OpenXPKI. Publishing of certificates to LDAP database.

(Dated: Version 3rd May 2007)

A reading for OpenXPKI developers who are familiar with details of LDAP server operation

First Step. Reproducing of the OpenCA functionality

OpenXPKI, 2007

Draft document

CONTENTS

ABBREVIATIONS

LDAP: Lightweight Directory Access Protocol.

DN: Distinguished Name.

RDN: Relative Distinguished name.

L: Locality.

S: State.

O: Organization.

OU: Organizational Unit.

C: Country.

CN: Common name.

SN: Surname.

S: State.

UID: Unique identifier.

DC: Domain component.

1. INTRODUCTION

This document describes the current implementation of publishing issued certificates to LDAP database in OpenXPKI (working version was committed in 828 svn trunk).

The purposes of the document:

1. To outline problems
2. To stimulate a discussion
3. To develop a plan of adding new functionality and development framework components or changing the current implementation

The OpenCA Guide and the text of OpenCA LDAP.pm module were used as a base while writing OpenXPKI workflow code for certificate publishing to LDAP database.

That is why this document has a lot of references to the OpenCA Guide.

2. LDAP SERVER CONFIGURATION

Before you start working with OpenXPKI's LDAP code please be sure that your LDAP server knows the following objectclasses:

- pkiUser
- pkiCA

Those classes include attributes to store certificates and revocation lists. If you are using OpenLDAP v.2.3.34_1 the simplest way is to use the following directives in slapd.conf (directories are FreeBSD specific in this example):

```
include /usr/local/etc/openldap/schema/core.schema
include /usr/local/etc/openldap/schema/cosine.schema
include /usr/local/etc/openldap/schema/inetorgperson.schema
include /usr/local/etc/openldap/schema/openca.schema
```

Core, **Cosine** and **Inetorgperson** schemas are included with the openldap package. As for "openca.schema" it is borrowed from the OpenCA and modified to exclude object classes and attributes already defined in previously included schema files.

At the moment OpenXPKI uses the following set of objects defined in the openca.schema:

```
attributetype ( 1.2.840.113549.1.9.2 NAME 'unstructuredName'
        EQUALITY caseIgnoreMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256} )
attributetype ( 1.2.840.113549.1.9.8 NAME 'unstructuredAddress'
        EQUALITY caseIgnoreMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256} )
objectclass ( 1.3.6.1.4.1.18227.2.1.1 NAME 'opencaUniquelyIdentifiedUser'
        SUP top AUXILIARY MAY ( serialNumber ))
```

```
objectclass ( 1.3.6.1.4.1.18227.2.1.2 NAME 'opencaEmailAddress'
        SUP top AUXILIARY MAY ( mail $ emailAddress ))
objectclass ( 1.3.6.1.4.1.18227.2.1.3 NAME 'opencaSCEPDevice'
        SUP top AUXILIARY MAY ( unstructuredName $ unstructuredAddress))
```

At the moment OpenXPKI does not support TLS and SASL in LDAP-publishing workflow (only simple bind using rootdn and password). The LDAP server which is supposed to store issued by OpenXPKI certificates must be configured to permit connection without TLS and SASL.

3. LDAP-PUBLISHING OPTIONS IN OPENXPKI SERVER CONFIGURATION

LDAP-publishing options in OpenXPKI server configuration are stored in **ldappublic.xml** which is included to the realm section of the **config.xml**.

Options marked "not used" does not affect the current version of LDAP-workflow

<ldap_suffix>: LDAP database suffix which will be used to store certificates

<ldap_server>: LDAP server hostname

<ldap_port>: LDAP server port

<ldap_version>: LDAP version (not used, assumed to be 3)

<ldap_tls>: Use TLS if set to "yes" (not used)

<ldap_sasl>: Use SASL framework if set to "yes" (not used)

<ldap_chain>: Path to the certificates to be used in TLS session (not used)

<ldap_login>: The bind DN of the user which OpenXPKI uses to speak to LDAP server.

