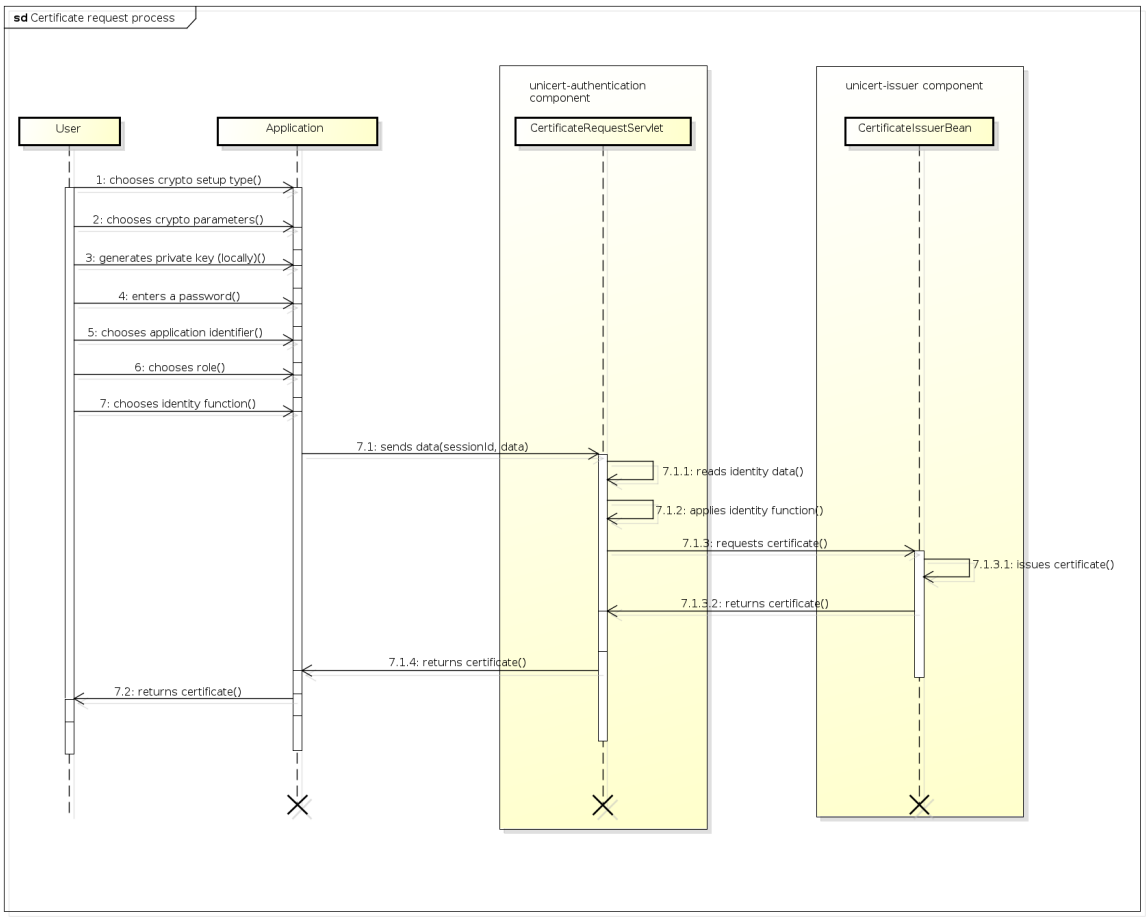

Figure 2.2: Process of requesting a certificate

3 Architecture and Interface

This chapter describes the interface of *unicert-authentication* and provide explanations on some classes and the architecture concepts of both *unicert-authentication* and *unicert-issuer* components.

3.1 unicert-authentication

unicert-authentication is a web application, but all the tasks are done in the Java part in servlets, namely the *AuthenticationServlet*, *CertificateRequestServlet* and *CallbackServlet*. Some important concepts related to the data that must be sent to UniCert to request a certificate will be described here.

3.1.1 Interface

This subsection describes the data that must be sent to *unicert-authentication* for the authentication phase and the certificate request phase.

Authentication process For the authentication request, information displayed in Table 3.1 must be sent as HTTP GET parameters. Table 3.2 shows the information returned in the url after successful authentication.

