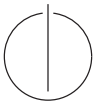
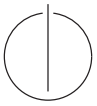



Technische Universität München
Department of Informatics

Interdisciplinary Project in Electrical Engineering

Improving the Interoperability of a Secure Directory Service using HKP, LDAP and DANE

Valentin Hauner

Technische Universität München

Department of Informatics

Interdisciplinary Project in Electrical Engineering

Improving the Interoperability of a Secure Directory Service using HKP, LDAP and DANE

Erweiterung der Interoperabilität eines sicheren Verzeichnisdienstes um HKP, LDAP und DANE

Author	Valentin Hauner
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Dr. rer. nat. Matthias Wachs
Date	October 5th, 2016

Informatik VIII
Chair of Network Architectures and Services

Abstract

Numerous e-mail and address book clients support the retrieval of end-user certificates from servers on the internet via well-known certificate exchange protocols. Next-generation directory services that exclusively provide authenticated certificates need to be accessible via these protocols, too. The aim of this project is to improve the interoperability of those secure directory services with client applications regarding the exchange of certificates. For this purpose, server-side Java implementations of the protocols HTTP Keyserver Protocol (HKP), Lightweight Directory Access Protocol (LDAP) and DNS-based Authentication of Named Entities (DANE) are provided that can be deployed on any target platform with the Java environment installed. The implementations support transport layer security as well as mechanisms for access control. The communication with the back-end of the secure directory service is performed via a clear-cut interface specifically designed for this project, making it possible to re-use this suite for arbitrary secure directory services.

Zusammenfassung

Zahlreiche E-Mail- und Adressbuch-Clients unterstützen den Abruf von Endanwenderzertifikaten von Servern im Internet mittels bekannter Zertifikatsaustauschprotokolle. Verzeichnisdienste der nächsten Generation, die ausschließlich authentifizierte Zertifikate bereitstellen, müssen ebenso über diese Protokolle erreichbar sein. Das Ziel dieses Projekts ist es, die Interoperabilität dieser sicheren Verzeichnisdienste mit Client-Applikationen hinsichtlich des Austausches von Zertifikaten zu verbessern. Dafür werden serverseitige Implementierungen der Protokolle HTTP Keyserver Protocol (HKP), Lightweight Directory Access Protocol (LDAP) und DNS-based Authentication of Named Entities (DANE) in Java bereitgestellt, die auf jeder Zielplattform mit installierter Java-Umgebung eingesetzt werden können. Die Implementierungen unterstützen sowohl Transportschichtsicherheit als auch Mechanismen zur Zugriffskontrolle. Die Kommunikation mit dem Backend des sicheren Verzeichnisdienstes wird über eine klar definierte Schnittstelle abgewickelt, die speziell für dieses Projekt konzipiert wurde, um die Verwendung dieser Suite für sämtliche sichere Verzeichnisdienste zu ermöglichen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Before users can send encrypted e-mail using asymmetric cryptography, they need to retrieve the certificate of the recipient with the associated public key in it. This can either be done by requesting the certificate from the owner directly or by downloading it from a public server on the internet. In the latter case, the sender will have to verify that the retrieved certificate is owned by the person he or she wants to communicate with, e.g. by checking its fingerprint via an independent channel like a personal meeting or by validating a chain of trust, e.g. with the help of a public key infrastructure (PKI). There are several problems with this approach: While the manual verification is unfeasible when it comes to mass communication, a chain of trust is available for the fewest end-user certificates. At this point, the concept of secure directory services comes into play.

With the help of a secure directory service, users can exchange authenticated certificates over the internet. It is the responsibility of the entity running the service to verify the real identities of the users and ensure that they can upload certificates for their own identity exclusively. With this approach, the forgery of identities can be avoided so that everyone who retrieves a certificate from the secure directory service can be sure that it is owned by the claimed identity.

Users need to be able to conveniently retrieve certificates from the secure directory service with their e-mail clients or comparable applications. The design and introduction of a completely new certificate exchange protocol supporting transport layer security, authentication and life cycle management is desirable, but cannot be realized that quickly. In order to enable users to retrieve certificates from the secure directory service by now, it has to be accessible via interfaces that are currently supported by client applications for the exchange of certificates. As many existing implementations of certificate exchange servers like the Synchronizing Key Server (SKS) lack platform independence, do not provide security features such as authentication and are maintained and extended by the

corresponding developer team, they should not be re-used for the secure directory service. Instead, it is reasonable to provide new and detached implementations of those interfaces that are platform independent and not tailored to specific secure directory services, but can be re-used in arbitrary environments.

1.1 Goals of the project

This project's goal is to improve the interoperability of secure directory services with mail user agents (MUA) and other client software by providing implementations of well-known interfaces used to retrieve end-user certificates from servers on the internet. In particular, it deals with Java-based implementations for the protocols HTTP Keyserver Protocol (HKP), Lightweight Directory Access Protocol (LDAP) and DNS-based Authentication of Named Entities (DANE). The interface components developed during this project can be attached to any back-end component providing the defined Java methods for the different query types that are described later on. The job of the back-end is to hand over the requested certificates after having authenticated the inquirer and fetched the data from the connected storage service while taking into account the access policy defined.