<ldap_password>: User's password

Example of using those options:

```
<ldap\_suffix>dc=OpenXPKI,dc=org</ldap\_suffix>
<ldap\_server>panther3.ipmce.ru</ldap\_server>
```

```
<ldap\_port>389</ldap\_port>
<ldap\_version>3</ldap\_version>
<ldap\_tls>no</ldap\_tls>
<ldap\_sasl>no</ldap\_sasl>
<ldap\_chain>chain</ldap\_chain>
<ldap\_login>cn=Manager,dc=OpenXPKI,dc=org</ldap\_login>
<ldap\_password>secret</ldap\_password>
```

The following options are also stored in ldappublic.xml and used to control the workflow:

<ldap_enable>: Activates LDAP-publishing workflow if set to "yes". Workflow is spawned just after the certificate issued is stored in SQL database.

<ldap_excluded_roles>: Comma-separated list of certificate roles which will not be published (say "CA Operator, RA Operator").

Examples of using those options to enable publishing of all the certificates to the LDAP:

```
<ldap\_enable>yes</ldap\_enable>
<ldap\_excluded\_roles>publish\_all\_roles</ldap\_excluded\_roles>
```

4. LDAP ATTRIBUTES AND OBJECT CLASSES SUPPORTED IN OPENXPKI WORKFLOW

LDAP attributes and object classes supported in OpenXPKI are listed in tables 1, 2 and 3. **"Supported"** here means that those attributes are added while creating new nodes in LDAP-tree during LDAP-publishing of the certificate issued by OpenXPKI.

The word "schema" in this section is used to specify the part of **ldappublic.xml** OpenXPKI configuration file where all the attributes and classes are enumerated and grouped according to structures shown in tables 1, 2 and 3. Schema-files (included in slapd.conf for OpenLDAP) are used for the other purpose - they just describes classes and attributes which LDAP server accepts.

The first table (**default** part of the schema) is used to create intermediate nodes of the DN. The second table (**certificate** part of the schema) is used to create nodes which are supposed to keep certificates. The third table (**ca** part of the schema - not used at the moment) is used to create nodes which are supposed to keep ca certificates (and maybe CRL ?).

The first column contains the attributes which may be included in DN. The second column shows attributes which **must** be included if the attribute from the first column is detected in some RDN. Attributes listed in the third column **may** be included in such a case. The last column shows object classes which OpenXPKI will add to the node being created.

Attribute values are added in three ways:

1. They extracted from the certificate DN
2. They extracted from the certificate Alternative Names
3. They substituted artificially according to some predefined rules

Examples of using those methods:

1. The certificate DN is **CN=John+SN=Smith,DC=OpenXPKI,DC=org**. While adding the node with this DN the workflow activity detects the attribute name CN which requires the attribute SN to be added too. Activity extracts attribute value **"Smith"** from the DN.

2. The certificate DN is **CN=John+SN=Smith,DC=OpenXPKI,DC=org**. While adding the node with this DN the workflow activity detects the attribute name CN which permits **mail** to be added too. Activity extracts attribute value **"smith@openxpki.org"** from the proper certificate alternative name.

3. The certificate DN is **CN=John Smith,DC=OpenXPKI,DC=org**. While adding the node with this DN the workflow activity detects the attribute name CN which requires the attribute SN to be added too. Activity adds attribute value **"NOT SUBSTITUTED YET"**. In the same situation OpenCA LDAP-module splitted CN to two fields "John" and "Smith" and substituted the second one as SN attribute value. It seems reasonable to plan supporting this substitution (and maybe some others like extracting **CN** from **mail**) in OpenXPKI too.

The configuration described in tables 1, 2 and 3 is borrowed from OpenCA **ldap.xml.template** and is an object to discuss and change in the way that better matches OpenXPKI roadmap. At the moment it is used in LDAP-publishing worflow without changes.