Key	Format	Description
idp	"switchaai"/"google"	Key of the identity provider to use for authentication
returnurl	URL	Url to call after successful authentication
params	String (optional)	Additional parameter that must be past when calling return url

Table 3.1: Information to send with authentication request

Key	Format	Description
JSESSIONID	String	Id of the session on the server
params	String	Parameter sent with of authentication request
mail	E-mail	Email address of user returned by the identity provider
id	String	Unique identifier of user returned by the identity provider
idp	"SwitchAAI" or "Google"	The identity provider used for authentication

Table 3.2: Information sent after successful authentication

Certificate request For the certificate request, some information must be provided. For a RSA certificate, information displayed in Table 3.3 must be sent, for a discrete logarithm certificate Table 3.4 shows them. This time, these parameters must be sent as HTTP POST parameters.

Key	Format	Description
crypto_setup_type	"DiscreteLog"/"RSA"	Type of certificate to issue
crypto_setup_size	String representing an integer	Size of the public key (1024 or 2048)
public_key	String	Base 10 representation of the public key to certify
rsa_modulo	String	Base 10 representation of RSA modulo
application_identifier	String	Identifier of the application in which the certificate will be used
role	String	Roles in the application the certificate will be used for. If multiple roles must be defined, they must be comma separated.
identity_function	String representing an integer	Index of identity function to use during certification process
signature	String	Base 10 representation of signature proving knowledge of the private key

Table 3.3: Information to send with certificate request for RSA certificates

Each identity function is identified by a number. Table 3.5 shows which id correspond to which function.

The message of the signature that must be submitted with the RSA certificate request is in fact the concatenation of the other parameter sent. Besides proving the knowledge of the private key corresponding to the public key sent, this signature allows the server to verify that the parameters received have not been modified by someone during the request. The same feature is obtained in the discrete logarithm case by pushing the same message in the hash function used during challenge generation. This messages is structured as follows (separator between values is a pipe "|"):

idp/user_email/user_unique_id/crypto_setup_type/crypto_setup_size/public_key/rsa_modulo/identity_function/application_identifier/role

for RSA and

idp/user_email/user_unique_id/crypto_setup_type/crypto_setup_size/public_key/dlog_p/dlog_q/dlog_generator/identity_function/application_identifier/role

for discrete logarithm.

3.1.2 UserData and IdentityFunction

The format of the user data returned by the identity provider is not the same for all identity providers. The way these data are transmitted to UniCert is also different for various identity providers. These facts require a flexible solution to provide support of multiple identity providers and to allow to easily add a new identity provider.

SwitchAAI use a Shibboleth Apache module for the authentication. This technology requires a user to be logged in before displaying a web page or accessing a servlet. In this

Key	Format	Description
crypto_setup_type	"DiscreteLog"/"RSA"	Type of certificate to issue
crypto_setup_size	String representing an integer	Size of the public key (1024 or 2048)
public_key	String	Base 10 representation of the public key to certify
dlog_p	String	Base 10 representation of prime value P
dlog_q	String	Base 10 representation of prime value Q
dlog_generator	String	Base 10 representation of generator G
application_identifier	String	Identifier of the application in which the certificate will be used
role	String	Roles in the application the certificate will be used for. If multiple roles must be defined, they must be comma separated.
identity_function	String representing an integer	Index of identity function to use during certification process
dlog_proof_commitment	String	Base 10 representation of commitment of proof of knowledge of private key
dlog_proof_challenge	String	Base 10 representation of challenge of proof of knowledge of private key
dlog_proof_response	String	Base 10 representation of response proof of knowledge of private key

Table 3.4: Information to send with certificate request for discrete logarithm certificates

Id	Identity function
1	Standard SwitchAAI identity function
2	Anonymized SwitchAAI identity function
3	SwitchAAI identity function for University of Zürich
4	Standard Google identity function
5	Anonymized Google identity function

Table 3.5: Index of identity functions

concrete case, it is the page *AuthenticationServlet*. If the user is not logged in when calling the address of this servlet, a redirection takes place to SwitchAAI login page. After a successful login, the servlet *SwitchAAICallbackServlet* is called, where the user's identity are read. These user data are then stored in the session in an object of the class *SwitchAAIUserData*.

For Google, the user data are returned to a servlet called *OAuth2CallbackServlet* after successful login. This servlet stores the user data in a object of the class *GoogleUserData* in the session. Both *SwitchAAIUserData* and *GoogleUserData* inherits from the *UserData* interface but their content is not the same as shown in Figure 3.1, .