1.2 Outline

This thesis is structured as follows: Chapter 2 introduces basic information on directory services, certificate standards and the protocols HKP, LDAP and DANE, before Chapter 3 presents related work. In Chapter 4, the conceptual design of the software is explained using the Unified Modeling Language (UML) and the Interface Definition Language (IDL). Chapter 5 describes the implementation details and provides examples on how to integrate the artifacts into existing projects. Finally, the thesis is concluded in Chapter 6 where the main findings are summarized, limitations of the current work are identified and future plans regarding this project are sketched.

Chapter 2

Background

This chapter introduces the characteristics of a secure directory service and the background information on the certificate standards and exchange protocols required to understand this project's design and implementation.

2.1 Directory services

A directory service is used to retrieve and store resources of arbitrary types in a hierarchical structure independent of the physical storage location. Typically, it is used in a network environment, provided by a so-called directory server. The client queries the server for resources with a specific name or certain attributes using a directory access protocol and gets back a collection of results. Depending on the service's configuration, clients may be able to modify resources, too. Modern implementations of directory services offer enhanced abilities for user authentication and authorization.

There is no requirement for directory services to deliver authenticated information. That is why the concept of secure directory services has been established. Within the scope of this project, its goal is to exclusively provide information that has been verified and authenticated by the entity who is running the service. To avoid tampering and eavesdropping of the data that is transferred between the clients and the server, transport layer security is mandatory.

This project focuses on secure directory services that manage end-user certificates deployed for signing and encrypting data. To avoid the forgery of identities, only authenticated and authorized users are able to upload certificates, exclusively for their own identities. The provider is responsible for checking the real identity of each new user, e.g. by reviewing the identity card, sending a verification code via regular mail or by performing a standardized digital identification technique.

Furthermore, the ownership of the e-mail addresses associated with the user's certificates has to be verified, e.g. by sending e-mails with verification codes to that addresses. If allowed by the certificate format, signatures of existing certificates can be uploaded, but must be reviewed by the respective owner. Secure directory services may even provide more functionalities: The visibility of certificates or even certain identities of a certificate may be restricted to internal, external or customized user groups, protecting the owners from leaking sensitive information. As only authenticated users can upload certificates, users retrieving certificates from the server do not necessarily have to verify that the received certificates are owned by the persons they want to communicate with, but are advised to make sure that the user identity of the certificate is really the queried one.

2.2 Certificate standards

2.2.1 OpenPGP

OpenPGP is a standardized message format used for encrypting and signing data and for authentication. It is described in RFC 4880 [1]. Each OpenPGP certificate, often referred to as *OpenPGP public key*, consists of several independent components: a single primary key, one or more user identities and optional sub keys. These components must be signed separately by the owner and may be signed by multiple third parties. Since there are no signatures for the whole certificate, it is possible to strip certain components away without making the certificate invalid. The calculation of the certificate's fingerprint is based on the primary key. OpenPGP uses hybrid encryption supporting RSA, AES and elliptic curve cryptography; signatures are created with RSA or DSA/ECDSA.

After having exchanged OpenPGP certificates, the parties involved must ensure their authenticity, either by checking the certificates' fingerprints or by tracing an appropriate chain of trust. Since the OpenPGP standard defines no public key infrastructure, it introduces the so-called *Web of Trust* (WOT), a peer-to-peer model to verify the authenticity of a certificate. Key owners can sign other keys' identities and build a chain of trust: If participant A signs the key of participant B and trusts the signatures created by participant B and participant B signs the key of participant C, participant A considers the key of participant C to be authentic.

2.2.2 X.509

X.509 is a public key infrastructure standard described in RFC 5280 [2]. X.509 certificates can be used for encrypting and signing data as well as for securing

connections with TLS. A X.509 certificate contains a serial number, a validity period, the distinguished name of the certificate owner, the public key, the issuer and finally the signature. The algorithms used for signing and encrypting must be specified in the certificate; supported algorithms are, among others, RSA and DSA/ECDSA. The certificate may be signed by the owner, referred to as a *self-signed certificate*, or by a certificate authority (CA). In the latter case, the authenticity of the certificate can be checked by everyone who has installed the root certificate of the corresponding certificate chain on their system. In contrast to the Web of Trust, a public key infrastructure is a hierarchical approach for verifying the authenticity of a certificate.

2.3 Certificate exchange protocols

2.3.1 HKP/SKS

The OpenPGP HTTP Keyserver Protocol (HKP) defined in an IETF draft of 2003 [3] is a HTTP-based key server protocol. Although it was never standardized, HKP is the most popular protocol for exchanging OpenPGP keys. It offers different methods to query the key server via HTTP GET requests and specifies a human-readable response format as well as a machine-readable one. Submissions of keys are done via HTTP POST requests. Table 2.1 gives an overview of the interface. As the draft defines no own authentication mechanisms, HTTP authentication is the sole way to restrict the access to the key database. Most client software, however, does not support HTTP authentication together with HKP. The default port for HKP is 13371, while the TLS-secured HKPS is usually accessible via the standard HTTPS port 443.

