Technical Specification

Title	Shamirs Webservice
Author	Christof Reichardt Paul-Ehrlich-Weg 1 D-63110 Rodgau E-Mail: projektstudien@christofreichardt.de
Created on	2021-03-05
Version	1.0.0
Status	Draft

Table of Contents

1 Introduction....4
1.1 Terminology....4
1.2 Product Outline....5
1.3 Security Goals....6
1.3.1 Multiple-Eye Principle....6
1.3.2 System Administration....6
1.3.3 Data Protection....6
1.4 Design Principles....7
2 Use Cases....7
2.1 Generation of Keystores....7
2.2 Query all Keystores....8
2.3 Query single Keystore....8
2.4 Import of the Participants....9
2.5 Session Activation....9
2.6 Session Closure....10
2.6.1 Automatic Closure....10
2.6.2 On Demand....10
2.7 Query all Sessions of a Keystore....11
2.8 Submission of Documents for Review....11
2.9 Digital Signature, Encryption and Decryption....12
2.10 Query all Metadata of Documents belonging to a Session....12
2.11 Query the Metadata of a single Document....13
2.12 Request the content of a Document....13
2.13 Query the Participants of a Secret Sharing Scheme....13
2.14 Query all Slices....14
2.15 Query single Slice....14
2.16 Change the status of a single slice to Fetched....15
2.17 Retransmit password shares to a single Slice....15
3 Solution....17
3.1 External Dependencies....17
3.2 Data Model....17
3.3 Business Logic....18
3.4 Sanitization of the User Input....19
3.5 Resources and Representations....21
3.5.1 Keystore Resource....21
3.5.1.1 Keystore Generation....21
3.5.1.2 Query single Keystore....23

3.5.1.3 Query all Keystores....24
3.5.1.4 Keystore Representation....26
3.5.2 Session Resource....26
3.5.2.1 Query all Sessions of a Keystore....26
3.5.2.2 Update Session....27
3.5.2.3 Query single session....29
3.5.2.4 Session Representation....30
3.5.3 Document Resource....30
3.5.3.1 Process Document....30
3.5.3.2 Query all Metadata of a Session....31
3.5.3.3 Query Metadata of a single Document....32
3.5.3.4 Retrieve the content of a Document....33
3.5.3.5 Metadata Representation....33
3.5.4 Slice Resource....34
3.5.4.1 Query all Slices....34
3.5.4.2 Query single Slice....35
3.5.4.3 Change status to Fetched....36
3.5.4.4 Retransmit password shares....37
3.5.4.5 Slice Representation....39
3.6 Test Plan....40
3.6.1 Automatic Tests....40
3.6.1.1 Unit Tests....40
3.6.1.2 Integration Tests....40
3.6.2 Manual Tests....41
3.6.2.1 Keystore Generation....42
3.6.2.2 View the provisioned Session....44
3.6.2.3 Fetch Slices....45
3.6.2.4 Post documents for review....52
3.6.2.5 Retransmit shares....54
3.6.2.6 Session Closure....57
3.7 Deployment....59
4 Security Considerations....59
5 Privacy....59

1 Introduction

The software intends to provide several cryptographic services – such as the generation of symmetric and assymetric keys, the distribution of secret shares or the digital signature and encryption of electronic documents – by exposing an appropriate REST API. The cryptographic keys used for signing and encrypting usually remain on the server within PKCS12 keystores. Key entries within such PKCS12 keystores are typically encrypted as well, e.g. via password based encryption algorithms.

1.1 Terminology

AES
: Advanced Encryption Standard.

ECDSA
: Elliptic Curve Digital Signature Algorithm.

GET
: A method defined by the HTTP protocol.

HTTP
: Hypertext Transfer Protocol.

JPA
: Java Persistence API, e.g. provided by Hibernate.

JSON
: A lightweight data-interchange format.

MIME
: Multipurpose Internet Mail Extensions.

OAuth2
: An authorization framework specified by RFC 6749.

OpenId Connect
: Specifies services on top of OAuth2 for the identification of End-Users.

Participant
: A user which has subscribed to a secret sharing sheme.

Partition
: A certain distribution of password shares between participants.

PATCH
: A method defined by the HTTP protocol.

PKCS12
: Defines a file format used to store cryptographical objects, e.g. private keys, within a single file.

POST
: A method defined by the HTTP protocol.

PUT
: A method defined by the HTTP protocol.

REST
: Representational state transfer denotes an architectural style related to interactive applications using web services.

Secret Sharing
: Methods for distributing shares of a secret, e.g. a password, between certain participants. The participants need to combine a subset of the shares in able to recover the original secret.

Slice
: A JSON file containing some password shares.

URI
: Uniform Resource Identifier.

XML
: An extensible markup language for encoding of documents.

1.2 Product Outline

The center piece of the service is the administration of keystores which belong to certain participants. These keystores will be generated by the server on demand together with specified cryptographic keys according to submitted instructions. The participants may use their associated keystores for the encryption/decryption or rather digital signing/verification of electronic documents. In order to use a keystore someone must utilize its provisioned session which can only be activated if sufficient and appropriate password shares are available. The expired sessions of a keystore maintain a history of actions which have been executed by using the keystore. Password shares are only valid for a single session. After the closure of a session new password shares will be generated for the keystore and its entries will be reencrypted via usual password based encryption algorithms. As an option the just now computed password shares will be actively distributed to the participants. Alternatively the password

shares might be fetched from the server by their owners via the REST API. If the password shares remaining on the server fall below a certain threshold the associated keystore becomes unloadable.

1.3 Security Goals

1.3.1 Multiple-Eye Principle

It should be possible to enforce the multiple-eye principle when granting access to one of the keystores comprising the cryptographic keys. Consider, for example, a company signing key within such a keystore and an electronic payment order which must be reviewed and digitally signed. This can be achieved by applying a secret sharing algorithm in addition to the password based encryption of the keystores.

1.3.2 System Administration

Ideally, the system administration of the cryptographic services should not be able to gain access to the cryptographic keys even with root access to the server. This means that the passwords – or rather password shares – of the keystores should not remain on the server after their initial generation. Either the password shares will be distributed, e.g. via (encrypted) E-Mail messages, immediately after their creation or the clients fetch them via the REST API in due course. In order to open a session related to a certain keystore the appropriate shares must be again transmitted back to the server by the participants. After the closure of the session the key entries of the involved keystore will be re-encrypted and new fitting password shares will be distributed. That means that a seizure of the server alone should normally not lead to a security breach. Encrypted documents or private keys required for digital signatures remain safe because the related keystores cannot be loaded without the appropriate password shares.

1.3.3 Data Protection

The keystores can be optionally exported to a physically separate backup system on a regular basis to prevent data loss e.g. due to a hard disk failure of the main system. As soon as the associated password shares become invalid after a session closure, the backup system will be notified to wipe out the corresponding expired keystore incarnations. The whole database system might be backed up when a save point is reached. A save point is reached when all expired password shares have been cleaned up and all keystore instances have become unloadable due to insufficient available password shares.