The *certificate-issuer* component, however, must know the content and structure of the UserData object in order to be able to extract the data needed in the certificate. Therefore, an identity function is used. There is at least one identity function per identity provider that knows the format of the corresponding user data. So, the GoogleIdentityFunction class knows the fields present in a GoogleUserData object, and the SwitchAAIIdentityFunction

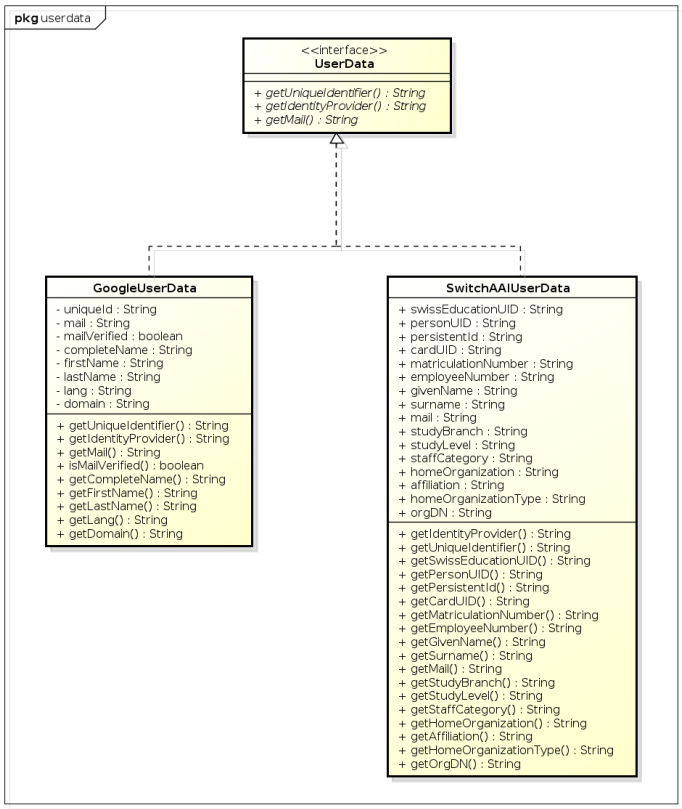

Figure 3.1: Class diagram of user data classes

knows the content of the SwitchAAIUserData. These IdentityFunction objects allows to transform a concrete UserData object into a generic class called *IdentityData* containing the identity information needed for the certificate. This IdentityData object can then be used by the *certificate-issuer* component.

The way how the identity information are extracted from a concrete UserData object is defined in a concrete IdentityFunction object. This process must be adaptable: for example an application could require the e-mail address to appear in certificate, another application could require an other unique identifier, or a third application could require an anonymized form of the e-mail address, and so on. Therefore, multiple IdentityFunction can exist for the same identity provider. In this implementation, there is a standard and an anonymized identity function for each identity provider (*StandardSwitchAAIIdentityFunction*, *StandardGoogleIdentityFunction*, *AnonymizedSwitchAAIIdentityFunction*, *AnonymizedGoogleIdentityFunction*). For SwitchAAI, there is a third variant specially designed for the university of Zürich which uses the student identification number as common name (*ZurichSwitchAAIIdentityFunction*).

The organisation of these classes is shown in Figure 3.2. The *apply* method is the publicly available method extracting the information out of the given UserData object and returning the generated IdentityData object shown in Figure 3.3. Different protected methods are used for this process. *selectCommonName* is responsible for extracting the common name from the UserData. In standard functions, the e-mail address is used. In anonymized function, a hash of the e-mail address is used. In the special function for Zürich, the

student identification number is used. *selectUniqueId* is responsible for extracting a unique identifier from the UserData. *putInOtherValues* allows to add information in extensions of the certificate in case it cannot be included in the fields proposed in the IdentityData class. Finally, method *createIdentityData* indicate which fields are put in the IdentityData object. This architecture allows to create subclasses which can overwrite some behavior of the super class. For example, method *createIdentityData* in anonymized function does not include "firstname" and "surname" in the IdentityData object as this is done in standard functions.