Lookup keys via `GET /pks/lookup HTTP/1.1`		
attribute	**value**	**description**
search	[pattern]	search pattern (key IDs are prefixed with `0x`)
op	index	return result list with meta information about keys
	vindex	return result list with extended meta information about keys
	get	return ASCII-armored representation of the resulting key ring
options	mr	return machine-readable output
exact	on	return exact matches only
Add keys via `POST /pks/add HTTP/1.1`		
attribute	**value**	**description**
keytext	[key data]	ASCII-armored representation of the key ring to submit

Table 2.1: Interface of the OpenPGP HTTP Keyserver Protocol (HKP)

Synchronizing Key Server (SKS) is an implementation of an OpenPGP-based key server that speaks HKP [4]. It is written in OCaml and focuses on efficient and reliable synchronization with other key servers. By now, almost 90 key servers are connected to the SKS pool [5]. Each key that is uploaded to one of these servers gets transferred to the others to achieve worldwide distribution.

2.3.2 LDAP

The Lightweight Directory Access Protocol (LDAP) standardized in RFC 4511 [6] is one of the most popular directory service protocols and follows the request-response pattern. The directory is provided as a tree structure, allowing both inner nodes and leaves to store data. Nodes are accessible via a so-called *distinguished name* (DN), a path statement consisting of several *relative distinguished names* (RDN). A *schema* defines *object classes* that in turn specify obligatory and optional attributes. Each entry can be assigned to several object classes, requiring it to provide the obligatory attributes. Figure 2.1 illustrates this concept on the basis of an exemplary *directory information tree* (DIT). Since LDAP supports enhanced authentication and authorization mechanisms, it is possible to enforce sophisticated access policies.

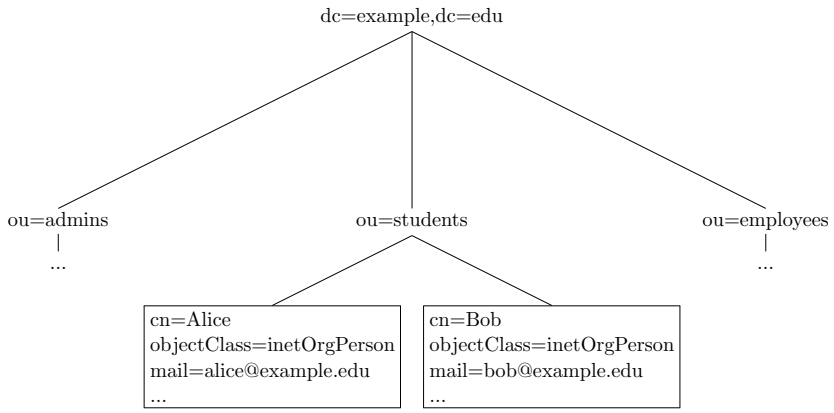


Figure 2.1: An exemplary directory information tree (DIT); the DN of Alice is `cn=Alice,ou=students,dc=example,dc=edu`

2.3.2.1 OpenPGP

For retrieving OpenPGP keys from a LDAP server, a special schema supported by the GnuPG suite is provided [7]. It defines a new object class named *pgpKeyInfo* that holds attributes for the ID, user identities, time of creation and expiration and the raw data of an OpenPGP key. The location of these key entries must be

propagated via a separate entry with the fixed DN `cn=PGPServerInfo,ou=system`. GnuPG supports the LDAP authentication mechanism using the *bind* directive.

2.3.2.2 X.509

The object class *inetOrgPerson* that is defined in RFC 2798 [8] and included in most LDAP server implementations provides, among basic personal attributes like the name, street, locality, telephone number and e-mail address, a dedicated attribute for storing X.509 certificates named *userCertificate* (or alternatively *userSMIMECertificate*). Various e-mail clients supporting X.509 certificates are trying to query for this attribute to encrypt e-mails with the recipient's public key when a LDAP directory is used as an address book.

2.3.3 DANE

DNS-based Authentication of Named Entities (DANE) [9] is a protocol based on the Domain Name System (DNS) to bind certificates of servers and end-users to DNS names and securing the corresponding resource records with DNS Security Extensions (DNSSEC). Originally designed to enable owners of a DNS zone to deliver their TLS server certificates to clients securely using TLSA records and therefore reduce the dependency on certificate authorities [10], it is possible to assign S/MIME certificates and OpenPGP keys to DNS names in the meantime.

The SMIMEA resource record has not been standardized by the time of the writing of this thesis, but is available as an IETF draft [11]. Its structure is equal to that of the TLSA record used for binding TLS server certificates to DNS names. The first two fields of the record describe the certificate's usage and encoding. The matching type field specifies if the record contains the certificate's raw data or just a hash of it. Since the DNS does not allow all characters supported in the local part of an e-mail address, the record's name is prepared as follows: First, the local part of the e-mail address is canonicalized, then hashed with SHA-256 and finally suffixed with the type label `_smimecert` and the e-mail domain name. Figure 2.2 shows an exemplary SMIMEA record.

```
c93f1e400f26708f98cb19d936620da35eec8f72e57f9eec01c1afd6
._smimecert.example.com IN SMIMEA (
   3 0 0 30820307308201efa003020102020... )
```

Figure 2.2: An exemplary SMIMEA record for the e-mail address *hugh@example.com*, providing a so-called domain-issued certificate