An example of adding the node **"ou=Security,o=ipmce,dc=openxpki,dc=org"** in the workflow adding a new certificate with DN **"cn=John,ou=Security,o=ipmce,dc=openxpki,dc=org"** is shown in Fig.2. Perl module AddMissingNode.pm parses the DN, detects OU attribute in the third RDN and then acts like this:

1. looks for the **OU** attribute in **default** part of the realm schema;

2. extracts the proper list of **Must** and **May** attributes from the schema;

3. extracts the values of the attributes from the certificate and adds the node using Net::LDAP->add method with prepaired set of parameters, including names of object classes also extracted from realm schema.

Table 1. Attributes supported by schema "default"

Attribute in RDN	**MUST attributes**	**MAY attributes**	**Object Classes** (top - always)
serialNumber	serialNumber cn	ou o l	device pkiCA
cn	cn	ou st l mail emailAddress	organizationalRole opencaEmailAddress pkiCA
sn	cn	ou st l mail emailAddress	organizationalRole opencaEmailAddress pkiCA
emailAddress	cn	ou st l mail emailAddress	organizationalRole opencaEmailAddress pkiCA
mail	cn	ou st l mail emailAddress	organizationalRole opencaEmailAddress pkiCA
uid	cn sn	uid mail emailAddress ou o st l	opencaEmailAddress pkiUser
dc	dc		dcObject pkiUser pkiCA
unstructuredName	cn	unstructuredName unstructuredAddress serialNumber ou o l	device opencaSCEPDevice pkiUser

Table 1. (continued)

Attribute in RDN	**MUST attributes**	**MAY attributes**	**Object Classes** (top - always)
unstructuredAddress	cn	unstructuredName unstructuredAddress serialNumber ou o l	device opencaSCEPDevice pkiUser
ou	ou	l st	organizationalUnit pkiUser pkiCA
o	o	l st	organization pkiUser pkiCA
c	c		country pkiUser pkiCA
l	l	l	locality pkiUser pkiCA
st	st	st	locality pkiUser pkiCA

Table 2. Attributes supported by schema "certificate"

Attribute in RDN	**MUST attributes**	**MAY attributes**	**Object Classes** (top - always)
serialNumber	cn sn	serialNumber mail emailAddress ou o st l	opencaEmailAddress opencaUniquelyIdentifiedUser pkiUser
cn	cn sn	mail emailAddress ou o st l	opencaEmailAddress pkiUser
sn	cn sn	mail emailAddress ou o st l	opencaEmailAddress pkiUser
emailAddress	cn sn	mail emailAddress ou o st l	opencaEmailAddress pkiUser
mail	cn sn	mail emailAddress ou o st l	opencaEmailAddress pkiUser
uid	cn sn	uid mail emailAddress ou o st l	opencaEmailAddress pkiUser
dc	dc		dcObject pkiUser
unstructuredName	cn	unstructuredName unstructuredAddress serialNumber ou o l	device opencaSCEPDevice pkiUser

Table 2. (continued)

Attribute in RDN	**MUST attributes**	**MAY attributes**	**Object Classes** (top - always)
unstructuredAddress	cn	unstructuredName unstructuredAddress serialNumber ou o l	device opencaSCEPDevice pkiUser
ou	ou	l st	organizationalUnit pkiUser
o	o	l st	organization pkiUser
c	c		country pkiUser
l	l	l	locality pkiUser
st	st	st	locality pkiUser

Table 3. Attributes supported by schema "ca"

Attribute in RDN	**MUST attributes**	**MAY attributes**	**Object Classes** (top - always)
serialNumber	cn	serialNumber ou o l	device pkiCA
cn	cn	ou st l	organizationalRole pkiCA
sn	cn	ou st l	organizationalRole pkiCA

Table 3. (continued)

Attribute in RDN	**MUST attributes**	**MAY attributes**	**Object Classes** (top - always)
emailAddress	cn	ou st l mail emailAddress	organizationalRole opencaEmailAddress pkiCA
mail	cn	ou st l mail emailAddress	organizationalRole opencaEmailAddress pkiCA
dc	dc		dcObject pkiCA
ou	ou	l st	organizationalUnit pkiCA
o	o	l st	organization pkiCA
c	c		country pkiCA
l	l	l	locality pkiCA
st	st	st	locality pkiCA

5. HOW IT WORKS

The workflow definition for LDAP publishing is shown in Fig. 1.