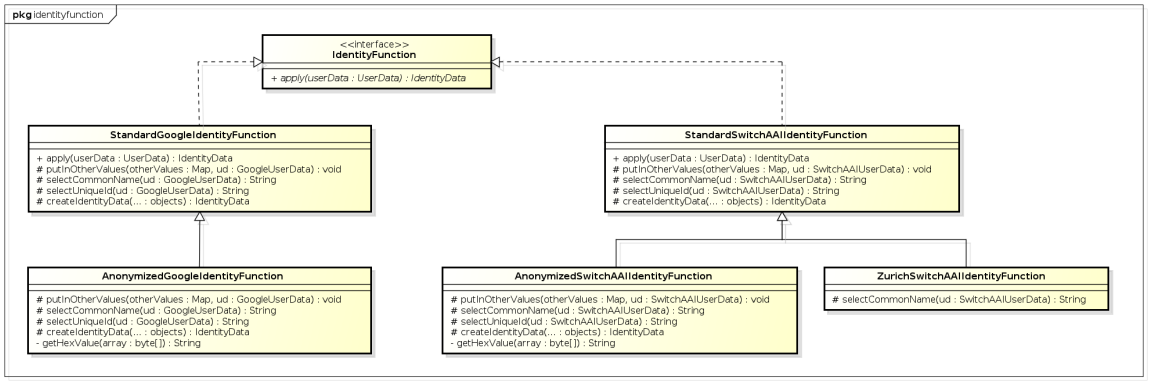

Figure 3.2: Class diagram of the identity function classes

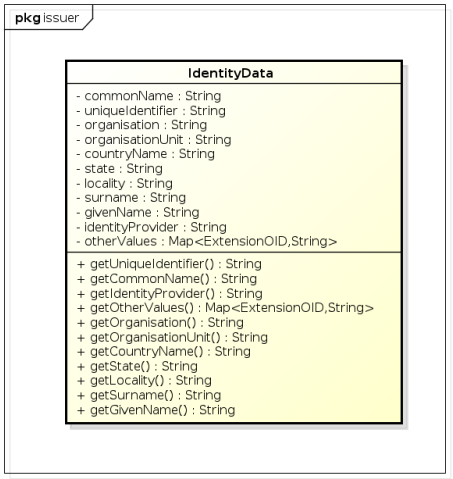

Figure 3.3: IdentityData class

3.1.3 Error description

Errors occurring during the issuance process should be communicated in an understandable way to the user. Therefore, error codes have been introduced. When an error occurs, an HTTP 500 response is sent with a JSON content containing an error code and a default error description. The error codes are listed in Table 3.6 and Table 3.7.

Code	Name	Description
100	Not authenticated	User data not found in session, user is not authenticated in this session
101	Missing argument	Some parameter is missing when requesting certificate
102	Unknown IDP	The identity provider selected is not supported/known by UniCert
111	Server error	Error on server side (EJB not injected correctly)
120	Unknown identity function	The identity function selected in not supported
121	Important user data missing	An identity information required in the certificate is missing in the user data
122	Problem while anonymizing	An error occurred during the anonymization process
123	Incompatible identity function	The selected identity function is not applicable on the user data provided by the identity provider used
124	Unique identifier missing	Important identity data missing to initialize unique id
125	Unverified email address	Email provided by identity provider was not verified
130	Cryptographic error	An error occurred during a process using cryptography
131	Unknown key type	The selected key type is not known by UniCert

Table 3.6: Errors originating from *unicert-authentication*

Code	Name	Description
200	Server error	Generic error during certificate issuance
211	Unknown role	Selected role is not known/supported
221	Illegal crypto setting	Error in the cryptographic setting
222	Invalid signature	RSA signature is invalid
223	Invalid proof	Zero knowledge proof of knowledge is invalid
224	Incompatible crypto values	Some cryptographic values are incompatible
230	UniBoard error	Error during publication on UniBoard

Table 3.7: Errors originating from *unicert-issuer*

3.2 unicert-issuer

unicert-issuer is an EJB component responsible for the issuance of the certificate. The main class is the *CertificateIssuer* EJB which checks the data received and issues the certificate. Some other classes used in this process will be described in this section.

3.2.1 CryptographicSetup

The class *CryptographicSetup* is a container for all cryptographic values that are submitted by the user and that must be included in the certificate or verified in the *unicert-issuer* component. Two concrete CryptographicSetup exist: *DiscreteLogSetup* and *RSASetup* as shown in Figure 3.4. They both include all values that define such a setup and the signature or proof generated in the frontend. Objects of these classes are created by the *unicert-authentication* component on certificate request.