1.4 Design Principles

The design of the REST API follows Fieldings ideas where it seems applicable and desirable. The domain objects (keystores, participants, shares or slices, sessions and documents) will be mapped on resources and can be manipulated through representations (or sometimes rather instructions). The service will answer calls by a set of URIs representing possible state transfers ("Hypermedia as the engine of application state"). For example POSTing a keystore representation (or rather instructions how such a keystore should be generated) will be answered with a HTTP 201 CREATED together with a JSON representation of a keystore comtaining a "self" link identifying the new keystore resource on the server. Following the "self" link gives a more complete JSON representation together with some more URIs denoting related domain objects, e.g. session resources. Such a session resource could be used to POST documents for further processing (encryption, signing, …).

The resources of the REST API are backed by an object oriented model of the domain objects which in turn are mirroring the physical database schema.

2 Use Cases

The use cases listed below describe the "happy path" within the Postconditions section. Many things can go wrong. For example, the preconditons weren't met or the decryption of a document fails because an authentication tag does not have matched the computed value. In these cases an appropriate HTTP 4xx code will be returned together with a message in JSON format hinting the error in more detail if possible.

2.1 Generation of Keystores

Participants may trigger the generation of PKCS12 keystores together with the desired key entries by POSTing keystore instructions. Such key entries are always related to certain algorithms, e.g. ECDSA or AES. The password needed to access the keystore will be splitted into shares and made available for the distribution between the denoted participants.

URI

/shamir/v1/keystores

Method

POST

Preconditions

(1) The actor POSTing a set of instructions must be a registered user with the necessary authorizations.
(2) The denoted participants must be known to the system.

Postconditions

(1) The keystore together with the required key entries has been created.
(2) The related password shares have been computed and (optionally) distributed.
(3) A session with state PROVISIONED has been assigned to the keystore.
(4) The server has responded with HTTP 201 CREATED together with a JSON response comprising a keystore representation.

2.2 Query all Keystores

Infos about the available keystores can be requested. The infos include, inter alia, the keystore IDs, descriptive names of the keystores, the particular sharing shemes (number of shares and threshold), the creation time and an URI to each keystore.

URI

/shamir/v1/keystores

Method

GET

Preconditions

(1) The actor must own the necessary permissions to request the infos.

Postconditions

(1) The server has responded with HTTP 200 OK together with the infos in JSON format.

2.3 Query single Keystore

Extended information about a single keystore can be requested. Besides the name of the keystore, the sharing sheme and the creation date an overwiew of the key entries (alias, algorithm, key size) will be delivered if possible. The keystore may be unloadable due to an insufficient number of available shares. Furthermore a complete list of URIs regarding related entities (sessions and participants) will be presented.

URI

/shamir/v1/keystores/<keystore_id>

Method

GET

Preconditions

(1) The actor must own the necessary permissions to request the infos.

Postconditions

(1) The server has responded with HTTP 200 OK together with the infos in JSON format.

2.4 Import of the Participants

Appropriate information about the participants and actors must be imported to system. It is out of scope to maintain an independent identity management system. Instead the system relies on a separate OpenId Connect system for this purpose. In order to relate the participants referenced by the instructions to generate a certain keystore the system must maintain or buffer some information about the approved participants.

Preconditions

(1) The participants together with their roles and corresponding permissions must be known to a OpenID Connect Provider.

Postconditions

(1) The system has created the participant entities and is able to relate them to the users maintained by the OpenId Connect provider.

2.5 Session Activation

In order to encrypt or digitally sign electronic documents a session belonging to a certain keystore must be activated by PATCHting the appropriating session instructions as JSON object.

URI

/shamir/v1/keystores/<keystore_id>/sessions/<session_id>

Method

PATCH

Preconditions

(1) The present session belonging to the given keystore has the state PROVISIONED.

(2) A sufficient subset of password shares required to recover the password of the keystore is available.

(3) The actor that opens the session owns the required authorizations.

Postconditions

(1) The session belonging to the keystore has got the state ACTIVE.

(2) The session is ready to encrypt/sign documents by applying key entries of the related keystore.

(3) The server has responded with HTTP 200 OK and has transmitted a session representation in JSON format to the actor.
(4) Pending documents which had been scheduled for digital signing or encryption have been processed.

2.6 Session Closure

2.6.1 Automatic Closure

During the session activation a maximum idle period had been transmitted. A concurrent thread running the session sanitizer checks periodically for idle sessions and closes them.

Preconditions

(1) One ore more sessions have been inactive for longer than the denoted idle time.

Postconditions

(1) The idle sessions have been asssigned the state CLOSED and won't process documents anymore.
(2) The related keystores have been assigned new sessions with state PROVISIONED.
(3) The related keystores have been given new passwords and the comprising key entries have been re-encrypted.
(4) Corresponding password shares have been computed and (optionally) distributed.
(5) The preceding password shares related to the idle session have been marked as EXPIRED.

2.6.2 On Demand

A Session can be closed on demand by PATCHing an appropriate session instruction as JSON object.

URI

/shamir/v1/keystores/<keystore_id>/sessions/<session_id>

Method

PATCH

Preconditions

(1) The actor PATCHing the session instruction has the required authorizations.
(2) The session has the state ACTIVE.

Postconditions

(1) The session has been asssigned the state CLOSED and won't process documents anymore.
(2) The related keystore has been assigned a new session with state PROVISIONED.
(3) The related keystore has been given a new password and the comprising key entries have been re-encrypted.

(4) Corresponding password shares have been computed and (optionally) distributed.

(5) The preceding password shares related to the closed session have been marked as EXPIRED.

(6) The server has responded with HTTP 200 OK together with the representation of the closed session in JSON format.

2.7 Query all Sessions of a Keystore

Infos about the sessions belonging to a keystore can be requested. This includes already expired sessions.

URI

/shamir/v1/keystores/<keystore_id>/sessions

Method

GET

Preconditions

(1) The actor must own the necessary permissions to request the infos.

Postconditions

(1) The infos have been delivered in JSON format (HTTP 200 OK).

2.8 Submission of Documents for Review

Documents can be submitted to provisioned sessions scheduling them for digital signing, encryption or further actions. So long as the session won't be activated the submitted original documents can be reviewed by the participants. Two query parameters indicate the key entry to be used by its alias and the desired action which can be any of: encrypt, decrypt, sign or verify.

URI

/shamir/v1/sessions/<session_id>/documents?action=<action>&alias=<key_entry>

Method

POST

Preconditions

(1) The actor must own the necessary permissions to submit the documents.

(2) The session has the state PROVISIONED.

Postconditions

(1) The document has been stored together with its metadata and can be reviewed by the participants.

(2) A metadata representation in JSON format has been returned (HTTP 201 CREATED) including an URI indicating the location on the server.

2.9 Digital Signature, Encryption and Decryption

Documents of certain media types can be digitally signed or rather verified by POSTing them to an URL referencing an active session. The actual behaviour of the service depends on the media type. A posted XML document will be digitally signed by applying XML Signature Processing whereas a PDF document will be processed by different rules. Two query parameters indicate the key entry to be used by its alias and the desired action which can be any of: encrypt, decrypt, sign or verify. The posted documents will be stored on the server. Depending on the size of the transferred document the service might decide to process the request asynchronously.

URI

/shamir/v1/sessions/<session_id>/documents?action=<action>&alias=<alias>

Method

POST

Preconditions

(1) The denoted session is ACTIVE.

(2) The actor has the required permissions.

Postconditions

(1) The document has been processed (encrypted, decrypted, signed or verified) and the the resulting document and its metadata has been stored.