The OPENPGPKEY record is on its way to standardization as RFC 7929 [12]. Compared to the SMIMEA record, its structure is much simpler, only consisting of a value for the raw data of a single OpenPGP key. In particular, it is not possible to publish the hash or fingerprint of the key only. Due to the size of OpenPGP keys, the authors recommend using TCP instead of UDP to perform queries for this record type. Moreover, they suggest to strip away unnecessary data incorporated into the key, such as embedded images, non-matching user IDs and unimportant third-party signatures, in order to keep the record's size as small as possible. The record's name is built similarly to SMIMEA, but with the type label `_openpgpkey`. Figure 2.3 provides an example.

```
c93f1e400f26708f98cb19d936620da35eec8f72e57f9eec01c1afd6
._openpgpkey.example.com IN OPENPGPKEY (
   mQINBFeZ2kkBEACt818CrW3/sl7yQabKp0r... )
```

Figure 2.3: An exemplary OPENPGPKEY record for the e-mail address *hugh@example.com*

Chapter 3

Related work

Currently, there are few implementations of secure directory services or comparable solutions. Many organizations are known to host their own internal certificate server in order to control which entities can upload and download certificates. However, this project focuses on open-source implementations of secure directory services that can be run by everyone publicly.

The PGP Corporation developed a so-called *next-generation key server technology* named *PGP Global Directory* [13] that sends verification codes to the e-mail addresses of submitted OpenPGP keys. These verification messages are sent repeatedly every six months to ensure that the e-mail accounts are still active. Moreover, owners can remove keys from the directory. Technically, this is not a revocation since the key is removed physically after the owner has verified the process by another verification message. Each verified key is signed by a dedicated OpenPGP verification key of the provider to attest that the corresponding e-mail addresses are valid and the key owner has given the permission to publish the key. Any user having uploaded his key to the directory can sign other keys with it, without the need of the owner to approve the change. Signatures created by keys that are not hosted on the directory get stripped away. The directory can be queried via the web interface as well as the GnuPG suite with LDAP. Although the developer highlights that it is difficult for spammers to gather large quantities of e-mail addresses due to CAPTCHAs and exact search criteria, these measures can be bypassed by simply using the LDAP interface. As a major drawback, the implementation is closed-source and thus can neither be enhanced by the open-source community nor installed on one's own server.

There are several e-mail providers that offer an integrated encryption and key life cycle management system. A popular one is ProtonMail, a hosted e-mail service that was developed by CERN in 2013 as a reaction to the Snowden affair and provides several special features [14]. The keys used for encryption are

generated by the client's browser, ensuring proper end-to-end encryption and avoiding eavesdropping even by the provider itself. Furthermore, the web client of ProtonMail has been licensed under the MIT license in 2015 and therefore is completely open-source. This makes it easy to verify that strong and securely implemented cipher suites without any backdoor are used. However, ProtonMail is a hosted software-as-a-service solution and thus not fully compatible with the existing e-mail architecture: The service has to be used via the website or the provided mobile applications; neither IMAP nor SMTP are supported. And when sending an encrypted e-mail to an address that is not registered on ProtonMail, the recipient cannot read the e-mail with his MUA, but only after having opened the ProtonMail website using the received link.

Another famous e-mail service focused on encrypted communication is Posteo [15], a German provider with servers housed in Germany. It uses DANE/TLSA as well as dynamic HTTP certificate pinning to prevent fraudulent certificate authorities from issuing non-authorized server certificates. As a major drawback, Posteo does not offer end-to-end encryption since users have to transmit their private keys to the provider. To still achieve end-to-end encryption, Posteo recommends a separate browser add-on developed by a third party that encrypts the e-mails locally.

Chapter 4

Design

The access layer designed in this project enables clients to retrieve end-user certificates from the attached secure directory service via the well-known protocols HKP, LDAP and DNS. For this, the Application Programming Interface (API) consists of three so-called certificate exchange interfaces. Each of these interfaces is provided by a separate controller, whereas each controller is responsible for exactly one protocol. This approach ensures high cohesion, a paradigm of software engineering which requires that the functionalities within a software component should be related strongly and fulfill a single and distinct task.

The controllers retrieve the data needed for their transactions from the back-end component via a well-defined interface with dedicated operations for the different query types. With this concept, the API and the back-end component are coupled loosely which allows the substitution of the back-end's implementation at any time and thus the re-use of the system for arbitrary directory services. The component diagram in Figure 4.1 illustrates the overall design.

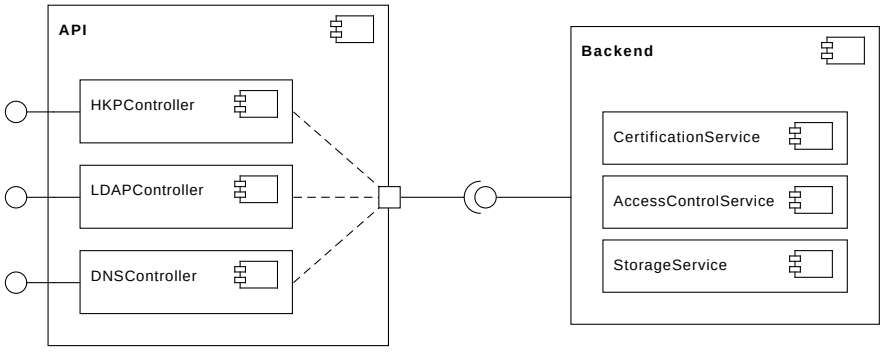


Figure 4.1: High-level design of the API and the interacting back-end, illustrated in an UML component diagram