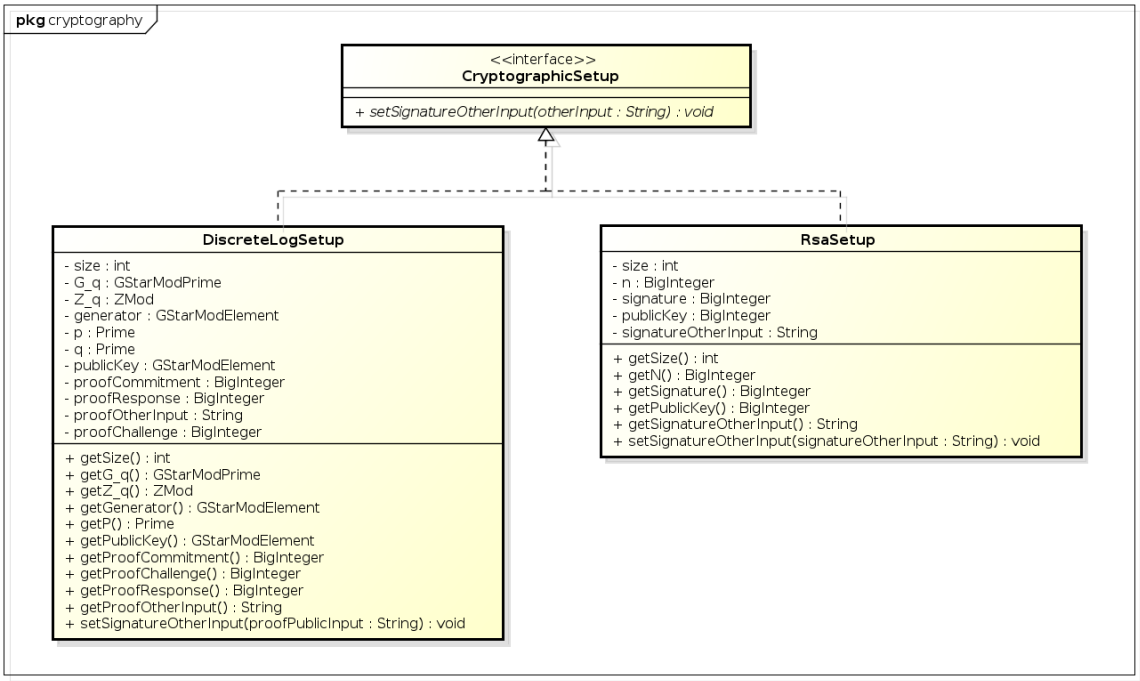

Figure 3.4: Class diagram of the cryptographic setup classes

3.2.2 Certificate class

Certificate class is the Java representation of the X.509 certificate that is issued. The fields that appear in that certificate are stored here together with the PEM format of the issued certificate as shown in Figure 3.5. When posting the certificate on UniBoard, a JSON representation of this class is created through the method *toJSON*. This allows UniBoard to make query on the content of a certificate. The listing below shows the schema of the generated JSON certificate.

```
{
        "title": "Schema for UniCert certificates",
        "description": "This schema describes the format of a
           UniCert certificate in JSON format",
        "type":"object",
        "$schema": "http://json-schema.org/draft-04/schema",
        "properties": {
                "commonName": {
                        "type": "string",
                        "description": "Common name of
                           certificate owner"
                },
                "uniqueIdentifier": {
                        "type": "string",
                        "description":  "Unique identifier of
                           certificate owner"
                },
```