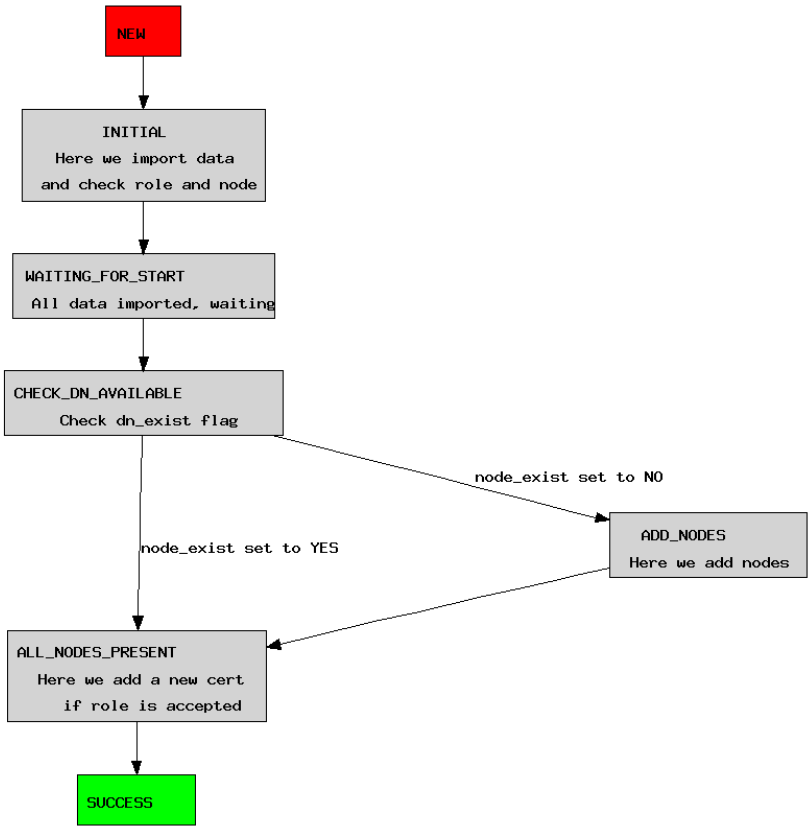

Figure 1. Workflow definition for LDAP publishing in OpenXPKI.

The procedure of adding LDAP node is shown in Fig. 2.

Figs. 3 and 4 describes the procedure of publishing a certificate in OpenXPKI workflow

6. FUNCTIONALITY AND FEATURES WHICH SEEM TO BE IMPORTANT BUT NOT ADDED YET

Let us point out some LDAP-workflow problems which deserve to be discussed. The interesting thing is priority in implementation schedule.

6.1. Architecture

Is it worth to separate utilities interacting with LDAP server (like **bind/unbind**, **start_tls**) to perl module LDAP.pm as it was done in OpenCA and initialize it at the stage of starting the OpenXPKI server? In that case all workflow modules dealing with LDAP will not spend time and memory to create new Net::LDAP instances.

6.2. Development framework

1. Tests for LDAP-publishing
2. If LDAP-workflow activity code is not quite suitable for tests, change it (split the massive chunk of code to functions etc.)
3. Invent a better way to wait for child workflows coming to SUCCESS state
4. HOWTO and FAQ for using LDAP in OpenXPKI;
5. Configuring LDAP at deployment stage;

6.3. Compatibility

Tests for LDAP protocol version 2.0 (shall we support it?)