4.1 Certificate exchange interfaces

The certificate exchange interfaces can be accessed by any client application using the respective exchange protocol. The HKP controller handles HKP requests as defined in the IETF draft of 2003 [3] and accepts connections over TLS. The LDAP controller speaks LDAPv3, including StartTLS and the directives needed for authentication, and supports the OpenPGP schema used by the GnuPG suite as well as the *inetOrgPerson* schema used by e-mail clients for their address books. The DNS controller manages a complete DNSSEC-secured zone and runs a basic DNS server speaking the zone transfer protocol AXFR.

4.2 Back-end interface

The back-end interface can be accessed by the controllers via several clear-cut operations, each of them designed for a dedicated query type and with parameters for the caller, the query pattern and the matching rule. Figure 4.2 shows its abstract description in the Interface Definition Language (IDL). It covers operations to retrieve persons by ID, e-mail address and several other meta attributes. The structure named InetOrgPerson is inspired by the corresponding LDAP object class, containing attributes such as full name, mail address, e-mail address and X.509 certificate. In addition to that, it is able to store an OpenPGP certificate. Nevertheless, the interface defines separate operations for retrieving OpenPGP certificates, making the handling of HKP and LDAP requests initiated by the GnuPG suite much more convenient.

The back-end is responsible for providing the query response after having authenticated the caller, checked the access rights according to the access policy and fetched the data from the attached storage service. The access policy can be used to restrict the access to certain certificates to a defined group of users. If allowed by the certificate format, it is even possible to return particular parts of a certificate only, e.g. those user identities of an OpenPGP certificate that are intended for the public.

Besides, the interface provides operations for retrieving and authenticating the so-called subjects. Subjects are registered in the back-end component with an identifier and a password, but do not necessarily own all the attributes of InetOrgPerson. For client requests containing authentication data, the controllers hand over the authenticated caller as an instance of Subject to the back-end's query operations. An anonymous subject is provided for requests where no authentication data has been supplied by the client.

```
interface Backend {

  OpenPGPCertificate getOpenPGPCertificateByFingerprint(
      Subject caller, string fingerprint, MatchingRule rule);

  OpenPGPCertificate getOpenPGPCertificateByID(
      Subject caller, string id, MatchingRule rule);

  sequence<OpenPGPCertificate> getOpenPGPCertificatesByMetaInfo(
      Subject caller, string pattern, MatchingRule rule);

  sequence<OpenPGPCertificate> getOpenPGPCertificatesByEmailDomain(
      Subject caller, string emailDomain);

  InetOrgPerson getInetOrgPersonByID(Subject caller, string id);

  sequence<InetOrgPerson> getAllInetOrgPersons(Subject caller);

  sequence<InetOrgPerson> getInetOrgPersonsByMetaInfo(
      Subject caller, string pattern, MatchingRule rule);

  sequence<InetOrgPerson> getInetOrgPersonsByEmailAddress(
      Subject caller, string pattern, MatchingRule rule);

  sequence<InetOrgPerson> getInetOrgPersonsByEmailDomain(
      Subject caller, string emailDomain);

  Subject getSubjectByID(Subject caller, string id);

  Subject getSubjectByPrincipal(Principal principal);

  Subject getAnonymousSubject();

  boolean authenticateSubject(string id, string password);

}
```

Figure 4.2: Interface of the back-end, described in the Interface Definition Language (IDL)

Chapter 5

Implementation

This project is implemented in Java 7 to achieve a high degree of platform independence. Therefore, it was not possible to reuse existing implementations like the OCaml-based Synchronizing Key Server (SKS) or the popular DNS server BIND which is written in C. Instead, the certificate exchange interfaces have been rewritten from scratch using free and well-maintained Java libraries.

Spring Boot[1] is used in order to provide a stand-alone application that can be executed right away for testing and demonstration purposes, without the need for time-consuming setup and configuration tasks. The different controllers, however, do not rely on Boot, making it easy to deploy them for environments and application servers of one's own choice. The project's build process and dependencies are managed with Apache Maven[2].

5.1 HKP

The HKPController is based on a web controller of the Spring Framework[3] which offers a convenient way for mapping HTTP requests onto handler methods and accessing the request parameters via ordinary method parameters. If Spring is not available on the target platform, it can be replaced with the servlet concept of Java EE without any great effort since Spring itself is built on it.

The controller supports not all of the features defined for HKP. In particular, it returns no HTML responses prepared for web browsers, but machine-readable output only. To get a human-readable representation of the result, HKP-compatible client software such as the GnuPG suite can be used. As the protocol offers no

[1] `https://projects.spring.io/spring-boot/`
[2] `https://maven.apache.org/`
[3] `https://projects.spring.io/spring-framework/`

dedicated authentication mechanism, the submission of OpenPGP certificates via HKP is not possible. Thereby, users are not able to forge identities by uploading certificates which are not owned by themselves.