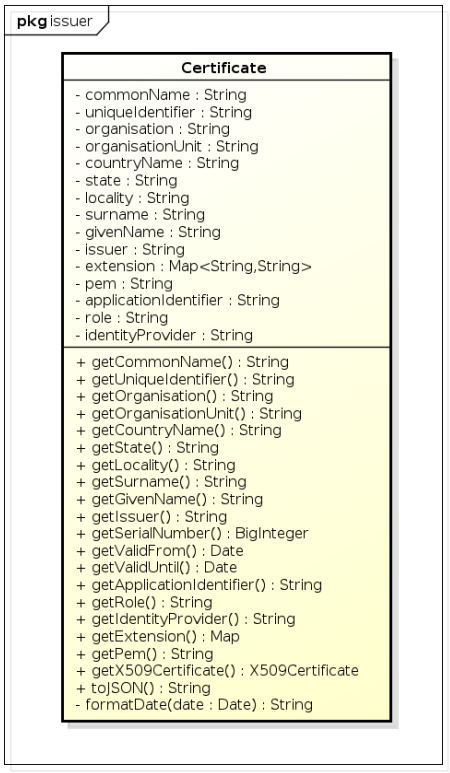

Figure 3.5: Certificate class

```
15                  "organisation": {
16                          "type": "string",
17                          "description": "Organisation of
                               certificate owner"
18                  },
19                  "organisationUnit": {
20                          "type": "string",
21                          "description": "Organisation unit of
                               certificate owner"
22                  },
23                  "countryName": {
24                          "type": "string",
25                          "description": "Country of certificate
                               owner"
26                  },
27                  "state": {
28                          "type": "string",
29                          "description": "State of certificate
                               owner"
30                  },
31                  "locality": {
32                          "type": "string",
```

```
                        "description": "Locality certificate
                            owner"
                },
                "surname": {
                        "type": "string",
                        "description": "Surname of certificate
                            owner"
                },
                "givenName": {
                        "type": "string",
                        "description": "Given name of
                            certificate owner"
                },
                "issuer": {
                        "type": "string",
                        "description": "Issuer of the
                            certificate"
                },
                "serialNumber": {
                        "type":"string",
                        "description": "Serial number of the
                            certificate"
                },
                "validFrom": {
                        "type": "string",
                        "description": "Date when certificate
                            starts to be valid"
                },
                "validUntil": {
                        "type": "string",
                        "description": "Date when certificate
                            stops to be valid"
                },
                "applicationIdentifier": {
                        "type": "string",
                        "description": "Application the
                            certificate has been issued for"
                },
                "role": {
                        "type":"array",
                        "description": "Role inside an
                            application the certificate has been
                             issued for",
                        "items": {
                                "type":"string"
                        }
                },
                "identityProvider": {
```

```
                        "type": "string",
                        "description": "Identity provider used
                            to verify the identity of the
                            certificate owner"
                },
                "pem": {
                        "type": "string",
                        "description": "Certificate in PEM
                            format"
                }
        },
        "required": ["commonName", "issuer", "serialNumber", "
            validFrom", "validUntil", "identityProvider", "pem"
            ]
}
```

4 UniCert Particularities

This chapter describes in more details some particularities of UniCert.

4.1 X.509 Certificate Extensions

Some special information must be added by UniCert in the issued certificate, like the identity provider used for the authentication, the application and the role the certificate is issued for. This information cannot really be included in existing X.509 fields. So, we defined some extensions for each of these elements. In X.509 certificates, an extension must always be identified by an object identifier (OID). The Bern University of Applied Sciences own a group OID: 1.3.6.1.4.1.13305. Following subgroup is reserved for UniCert 1.3.6.1.4.1.13305.1.101.x. Table 4.1 presents the elements included in the certificate and their corresponding OIDs. The content of these extensions are always ASN1 strings.

Element	OID
Identity provider	1.3.6.1.4.1.13305.1.101.1
Application identifier	1.3.6.1.4.1.13305.1.101.2
Role	1.3.6.1.4.1.13305.1.101.3
Language (only for Google)	1.3.6.1.4.1.13305.1.101.4
Test extension	1.3.6.1.4.1.13305.1.101.999

Table 4.1: OIDs

4.2 JNDI Propreties for *unicert-issuer*

To work properly, *unicert-issuer* require some configuration values. These values are defined as JNDI properties. Table 4.2 shows what are these values. They are stored in a property set called *unicertProps*.

Property	Description
keystoreExternal	If the keystore must be loaded from an external location
keystorePath	Path to the keystore containing the certificate and private key of UniCert for signing issued certificates
keystorePass	Password for the keystore
privateKeyPass	Password used to protect private key of UniCert certificate
issuerId	Name of the issuer of the generated certificates and id of the key in the keystore
validityYears	Time slot in which the certificate is valid (2 years)
uniboardURL	URL of UniBoard webservice (if not indicated, certificates are not published)
uniboardSection	Section where to publish on UniBoard
boardId	Id of the public key of the board in the keystore
googleRedirectURI	URL of the callback servlet after Google OAuth2 authentication
googleClientID	OAuth2 credentials for Google authentication
googleClientSecret	OAuth2 credentials for Google authentication

Table 4.2: JNDI properties for *unicert-issuer*

Bibliography

[1] R. Haenni, R. Koenig, S. Hauser, P. von Bergen, P. Locher, S. Fischli . UniVote System Specification. Technical report, 2014.