(2) The server has responded either with HTTP 201 CREATED or HTTP 202 ACCEPTED and has transmitted a JSON response comprising the result of the operation and an URI indicating the location of the processed document if applicable.

2.10 Query all Metadata of Documents belonging to a Session

The metadata of the documents submitted to a session can be requested. The metadata include the document title, the current state – that might be pending, processed or faulty – its media type, the intended action, timestamps and the self link..

URI

/shamir/v1/sessions/<session_id>/documents

Method

GET

Preconditions

(1) The actor must own the necessary permissions to request the metadata.

Postconditions

(1) The server has responded with HTTP 200 OK together with the metadata in JSON format.

2.11 Query the Metadata of a single Document

The metadata of a single document can be requested. That includes along with the usual metadata links to all related domain objects, e.g. the link needed to retrieve the actual document content.

URI

/shamir/v1/sessions/<session_id>/metadata/<document_id>

Method

GET

Preconditions

(1) The actor must own the necessary permissions to request the metadata.

Postconditions

(1) The server has responded with HTTP 200 OK together with the metadata in JSON format.

2.12 Request the content of a Document

Often the actual content of a document needs to be fetched, most notably digitally signed documents. Encrypted document content may be retrieved to submit it again for decryption.

URI

/shamir/v1/sessions/<session_id>/documents/<document_id>

Method

GET

Preconditions

(1) The actor must own the necessary permissions to fetch the content.

Postconditions

(1) The server has responded with HTTP 200 OK together with the content as octet stream.

2.13 Query the Participants of a Secret Sharing Scheme

List all participants which have password shares of a particular keystore.

URI

/shamir/v1/keystores/<keystore_id>/participants

Method

GET

Preconditions

(1) The actor must own the necessary permissions to query the participants.

Postconditions

(1) The server has responded with HTTP 200 OK together with the representations of the participants in JSON format.

2.14 Query all Slices

Shares owned by a certain participant that belong to a single session are bundled into a file called a slice. Slices related to closed sessions will eventually be cleared up by the session sanitizer. The self links of the current slices that are associated with provisioned sessions can be used to fetch the actual password shares from the server or to put them on the server again. The call accepts query parameter to limit the search.

URI

/shamir/v1/slices?keystoreId=<keystore_id>&participantId=<participant_id>

Method

GET

Preconditions

(1) The actor must own the necessary permissions to query the slices.

Postconditions

(1) The server has responded with HTTP 200 OK together with the representations of the slices in JSON format.

2.15 Query single Slice

Querying a single slice gives the full representation of a slice comprising, inter alia, its state (CREATED, FETCHED, DELIVERED, POSTED and EXPIRED) and the actual password shares if available.

URI

/shamir/v1/slices/<slice_id>

Method

GET

Preconditions

(1) The actor must own the necessary permissions to retrieve the slice.

Postconditions

(1) The server has responded with HTTP 200 OK together with the full representation of the slice in JSON format including the actual password shares if available.

2.16 Change the status of a single slice to FETCHED

Given a slice with status CREATED or POSTED the status of the slice may be changed to FETCHED.

URI

/shamir/v1/slices/<slice_id>

Method

PATCH

Preconditions

(1) The actor must own the necessary permissions to change the state.

(2) The slice is in status CREATED or POSTED.

Postconditions

(1) The server has responded with HTTP 200 OK together with the full representation of the updated slice.

(2) The status of the slice is FETCHED.

(3) The password shares have been deleted.

2.17 Retransmit password shares to a single Slice

Given a slice with status FETCHED the password shares can be retransmitted to the appropriate slice.

URI

/shamir/v1/slices/<slice_id>

Method

PATCH

Preconditions

(1) The actor must own the necessary permissions to transmit the password shares.

(2) The slice is in status FETCHED.

(3) The JSON object comprising the password shares refers to the same partition as the slice on the server.

Postconditions

(1) The server has responded with HTTP 200 OK together with the full representation of the updated slice.

(2) The status of the slice is POSTED.

3 Solution

The service is organized into three logical tiers: presentation tier (view), the application tier (business logic) and the data tier (database). The presentation tier for this service is razor thin. It comprises solely the (JSON) representations of the domain objects. But following RESTful API design principles these representations are transporting the application state to the clients and might be used to build a rich internet application. The business logic layer (application tier) processes these representations by taking into account the current database state. The application tier itself is completely stateless.

3.1 External Dependencies

The actual service is a Spring based application and has been built by using Spring Boot. The most important external dependencies are Jakarta Persistence backed by Hibernate and JAX-RS (Jakarta RESTful Web Services) backed by Jersey. Incoming and outgoing representations in JSON format will be mapped on the generic object model provided by Jakarta JSON Processing. The secret sharing algorithm and corresponding keystore engine is depending on the JCA provider Shamirs Keystore. XML encryption will be implemented drawing upon Apache Santuario. Bouncy Castle contributes some cryptographic algorithms or rather procedures related to PKCS #7 (Cryptographic Message Syntax) and PKCS #12 keystores.

The development database is a MariaDB instance. Since any database access will be handled by the Jakarta Persistence API, other databases might be supported as well at a later time.

The identy management will be given to an OpenId Connect provider, namely Keycloak.

Unit and integration tests are relying on JUnit 5 together with AssertJ.

3.2 Data Model

Participants of secret sharing shemes might have got shares of several keystores and a particular keystore has at least one owner but might have been shared between several participants. A slice consists of one or more shares. Different participants might have been given a different number of shares. A slice bundles up shares into one JSON file. Therefore, participants might have slices – bundled shares – of more than one keystore and vice versa access to a certain keystore has been splitted into different slices. Furthermore, the validity of slices expires when an active session of a keystore closes. Thereupon a new batch of slices will be created for this keystore and its participants. A Keystore entity might reference several sessions. One of these sessions – the most recently created – is the present session. Newly created sessions are in state PROVISIONED. Documents scheduled for encryption or digital sign-

ing can be submitted to provisioned sessions for review. As soon as a sufficient subset of slices comprising of password shares is available the provisioned session can be switched to state ACTIVE and all pending documents will be processed. Documents submitted to active sessions will be processed at once. The metadata and the document itself will be separately stored. See Figure 1 for details.