The central part of the controller is the handler method *lookup*. It evaluates the request parameters, queries the back-end and generates the appropriate response for the client. In addition to the HKP draft, a parameter named *limit* is supported. If set to a positive integer, the result set gets trimmed to the defined number of entries. This can be useful to decrease the network load and is supported by SKS as well.

HTTP status codes are used to signal problems with the client's query. The code 404 will be returned if the result set is empty, while the code 400 indicates that the query parameters themselves are invalid, e.g. if the mandatory option for generating machine-readable output is not set. The status 501 refers to a operation that is not supported, such as the submission of certificates. Finally, the code 500 will be sent if an internal server error occurs.

5.2 LDAP

The LDAPController is based on ApacheDS[4], a complete LDAP server written in Java by the Apache Software Foundation. It is fully compatible with LDAPv3 and embeddable in any Java application due to the well-documented API. The behaviour of ApacheDS can be extended and influenced using the interceptor pattern. Interceptors are components that alter the default sequence of actions of a software system by executing custom routines. Usually, these routines are triggered by events, like the arrival of a new client request.

Since the back-end of the secure directory service stores and manages the certificates and meta data via a dedicated storage service, the database features of ApacheDS are not used in this implementation. Instead, it acts as a virtual LDAP server, responsible for handling the authentication of clients, parsing LDAP requests, handing them over to the interceptors and sending back the responses received from the interceptors to the clients.

Figure 5.1 illustrates the internal structure of the LDAPController. The embedded instance of ApacheDS is extended with two custom-built interceptors for retrieving OpenPGP certificates and InetOrgPersons, respectively. The interceptors receive the LDAP requests that have been parsed and transferred to Java objects by ApacheDS, query the back-end component accordingly and send back the result objects to ApacheDS, which returns an LDAP-formatted response

[4] `https://directory.apache.org/apacheds/`

to the client. If the actual request is preceded with a *bind* request containing authentication data, a custom authenticator will check whether the subject is registered in the back-end with the given credentials. Only if this is the case, the authentication will succeed and the actual request will be continued.

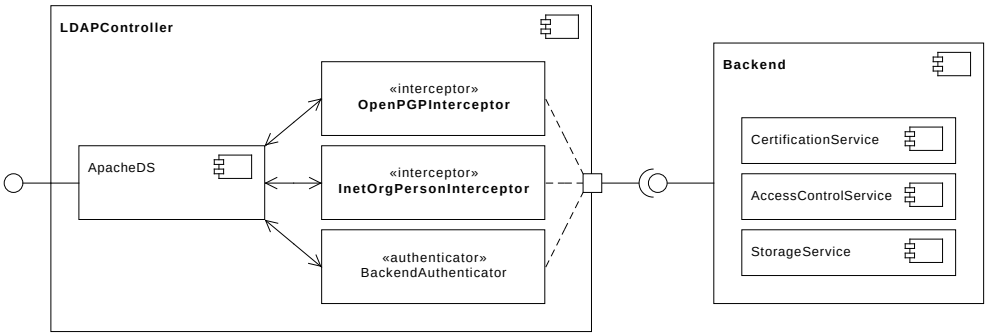


Figure 5.1: Simplified internal structure of the LDAPController, using an embedded instance of ApacheDS as a virtual LDAP server

As introduced in Chapter 2, a special LDAP schema is needed to handle requests for OpenPGP certificates performed by the GnuPG suite. This schema is imported into ApacheDS using a LDIF file. After having obtained the results from the back-end, the OpenPGPInterceptor constructs Java objects of the ApacheDS type Entry out of them. The LDAP object class *pgpKeyInfo* is assigned to these entries, requiring attributes for the certificate's ID, user identities, encoded data and so forth. Conceptually the same happens for InetOrgPerson requests: The InetOrgPersonInterceptor fetches the results from the back-end and creates entries of the LDAP object class *inetOrgPerson*, containing attributes for the person's full name, mail address, e-mail address and X.509 certificate. Both interceptors are implemented to exclusively react to requests for a certain base DN specified during the initialization process: While the OpenPGPInterceptor's base DN is set to the domain's organizational unit named *PGP Keys*, the InetOrgPersonInterceptor is responsible for that with the name *inetOrgPersons*.

The filters of LDAP search requests defined in Section 4.5.1.7 of RFC 4511 [6] can become randomly complex. Commonly, they are represented as a tree structure: On the one hand there are branch nodes for conjunction, disjunction and negation, on the other hand there are leaf nodes which model equality, substrings and further assertion types. The expression `(&(mail=*@example.org)(l=springfield))` is an exemplary filter, requiring the resulting entries to have an e-mail address of the domain *example.org* and the exact locality *springfield*.

However, the API of the back-end supports queries of low complexity only, consisting of a single search pattern for one or several attributes, complemented with a dedicated matching rule parameter to differentiate between exact and fuzzy matches. To be still able to query the back-end for LDAP requests, the equality or substring leaf node of the most decisive attribute supported by the back-end is extracted from the filter and used as a substitute. Most decisive in this context means that an order of precedence has been defined on the attributes, e.g. the e-mail attribute has obvious priority over the locality attribute. Since the gained result may be a superset of the actual one, the original filter is applied to all entries before sending the LDAP response to the client. This makes sure that every result entry satisfies the whole filter.