6.4. Functionality

1. Multi-suffix support
2. Using TLS to communicate with LDAP server
3. Using SASL to connect to LDAP server

4. Adding mail attribute values from the certificate to the existing node
5. Check if adding the node failed
6. Substitute SN (say "Gut") from the second field of CN (say "Alles Gut") if SN is missing
7. Substitute CN (say "Jan") from the mail prefix (say "jan@openxpki.org") if CN is missing

6.5. Consistency

1. Check if OpenXPKI LDAP schema matches LDAP-server schema
2. Check if OpenXPKI LDAP schema specified in ldappublic.xml matches DN templates in profile.xml
3. Check if OpenXPKI LDAP schema specified in ldappublic.xml permits to process all cases of DN built with profile.xml templates
4. Check if OpenXPKI LDAP schema specified in ldappublic.xml does not cause prepairing more than one structural stack of objectclasses for the node to be created

7. BYPASSED OBSTACLES WHICH PROBABLY SHOULD NOT BE BYPASSED

While working on LDAP-publishing code some strange things were detected in OpenXPKI. We could not find reasons for them but made some guesses and used workaround. May be somebody knows the true cause of troubles we had. Here they are:

1. Contrary to actions each condition method is called 3 times while doing a workflow step (as far as stderr.log is truthful). That is why all "Condition" modules in LDAP workflow just use context parameters created by the previous actions.

2. Sometimes checks of mutually exclusive conditions triggers exceptions. It looks like the value of condition is altered between two checks and both exclusive conditions appear to be valid (or not valid) simultaneously. So workflow factory cannot decide which action must be performed. Happens often during "waiting_for_child". This and previous obstacle are the reasons why at the moment the parent workflows are configured in such a way that they just wait a certain number of seconds before checking that all children are in SUCCESS state and stop forever if that check fails.

3. Serialization of dbi hash in PersistSertificate.pm in order to pass certificate attributes to LDAP-workflow via context meets problems: passing utf8 symbols crashes. It looks like it happens not due to wrong work of Serialize module (it passes utf8 tests). It could be that while processing of the context parameters, the workflow factory alteres utf8 flags. A following workaround is implemented for now: once again get a PEM-encoded certificate from the context, extract its attributes, and use them.