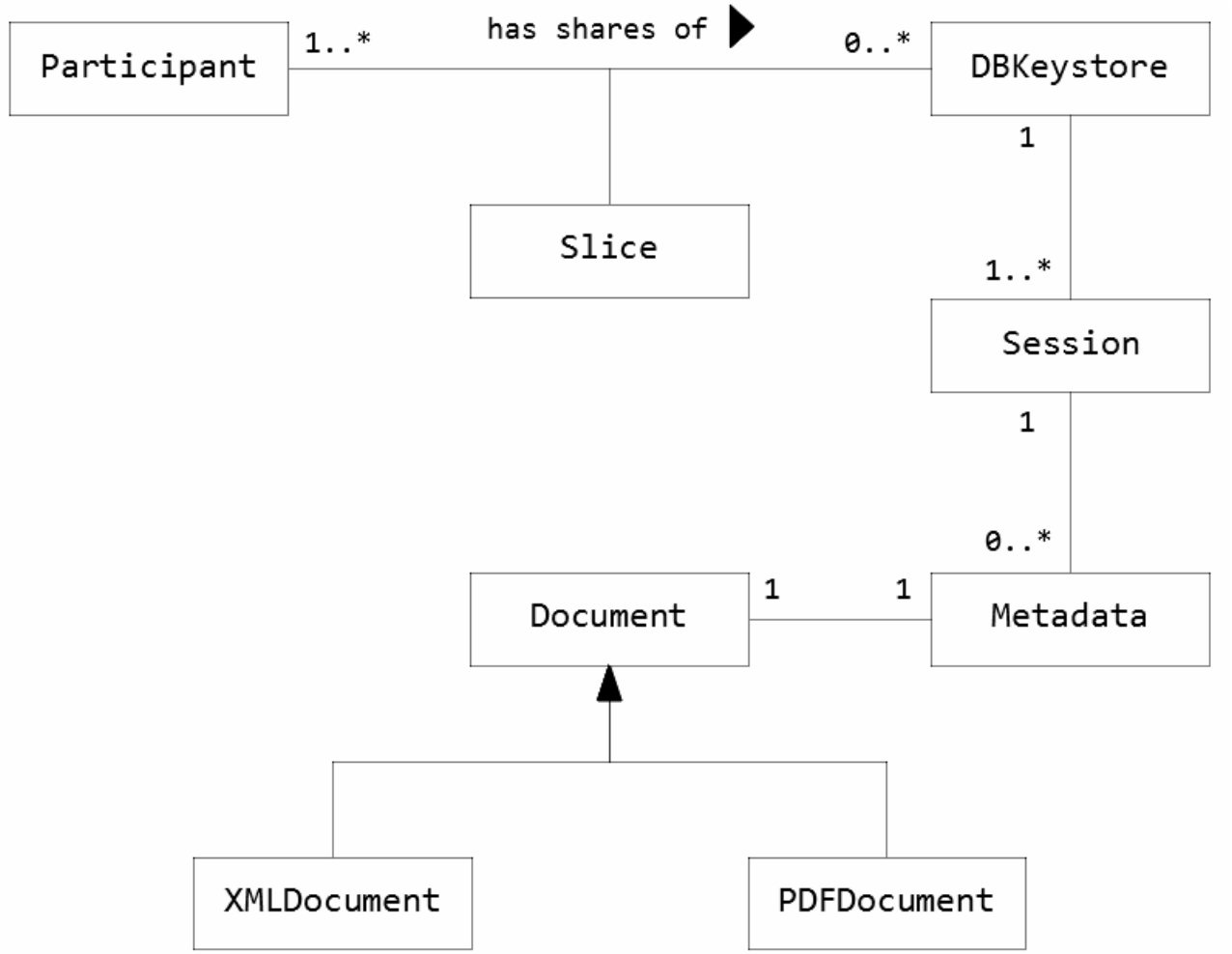

Figure 1: Data Model

3.3 Business Logic

The application tier has interfaces to the presentation tier and the back end. The classes responsible for processing the representations are called resource classes. These resources are primarily responsible to interprete the incoming instructions and delivering the appropriate presentations to the clients. In doing so, they must first sanitize the input provided as generic JSON object model. Illegal or invalid inputs result in HTTP 400 BAD REQUEST responses. POST requests usually lead to creating of domain objects, stitching them together and feeding the object graph to the database access routines. PATCH

or PUT requests need to query the database first to process the instructions for a resource. GET requests simply query the database and deliver the representations in JSON format. The domain classes provide a `toJson()`-method which produces such a representation to a given time.

All resource classes inherit from a abstract base class which provides some utilities useful for all of them, see Figure 2.

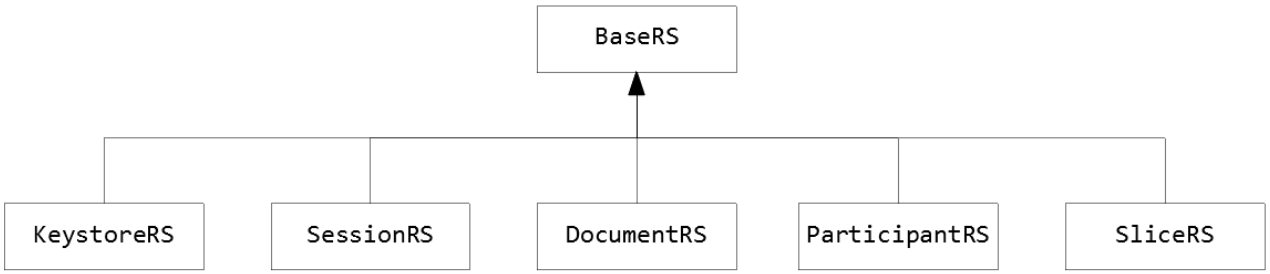

Figure 2: Resource classes

Some parts of the business logic are attached to the domain objects where convenient (rich domain model) leaving some glue code to the resource classes.

The database access routines are bundled into several services. Each of these services is spezialized in a particular domain class.

3.4 Sanitization of the User Input

Since the generic JSON object model provided by Jakarta JSON Processing is utilized to process the incoming JSON representations, any formally valid JSON input would potentially be mapped on some appropriate (nested) JSON value structure regardless of the expectations of the service. In order to prevent possibly obscure application errors downstream or even to prevent malicious attack attempts, the provided JSON input must be validated against the expectations of the service as soon as possible.

One way of doing so would be the use of JSON Schema which can be applied to describe the anticipated data format and subsequently for the validation of the submitted user input. The accompanying specifications JSON Schema Core and JSON Schema Validation are presently being drafted. Building upon JSON schema and Jakarta JSON processing there already exists Justify – an open source library hosted on GitHub and depoyed to Maven Central – which might be used for these purposes.

At the moment the prototype of the service uses a custom solution for the validation of the JSON input. Specialised (and nested) JsonValueConstraints are used to match instances of corresponding JSON

value types (String, Number, Object, …) against given Regular Expressions and optionally to validate certain value ranges.