5.3 DNS

The DNSController uses dnsjava[5], a well-maintained Java library written by Brian Wellington for managing DNS zones in an object-oriented way. It supports all standardized record types, including those for DNSSEC. During this project, the need for the new record types OPENPGPKEY and SMIMEA proposed by the DANE working group emerged. Brian Wellington has kindly extended his library with these, enabling a state-of-the-art way to assign OpenPGP and X.509 certificates to e-mail addresses via DNSSEC.

Zones can be exported to master files, which are ready to use for DNS server implementations like BIND, or provided via the zone transfer protocol AXFR. For the latter, a very basic server called *jnamed* shipped with dnsjava is started. Since the developer does not recommend his implementation for production, however, it should be used for demonstrative purposes only.

The DNSController is instantiated with a zone name, a key signing key (KSK) as well as a zone signing key (ZSK) needed for DNSSEC and several meta attributes like the default TTL value. Each instance manages a complete DNS zone, including the SOA and at least one NS record. According to the DNSSEC specification, the DNSKEY records are signed with the KSK, while all other resource record sets are signed with the ZSK. Unfortunately, dnsjava is not capable of creating a full NSEC3 chain to prove the non-existence of queried records. Therefore, a separate utility class has been developed in this project that is compatible with dnsjava and generates a NSEC3 chain fulfilling the requirements of RFC 5155 [16]. It traverses the given zone, inserts the so-called empty non-terminals, hashes the record names according to the parameters of the given NSEC3PARAM record, orders the hashes ascendingly and finally creates the NSEC3 records.

[5] `http://www.dnsjava.org/`

Each certificate that is made public by the back-end and associated with at least one e-mail address of the zone's domain name is published in the DNS zone using either an OPENPGPKEY or a SMIMEA record. Every record contains a full certificate and, as a matter of course, is signed with the ZSK. If several e-mail addresses of a certificate match the zone's domain name, they are assigned to CNAME records pointing to the actual certificate record. To be able to retrieve the relevant certificates conveniently, the back-end provides dedicated methods for querying certificates associated with a certain e-mail domain.

5.4 Back-end

The class SimpleBackend is a rudimentary implementation of the interface Back-end. It uses in-memory Java collections to store the certificates and the meta data. When instantiated, it imports a few dummy certificates and creates some test users that are suitable for test requests. Hence, SimpleBackend should not be used for production, but can serve as a source of inspiration on how to implement the different query methods.

The classes BCOpenPGPCertificate and BCX509Certificate are implementations of the interfaces OpenPGPCertificate and X509Certificate, respectively. As their names imply, they are based on Bouncy Castle[6], one of the most comprehensive open-source cryptography libraries available for Java. Internally, the implementations store attributes that are needed most frequently in dedicated fields in order to avoid the browsing of the Bouncy Castle objects on every access. Both classes can be used for production.

The class User is an implementation of the interface InetOrgPerson and ready for practical use, too. It owns dedicated fields for the getters required by the interface and provides setters for attributes where it is appropriate. E-mail addresses that are added via the corresponding method are parsed by the JavaMail API and have to conform to the address format defined in RFC 822 [17].

5.5 Integration into existing environments

The class SimpleApplication is the entrance point for the demo instance of this suite and can serve as an inspirational source on how to integrate it into existing environments. It uses Spring Boot to create and initialize the HKPController, but any Java EE application server capable of launching a Web Application Archive (WAR) can be used, too. The LDAP service and server are started by creating

[6] `https://bouncycastle.org/`

a new instance of the class LDAPController with the domain name as the sole parameter. Another constructor is available to manipulate the names of the organizational units of the certificates. To create a new DNS zone and start the basic name server, the class DNSController has to be instantiated with the zone's name and the key signing key pair as well as the zone signing key pair as parameters. Two other constructors are provided to generate the keys automatically or to define meta data like the TTL values and name servers manually.

Chapter 6

Conclusion

This project showed how to improve the interoperability of secure directory services with client applications by designing and implementing server-side interfaces for the well-known certificate exchange protocols HKP, LDAP and DANE. As a result, it is possible to access these services with existing e-mail clients and comparable software, facilitating the distribution and acceptance of secure directory services. The platform independence of the interface implementations as well as their stringent separation from the back-end component allow the re-use for arbitrary secure directory services and server environments. Due to the modular design of the suite, the support for further protocols can be added easily. While the implementation of HKP had to be rewritten completely because of lacking Java libraries, well-maintained third-party software for LDAP and DNS has been used to handle the actual protocol transactions, making the incorporation of future updates of these protocols much more conveniently.

In contrast to the proprietary and closed solutions introduced in Chapter 3, the artifacts of this project are free and open-source. They have been developed to be utilized and enhanced in further academic projects to achieve the overall goal of this contribution: the promotion and expansion of secure digital communication by simplifying the exchange of certificates needed for encrypting and signing data.