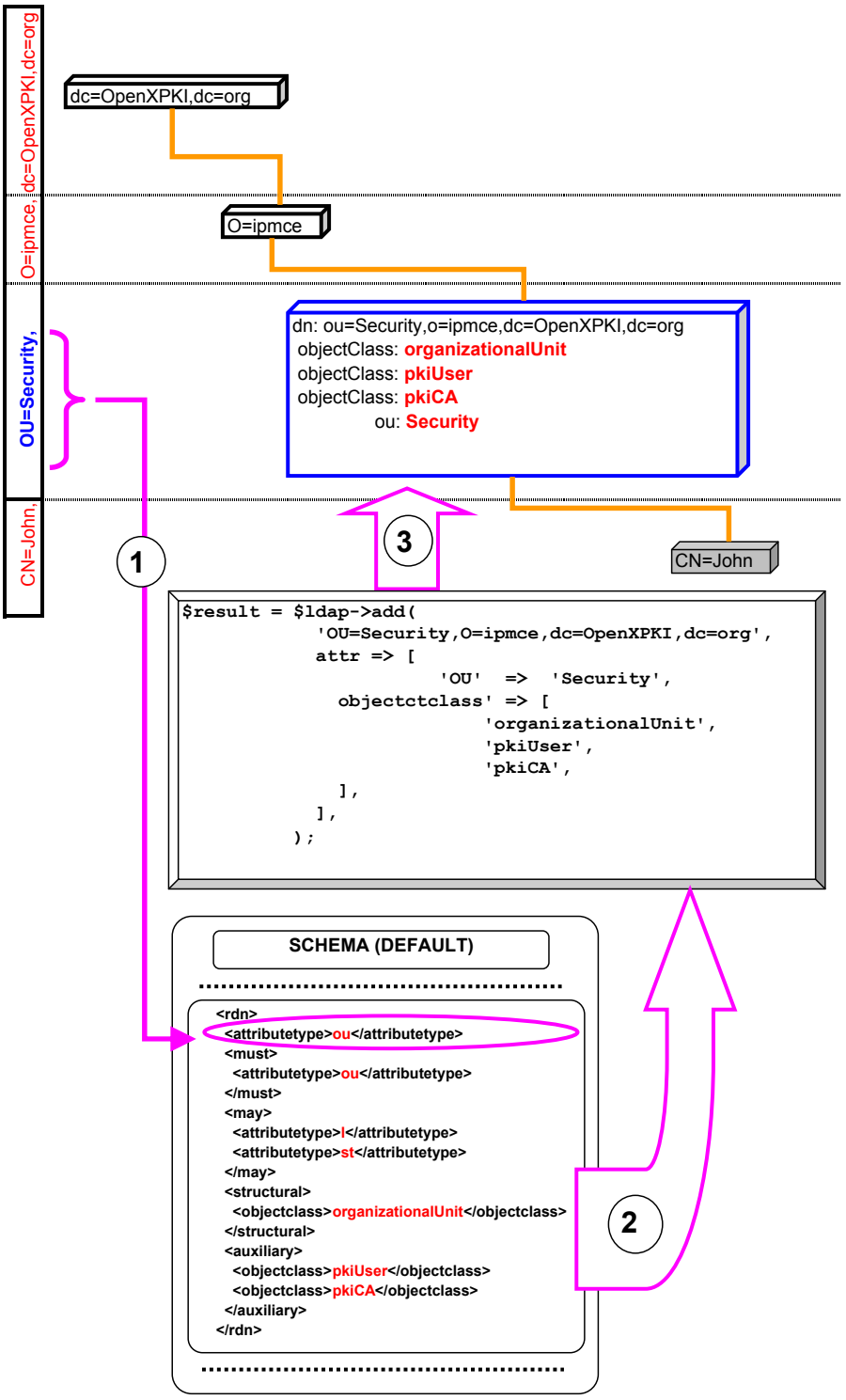

Figure 2. Adding LDAP node in OpenXPKI.

Action: **Details:**

Spawning **CreateCSR** workflow using OpenXPKI client

⇓

Spawning **CertificateIssue** workflow from **CreateCSR** workflow

⇓

Spawning **LDAP-publishing** workflow from **CertificateIssue** workflow

⇓

Importing certificate data from context:

- Getting certificate parameters from context
- Checking if the role marked as excluded and setting in context the 'permission' flag
- Parsing the certificate using OpenXPKI::Crypto::X509
- Checking if the LDAP node exists at that DN and setting the flag to context

- Getting a PEM-coded certificate **context(parameter => 'certificate')**
- Getting a role **context(parameter => 'role')** to check if we have to stop publishing due to role restrictions specified in <ldap_excluded_roles> option
- Getting a mail adresses from the certificate alternative names
- Getting the certificate DN to use it for LDAP-search

⇓

TO BE CONTINUED ON THE NEXT PAGE

Figure 3. Certificate LDAP-publishing workflow schematic. Part 1.

LDAP publishing - we have all the parameters imported

⇓

Adding Node:

- Checking if 'node_exist' flag is set to 'yes' and going to AddNode stage if not
- Checking if role 'permission' flag is set to 'yes' and stop publishing if not
- Parsing the certificate DN
- Process all the RDNS
  - Build the truncated DN ending with the RDN being processed
  - Get all attributes in RDN
  - Get object classes for all attributes according to schema
  - Get attributes values looking sequencely through three hashes
    1. hash of the current RDN
    2. hash of the extra attribures
    3. hash of all previously processed attributes in DN
  - Add a node using truncated DN and built set of attributes

⇓

Adding the certificate to existing LDAP node:

- **pkiUser** attribute is added to existing LDAP node if necessary
- Multiple certificates are supported but doubling is blocked

⇓

SUCCESS STATE: This is the last state of the workflow even if the certificate was not published due to role exclusion. After the LDAP-publishing workflow is finished CertificateIssue workflow meets the condition **check_child_finished** and finishes itself. Then CreateCSR workflow that spawned issuing does the same thing and finally all three workflows are in SUCCESS state.

Figure 4. Certificate LDAP-publishing workflow schematic. Part 2.