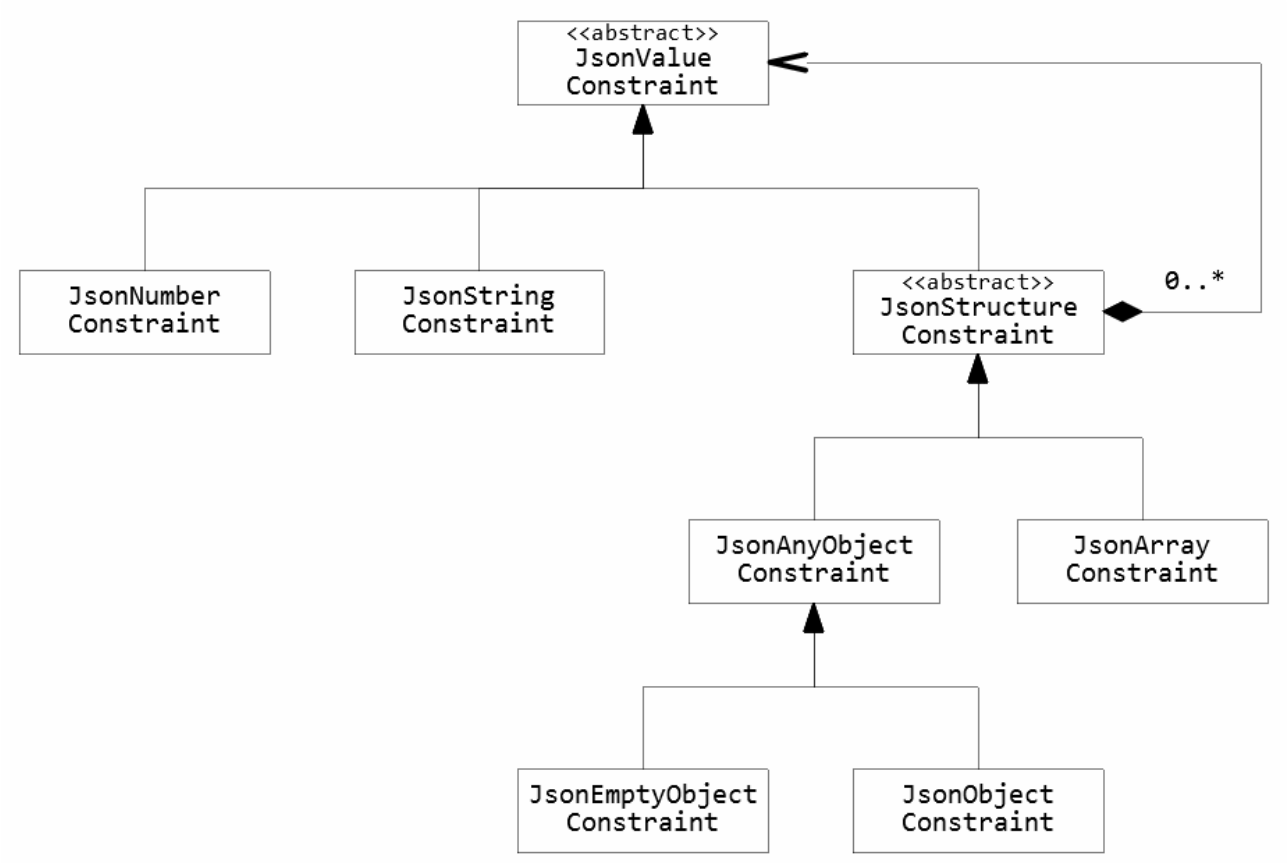

Figure 3: JsonValueConstraints

Consider the following constraint definitions

```
JsonStringConstraint jsonAliasConstraint = new JsonStringConstraint("[A-Za-z0-9-]{8,30}");
JsonStringConstraint jsonSecretKeyAlgoConstraint = new JsonStringConstraint("AES|HmacSHA512");
JsonNumberConstraint jsonSecretKeySizeConstraint = new JsonNumberConstraint("128|256|512");
JsonStringConstraint jsonSecretKeyTypeConstraint = new JsonStringConstraint("secret-key");
JsonObjectConstraint secretKeyInfoConstraint = new JsonObjectConstraint(
    Map.of("alias", jsonAliasConstraint, "algorithm", jsonSecretKeyAlgoConstraint, "keySize",
            jsonSecretKeySizeConstraint, "type", jsonSecretKeyTypeConstraint)
);
```

and the JsonObject `keyInfo` comprising

```
{
    "alias": "my-secret-key",
    "algorithm": "AES",
    "keySize": 256,
    "type": "secret-key"
}
```

Thereupon

```
assert secretKeyInfoConstraint.validate(keyInfo);
```

validates the given JsonObject.

3.5 Resources and Representations

The subsequent sections are somewhat related to the corresponding sections within chapter 2 Use Cases but with special attention to parameters and responses including the actual entities transmitted within the HTTP body. The examples shown have been sampled from different test runs. The randomly created UUIDs aren't always consistent. The prefix /shamir/v1 is omitted from all URIs.

3.5.1 Keystore Resource

3.5.1.1 Keystore Generation

POST /keystores

CONSUMES application/json
PRODUCES application/json

Parameter

Type	Name	Description	Schema
Body	Keystore Instructions	Instructions for generating Keystores	object, see Keystore Instructions

Keystore Instructions

Name	Required	Schema
shares	☑	integer
threshold	☑	integer
descriptiveName	☑	string
keyInfos	☑	array, see Secret Key Info and Private Key Info
sizes	☑	array

Secret Key Info

Name	Required	Schema
alias	☑	string
algorithm	☑	string
keysize	☑	integer
type	☑	string

Private Key Info

Name	Required	Schema
alias	☑	string
algorithm	☑	string
type	☑	string

x509	☑	object, see X509

X509

Name	Required	Schema
validity	☑	integer
commonName	☑	string
locality	☑	string
state	☑	string
country	☑	string

Response

HTTP Code	Status Info	Schema
201	Created	object, see Keystore Representation

Example

Supposing for instance that we want to generate a keystore comprising a secret key entry suitable for the AES algorithm and a private key entry suitable for the Elliptic Curve algorithm. Furthermore the access to the keystore should be partitioned into 12 shares whereas 4 of them are required to recover the secret. The 12 shares are given to 7 users. One user receives 4 shares, two user receive in each case 2 shares and four user are given in each case 1 share. That would be encoded by the subsequent JSON object:

```
POST /keystores
```

```
{
  "shares": 12,
  "threshold": 4,
  "descriptiveName": "my-posted-keystore",
  "keyinfos": [
    {
      "alias": "my-secret-key",
      "algorithm": "AES",
      "keySize": 256,
      "type": "secret-key"
    },
    {
      "alias": "donalds-private-ec-key",
      "algorithm": "EC",
      "type": "private-key",
      "x509": {
          "validity": 100,
          "commonName": "Donald Duck",
          "locality": "Entenhausen",
          "state": "Bayern",
          "country": "Deutschland"
      }
    }
  ],
  "sizes": [
    {
      "size": 4,
      "participant": "test-user-0"
```

```
    },
    {
      "size": 2,
      "participant": "test-user-1"
    },
    {
      "size": 2,
      "participant": "test-user-2"
    },
    {
      "size": 1,
      "participant": "test-user-3"
    },
    {
      "size": 1,
      "participant": "test-user-4"
    },
    {
      "size": 1,
      "participant": "test-user-5"
    },
    {
      "size": 1,
      "participant": "test-user-6"
    }
  ]
}
```


The server might answer with HTTP 201 CREATED and the subsequent message body:


```
{
  "id": "e54e1637-15a5-41dd-8a2c-7352f2932dbe",
  "descriptiveName": "my-posted-keystore",
  "currentPartitionId": "dcaf16a9-80b3-4217-aa0b-63425c36a29b",
  "shares": 12,
  "threshold": 4,
  "creationTime": "2021-06-04T14:57:54.7066307",
  "mofificationTime": "2021-06-04T14:57:54.7066307",
  "links": [
    {
      "rel": "self",
      "href": "/keystores/e54e1637-15a5-41dd-8a2c-7352f2932dbe",
      "type": [
        "GET"
      ]
    }
  ]
}
```


3.5.1.2 Query single Keystore

GET /keystores/<keystore_id>

PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	keystore_id	Keystore Id	string