6.1 Future work

Although this implementation is feature-complete with regards to the project's requirements, there is still future work to do. Currently, it is not possible to assign more than one OpenPGP or X.509 certificate to a single InetOrgPerson, respectively. While this is less critical for OpenPGP certificates which themselves can hold multiple user identities, it may be useful for upcoming use cases to

store several X.509 certificates per entity. It is, however, not recommended to send all certificates of a person to the client at once since the network load increases unnecessarily and the client application may not know how to handle the response. In order to ensure that only the certificate belonging to the requested e-mail address is transmitted, the e-mail addresses which the certificates have been issued for must be checked beforehand.

As another limitation, certificates cannot yet be added via LDAP although the protocol itself provides highly developed authentication mechanisms. The possibility of modifying information via LDAP would decrease the users' dependency on non-standardized interfaces of secure directory services like web front-ends. Unfortunately, most of today's e-mail clients do not support the upload of data to a LDAP server.

It may be desirable to extend the back-end's interface to support queries with higher complexity. For this, new tree-based data structures can be designed, modeling complex query expressions similar to those used in LDAP. Consequently, less filter operations would be performed by the interface component, but by the back-end's database management system which is usually optimized for performance and efficiency. This is particularly important when a huge amount of data is stored in there.

Moreover, the exemplary DNS server *jnamed* that is currently used for the handling of AXFR requests should be replaced with an implementation that has been approved for productive use. Within the DNSController, the process of creating DNS zones has further potential for optimization: records have to be re-generated and re-signed only after the underlying information has changed. Additionally, the key signing key can be kept offline most of the time for security reasons since it is just needed for signing a new zone signing key. As recommended in RFC 7929 [12], unnecessary information in OpenPGP certificates should be stripped away before publishing them via DNS to reduce the size of the OPENPGPKEY records. For SMIMEA records containing huge X.509 certificates, it has to be considered not to publish their raw data, but just their hashes.

Bibliography

[1] Callas J. and Donnerhacke L. and Finney H. and Shaw D. and Thayer R., "RFC 4880 - OpenPGP Message Format," https://tools.ietf.org/html/rfc4880, accessed on August 13th, 2016.

[2] Cooper D. and Santesson S. and Farrell S. and Boeyen S. and Housley R. and Polk W., "RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," https://tools.ietf.org/html/rfc5280, accessed on August 13th, 2016.

[3] Shaw D., "draft-shaw-openpgp-hkp-00 - The OpenPGP HTTP Keyserver Protocol (HKP)," https://tools.ietf.org/html/draft-shaw-openpgp-hkp-00, accessed on August 13th, 2016.

[4] Minsky, Yaron and Clizbe, John and Fiskerstrand, Kristian and Pennock, Phil, "Synchronizing Key Server (SKS)," https://bitbucket.org/skskeyserver/sks-keyserver/wiki/Home, accessed on August 14th, 2016.

[5] Fiskerstrand, Kristian, "Synchronizing Key Server (SKS) Pool," https://sks-keyservers.net, accessed on August 14th, 2016.

[6] Sermersheim, J., "RFC 4511 - Lightweight Directory Access Protocol (LDAP): The Protocol," https://tools.ietf.org/html/rfc4511, accessed on August 15th, 2016.

[7] Walfield, Neal, "LDAPKeyserver - GnuPG wiki - Installing an Additional Schema," https://wiki.gnupg.org/LDAPKeyserver#Installing_an_Additional_Schema, accessed on August 15th, 2016.

[8] Smith, M., "RFC 2798 - Definition of the inetOrgPerson LDAP Object Class," https://tools.ietf.org/html/rfc2798, accessed on September 6th, 2016.

[9] Kumari, W. and Guomundsson, O. and Farrell, S. and Lepinski, M., "DNS-based Authentication of Named Entities (DANE)," https://datatracker.ietf.org/wg/dane/charter/, accessed on September 6th, 2016.

[10] Hoffman, P. and Schlyter, J., "RFC 6698 - The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," https://tools.ietf.org/html/rfc6698, accessed on August 15th, 2016.

[11] Hoffman, P. and Schlyter, J., "draft-ietf-dane-smime-12 - Using Secure DNS to Associate Certificates with Domain Names For S/MIME," https://tools.ietf.org/html/draft-ietf-dane-smime-12, accessed on August 15th, 2016.

[12] Wouters, P., "RFC 7929 - DNS-Based Authentication of Named Entities (DANE) Bindings for OpenPGP," https://www.rfc-editor.org/rfc/rfc7929.txt, accessed on August 15th, 2016.

[13] Symantec Corporation, "PGP Global Directory - Terms and Conditions," https://keyserver.pgp.com/vkd/VKDHelpPGPCom.html, accessed on August 11th, 2016.

[14] Proton Technologies AG, "ProtonMail - Security Features," https://protonmail.com/security-details, accessed on August 11th, 2016.

[15] Posteo e.K., "Innovative encryption for all: We make email truly secure," https://posteo.de/en/site/encryption, accessed on September 13th, 2016.

[16] Laurie, B. and Sisson, G. and Arends, R. and Blacka, D., "RFC 5155 - DNS Security (DNSSEC) Hashed Authenticated Denial of Existence," https://tools.ietf.org/html/rfc5155, accessed on September 4th, 2016.

[17] Crocker, David H., "RFC 822 - Standard for the Format of ARPA Internet Text Messages," https://tools.ietf.org/html/rfc822, accessed on September 6th, 2016.