Response

HTTP Code	Status Info	Schema

200	Ok	object, see Keystore Representation

Example

GET /keystores/e54e1637-15a5-41dd-8a2c-7352f2932dbe

This gives the full representation of the single keystore including the key entries, provided that the keystore is loadable. The service might answer with HTTP 200 OK and the subsequent message body:


```
{
  "id": "e54e1637-15a5-41dd-8a2c-7352f2932dbe",
  "descriptiveName": "my-posted-keystore",
  "currentPartitionId": "dcaf16a9-80b3-4217-aa0b-63425c36a29b",
  "shares": 12,
  "threshold": 4,
  "creationTime": "2021-06-04T14:57:54",
  "mofificationTime": "2021-06-04T14:57:54",
  "keyEntries": [
    {
      "alias": "my-secret-key",
      "localKeyID": "54:69:6d:65:20:31:36:32:32:38:31:31:34:37:32:38:31:30",
      "friendlyName": "my-secret-key"
    },
    {
      "alias": "donalds-private-ec-key",
      "localKeyID": "54:69:6d:65:20:31:36:32:32:38:31:31:34:37:33:33:37:33",
      "friendlyName": "donalds-private-ec-key"
    }
  ],
  "links": [
    {
      "rel": "self",
      "href": "/keystores/e54e1637-15a5-41dd-8a2c-7352f2932dbe",
      "type": [
          "GET"
      ]
    },
    {
      "rel": "sessions",
      "href": "/keystores/e54e1637-15a5-41dd-8a2c-7352f2932dbe/sessions",
      "type": [
          "GET"
      ]
    },
    {
      "rel": "currentSession",
      "href": "/keystores/e54e1637-15a5-41dd-8a2c-7352f2932dbe/sessions/e7f64346-0d72-4048-aea7-0263b33d68c4",
      "type": [
          "GET",
          "PUT"
      ]
    }
  ]
}
```


3.5.1.3 Query all Keystores

GET /keystores

PRODUCES application/json

Response

HTTP Code	Status Info	Schema
200	Ok	object, see List of Keystores

List of Keystores

Name	Schema
keystores	array, see Keystore Representation

Example

Note that every retrieved keystore is outlined by the light representation without key entries and only with self links. An example response is shown below:

```
GET /keystores
```

The service might answer with HTTP 200 OK and the subsequent message body:


```
{
  "keystores": [
    {
      "id": "5adab38c-702c-4559-8a5f-b792c14b9a43",
      "descriptiveName": "my-first-keystore",
      "currentPartitionId": "467b268d-1a7f-4f00-993c-672b82494822",
      "shares": 12,
      "threshold": 4,
      "creationTime": "2021-06-07T18:01:17",
      "mofificationTime": "2021-06-07T18:01:17",
      "links": [
        {
          "rel": "self",
          "href": "/keystores/5adab38c-702c-4559-8a5f-b792c14b9a43",
          "type": [
            "GET"
          ]
        }
      ]
    },
    {
      "id": "7660d95e-4962-4fd0-8d4d-306a66c57ac6",
      "descriptiveName": "my-posted-keystore",
      "currentPartitionId": "291133b2-44e9-4a63-addc-1fdc17d351da",
      "shares": 12,
      "threshold": 4,
      "creationTime": "2021-06-07T18:01:28",
      "mofificationTime": "2021-06-07T18:01:28",
      "links": [
        {
          "rel": "self",
          "href": "/keystores/7660d95e-4962-4fd0-8d4d-306a66c57ac6",
          "type": [
            "GET"
          ]
        }
      ]
    },
    {
      "id": "e509eaf0-3fec-4972-9e32-48e6911710f7",
      "descriptiveName": "the-idle-keystore",
      "currentPartitionId": "75821741-ffce-49a6-bc94-5a2b9239a65e",
      "shares": 12,
      "threshold": 4,
      "creationTime": "2021-06-07T18:01:17",
      "mofificationTime": "2021-06-07T18:01:25",
      "links": [
```

```
        {
          "rel": "self",
          "href": "/keystores/e509eaf0-3fec-4972-9e32-48e6911710f7",
          "type": [
            "GET"
          ]
        }
      ]
    }
  ]
}
```


3.5.1.4 Keystore Representation

Name	Full	Schema
id		string
descriptiveName		string
currentPartitionId		string
shares		integer
threshold		integer
creationTime		string
keyEntries	☑	array
modificationTime		string
links		array

3.5.2 Session Resource

3.5.2.1 Query all Sessions of a Keystore

GET /keystores/<keystore_id>/sessions

PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	keystore_id	Keystore Id	string

Response

HTTP Code	Status Info	Schema
200	Ok	object, see List of Sessions

List of Sessions

Name	Schema
sessions	array, see Session Representation

Example

```
GET /keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions
```

The service might answer with HTTP 200 OK and the subsequent message body:


```
{
  "sessions": [
    {
      "id": "8bff8ac6-fc31-40de-bd6a-eca4348171c5",
      "phase": "CLOSED",
      "idleTime": 5,
      "creationTime": "2021-08-16T15:02:40",
      "modificationTime": "2021-08-16T15:02:51",
      "expirationTime": "2021-08-16T15:02:49",
      "links": [
        {
          "rel": "self",
          "href": "/keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5",
          "type": [
            "GET"
          ]
        }
      ]
    },
    {
      "id": "f1bba567-ba79-43a5-ad8b-03b450a63064",
      "phase": "PROVISIONED",
      "idleTime": 0,
      "creationTime": "2021-08-16T15:02:51",
      "modificationTime": "2021-08-16T15:02:51",
      "expirationTime": "null",
      "links": [
        {
          "rel": "self",
          "href": "/keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions/f1bba567-ba79-43a5-ad8b-03b450a63064",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    }
  ]
}
```


3.5.2.2 Update Session

PATCH /keystores/<keystore_id>/sessions/<session_id>

CONSUMES application/json
PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	keystore_id	Keystore Id	string
Path	session_id	Session Id	string
Body	Session Instructions	Instructions for updating Sessions	object, see Session Representation

Session Instructions concern either the activation of provisioned sessions together with parameters denoting the maximal idle time (measured in seconds) or the explicit closure of active sessions. In the latter case the idle time is obviously not needed. The Session Id must be included within the PATCH document and will be matched against the Session Id on the path.

Response

Use Case	HTTP Code	Status Info	Schema
Activation	200	Ok	object, see Session Representation

Example 1

This is an example concerning the activation of provisioned sessions:

```
PATCH /keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5
```

```
{
  "id": "8bff8ac6-fc31-40de-bd6a-eca4348171c5",
  "phase": "ACTIVE",
  "idleTime": 5
}
```

The service might answer with HTTP 200 OK and the subsequent message body:

```
{
  "id": "8bff8ac6-fc31-40de-bd6a-eca4348171c5",
  "phase": "ACTIVE",
  "idleTime": 5,
  "creationTime": "2021-08-16T15:02:40",
  "modificationTime": "2021-08-16T15:02:44.6428432",
  "expirationTime": "2021-08-16T15:02:49.6428432",
  "links": [
    {
      "rel": "self",
      "href": "/keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5",
      "type": [
        "GET",
        "PATCH"
      ]
    }
  ]
}
```

Example 2

Subsequent snippet instructs the explicit and immediate closure of an activated session:

```
PATCH /keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions/f1bba567-ba79-43a5-ad8b-03b450a63064
```

```
{
  "id": "f1bba567-ba79-43a5-ad8b-03b450a63064",
  "phase": "CLOSED"
}
```

The service might answer with HTTP 200 OK and the subsequent message body:

```
{
  "id": "f1bba567-ba79-43a5-ad8b-03b450a63064",
  "phase": "CLOSED",
  "idleTime": 300,
  "creationTime": "2021-08-16T15:02:51",
```

```
  "modificationTime": "2021-08-16T15:02:53",
  "expirationTime": "2021-08-16T15:07:52",
  "links": [
    {
      "rel": "self",
      "href": "/keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions/f1bba567-ba79-43a5-ad8b-03b450a63064",
      "type": [
        "GET"
      ]
    }
  ]
}
```


3.5.2.3 Query single session

GET /keystores/<keystore_id>/sessions/<session_id>

PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	keystore_id	Keystore Id	string
Path	session_id	Session Id	string

Response

HTTP Code	Status Info	Schema
200	Ok	object, see Session Representation

Example

```
GET /keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5
```

The service might answer with HTTP 200 OK and the subsequent message body:


```
{
  "id": "8bff8ac6-fc31-40de-bd6a-eca4348171c5",
  "phase": "ACTIVE",
  "idleTime": 5,
  "creationTime": "2021-08-16T15:02:40",
  "modificationTime": "2021-08-16T15:02:44",
  "expirationTime": "2021-08-16T15:02:49",
  "links": [
    {
      "rel": "self",
      "href": "/keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5",
      "type": [
        "GET",
        "PATCH"
      ]
    },
    {
      "rel": "documents",
      "href": "/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5/documents",
      "type": [
        "GET",
        "POST"
      ]
    },
    {
```

```
      "rel": "keystore",
      "href": "/keystores/5adab38c-702c-4559-8a5f-b792c14b9a43",
      "type": [
        "GET"
      ]
    }
  ]
}
```


3.5.2.4 Session Representation

Name	Schema	Patchable
id	string	
phase	string	☑
idleTime	string	☑
creationTime	string	
modificationTime	string	
expirationTime	string	
links	array	

3.5.3 Document Resource

3.5.3.1 Process Document

POST /sessions/<session_id>/documents

CONSUMES application/xml, application/pdf
PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	session_id	Session Id	string
Query	action	The intended processing, e.g. signing.	string
Query	alias	Points to the key to be used.	string
Header	content-type	The media type	string
Header	doc-title	Part of the metadata	string
Body	document	The to be processed document	xml | pdf

Response

HTTP Code	Status Info	Schema
201	Created	object
202	Accepted	object

Example

A xml document will be submitted to a provisioned session for digital signing. That will usually result in a created but pending document.

```
POST /sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5/documents?action=SIGN&alias=test-ec-key

Content-Type:  application/xml
doc-title:     payment-order-1
```

The service might answer with HTTP 201 CREATED and the subsequent message body:

```
{
  "id": "a1396065-dc1e-4889-8eae-b8dbca3f54a6",
  "title": "payment-order-1",
  "state": "PENDING",
  "action": "SIGN",
  "alias": "test-ec-key",
  "mediaType": "application/xml",
  "creationTime": "2021-06-11T15:32:15",
  "modificationTime": "2021-06-11T15:32:15",
  "links": [
    {
      "rel": "self",
      "href": "/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5/metadata/a1396065-dc1e-4889-8eae-b8dbca3f54a6",
      "type": [
        "GET"
      ]
    }
  ]
}
```

3.5.3.2 Query all Metadata of a Session

GET /sessions/<session_id>/documents

PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	session_id	Session Id	string

Response

HTTP Code	Status Info	Schema
200	Ok	object, see List of Metadata

List of Metadata

Name	Schema
documents	array, see Metadata Representation

Example

```
GET /sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5/documents
```

The service might answer with HTTP 200 OK and the subsequent message body:

```
{
  "documents": [
```

```
    {
      "id": "8baf5021-c6b5-404e-a98b-f324efe4d13d",
      "title": "payment-order-1",
      "state": "PENDING",
      "action": "SIGN",
      "alias": "test-ec-key",
      "mediaType": "application/xml",
      "creationTime": "2021-06-14T14:57:22",
      "modificationTime": "2021-06-14T14:57:22",
      "links": [
        {
          "rel": "self",
          "href": "/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5/metadata/8baf5021-c6b5-404e-a98b-f324efe4d13d",
          "type": [
            "GET"
          ]
        }
      ]
    }
  ]
}
```


3.5.3.3 Query Metadata of a single Document

GET /sessions/<session_id>/metadata/<document_id>

PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	session_id	Session Id	string
Path	document_id	Document Id	string

Response

HTTP Code	Status Info	Schema
200	Ok	object, see Metadata Representation

Example

```
GET /sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5/metadata/69f30c54-3154-4e78-b2a2-c69cf827f9e1
```

The service might answer with HTTP 200 OK and the subsequent message body:


```
{
  "id": "69f30c54-3154-4e78-b2a2-c69cf827f9e1",
  "title": "payment-order-1",
  "state": "PENDING",
  "action": "SIGN",
  "alias": "test-ec-key",
  "mediaType": "application/xml",
  "creationTime": "2021-08-16T15:03:04",
  "modificationTime": "2021-08-16T15:03:04",
  "links": [
    {
      "rel": "self",
      "href": "/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5/metadata/69f30c54-3154-4e78-b2a2-c69cf827f9e1",
      "type": [
        "GET"
```

```
      ]
    },
    {
      "rel": "content",
      "href": "/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5/documents/69f30c54-3154-4e78-b2a2-c69cf827f9e1",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "session",
      "href": "/keystores/5adab38c-702c-4559-8a5f-b792c14b9a43/sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5",
      "type": [
        "GET",
        "PATCH"
      ]
    }
  ]
}
```


3.5.3.4 Retrieve the content of a Document

GET /sessions/<session_id>/documents/<document_id>

PRODUCES application/octet-stream

Parameter

Type	Name	Description	Schema
Path	session_id	Session Id	string
Path	document_id	Document Id	string

Response

HTTP Code	Status Info	Schema
200	Ok	binary data

Example

```
GET /sessions/8bff8ac6-fc31-40de-bd6a-eca4348171c5/documents/8baf5021-c6b5-404e-a98b-f324efe4d13d
```

The service might answer with HTTP 200 OK

```
Content-Type:          application/octet-stream
Content-Length:        1070
```

and subsequent with the binary data.

3.5.3.5 Metadata Representation

Name	Schema
id	string
title	string
state	string

action	string
validated	true | false
alias	string
mediaType	string
creationTime	string
modificationTime	string
links	array

3.5.4 Slice Resource

3.5.4.1 Query all Slices

GET /slices

PRODUCES application/json

Parameter

Type	Name	Description	Schema	Required
Query	keystoreId	References the keystore	string	☐
Query	participantId	References the participant	string	☐

Response

HTTP Code	Status Info	Schema
200	Ok	object, see List of Slices

List of Slices

Name	Schema
slices	array, see Slice Representation

Example

```
GET /slices?participantId=8844dd34-c836-4060-ba73-c6d86ad1275d
```

The service might answer with HTTP 200 OK and the subsequent message body:


```
{
  "slices": [
    {
      "id": "31fe5c5d-3936-4f87-9c9a-c1214b7a7ffa",
      "partitionId": "576f2709-e0b2-4a5f-af3a-c031597960c7",
      "state": "POSTED",
      "size": 4,
      "creationTime": "2021-08-05T14:34:42",
      "modificationTime": "2021-08-05T14:34:42",
      "links": [
        {
          "rel": "self",
          "href": "/slices/31fe5c5d-3936-4f87-9c9a-c1214b7a7ffa",
          "type": [
            "GET",
```

```
            "PATCH"
          ]
        }
      ]
    },
    {
      "id": "87d106c9-2c3b-436d-9452-6f8c83d3b575",
      "partitionId": "5d6f722a-a357-42af-9005-04df6c38c54b",
      "state": "POSTED",
      "size": 4,
      "creationTime": "2021-08-05T14:34:52",
      "modificationTime": "2021-08-05T14:34:54",
      "links": [
        {
          "rel": "self",
          "href": "/slices/87d106c9-2c3b-436d-9452-6f8c83d3b575",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    },
    ...
  ]
}
```


3.5.4.2 Query single Slice

GET /slices/<slice_id>

PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	slice_id	Session Id	string

Response

HTTP Code	Status Info	Schema
200	Ok	object, see Slice Representation

Example

```
GET /slices/9a83d398-35d6-4959-aea2-1c930a936b43
```

The service might answer with HTTP 200 OK and the subsequent message body:


```
{
  "id": "9a83d398-35d6-4959-aea2-1c930a936b43",
  "partitionId": "467b268d-1a7f-4f00-993c-672b82494822",
  "state": "POSTED",
  "size": 4,
  "share": {
    "PartitionId": "467b268d-1a7f-4f00-993c-672b82494822",
    "Prime": 315934022480595788690846756243705863449548047083238167423552631,
    "Threshold": 4,
    "SharePoints": [
      {
        "SharePoint": {
          "x": 289543227158298838536061163001438516449436725492754334150988824,
          "y": 242979906530246750409384701514018873007348027653621357534986072
        }
      },
      {
```

```
        "SharePoint": {
          "x": 21267524743004238432414232437029819750254730673904529031677785 8,
          "y": 12072149259155870965392447183534314671510226720200589264253901 2
        }
      },
      {
        "SharePoint": {
          "x": 21892885986683646440820045761265778765532082899827466770645746 9,
          "y": 16597567217337941201410608541771393716085257643539496130004441 1
        }
      },
      {
        "SharePoint": {
          "x": 76973558922572272375542145204157366176426869992273947883545760,
          "y": 38247841313326241782838646768879686262435157793137140327598408
        }
      }
    ]
  },
  "creationTime": "2021-08-09T16:20:29",
  "modificationTime": "2021-08-09T16:20:29",
  "links": [
    {
      "rel": "self",
      "href": "/slices/9a83d398-35d6-4959-aea2-1c930a936b43",
      "type": [
        "GET",
        "PATCH"
      ]
    },
    {
      "rel": "participant",
      "href": "/participant/8844dd34-c836-4060-ba73-c6d86ad1275d",
      "type": [
      ]
    },
    {
      "rel": "keystore",
      "href": "/keystore/5adab38c-702c-4559-8a5f-b792c14b9a43",
      "type": [
        "GET"
      ]
    }
  ]
}
```


3.5.4.3 Change status to FETCHED

PATCH /slices/<slice_id>

CONSUMES application/json
PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	slice_id	Slice Id	string
Body	Slice	Changes to be applied	object, see Slice Representation

Response

HTTP Code	Status Info	Schema
200	Ok	object, see Slice Representation

Example

After retrieving the shares by querying a single slice (and storing them locally), see Query single Slice, the status of the slice may be set to FETCHED. If the status is transitioned to FETCHED the share object must be empty. The patch instruction must include the Slice id. The id will be matched against the path parameter by the service.

```
PATCH /slices/d517092d-ccee-4b4b-8ebf-fcf5b3c11297
```

```
{
  "id": "d517092d-ccee-4b4b-8ebf-fcf5b3c11297",
  "state": "FETCHED",
  "share": {
  }
}
```

The service might answer with HTTP 200 OK and the subsequent message body:

```
{
  "id": "d517092d-ccee-4b4b-8ebf-fcf5b3c11297",
  "partitionId": "10f893c7-367a-4906-96b9-a4160203e00d",
  "state": "FETCHED",
  "size": 4,
  "share": {
  },
  "creationTime": "2021-08-10T18:46:33",
  "modificationTime": "2021-08-10T18:46:35.0324065",
  "links": [
    {
      "rel": "self",
      "href": "/slices/d517092d-ccee-4b4b-8ebf-fcf5b3c11297",
      "type": [
        "GET",
        "PATCH"
      ]
    },
    {
      "rel": "participant",
      "href": "/participant/8844dd34-c836-4060-ba73-c6d86ad1275d",
      "type": [
      ]
    },
    {
      "rel": "keystore",
      "href": "/keystore/5adab38c-702c-4559-8a5f-b792c14b9a43",
      "type": [
        "GET"
      ]
    }
  ]
}
```

3.5.4.4 Retransmit password shares

PATCH /slices/<slice_id>

CONSUMES application/json
PRODUCES application/json

Parameter

Type	Name	Description	Schema
Path	slice_id	Slice Id	string
Body	Slice	Changes to be applied	object, see Slice Representation

Response

HTTP Code	Status Info	Schema
200	Ok	object, see Slice Representation

Example

Assuming the status of the slice is FETCHED, the status of the slice can be transitioned to POSTED by re-transmitting the password shares:

```
PATCH /slices/d517092d-ccee-4b4b-8ebf-fcf5b3c11297
```


```
{
  "id": "d517092d-ccee-4b4b-8ebf-fcf5b3c11297",
  "state": "POSTED",
  "share": {
    "PartitionId": "10f893c7-367a-4906-96b9-a4160203e00d",
    "Prime": 268057808983735289074253175850223149204880931401349803494332136515893023624091 17,
    "Threshold": 4,
    "SharePoints": [
      {
        "SharePoint": {
          "x": 127745298238849042201162445878905937633663642025299931427285046633418147086407 82,
          "y": 240762640992560245766970683092294127383627495764434996821393252879139912347354 69
        }
      },
      {
        "SharePoint": {
          "x": 121156173670578606288697300259084165009501232109803953692541079897502365121297 72,
          "y": 374861082322139944088173142254184144306264588768874068022872677917250300160616 9
        }
      },
      {
        "SharePoint": {
          "x": 159858406646794823307151991617769535196240065741147130267276395415878513200344 27,
          "y": 965014193923924397340741907279005873536520435604689081880541150024144268795642 3
        }
      },
      {
        "SharePoint": {
          "x": 681123772807037120319768650118490065819361782250492029227714829334912398411218 7,
          "y": 200986247800913790852251202239243577812158772249947473654548095612216123030948 89
        }
      }
    ]
  }
}
```


The service might answer with HTTP 200 OK and the subsequent message body:


```
{
  "id": "d517092d-ccee-4b4b-8ebf-fcf5b3c11297",
  "partitionId": "10f893c7-367a-4906-96b9-a4160203e00d",
  "state": "POSTED",
  "size": 4,
  "share": {
    "PartitionId": "10f893c7-367a-4906-96b9-a4160203e00d",
    "Prime": 268057808983735289074253175850223149204880931401349803494332136515893023624091 17,
    "Threshold": 4,
    "SharePoints": [
      {
```

```
        "SharePoint": {
          "x": 127745298238849042201162445878905937633663642025299931427285046633418147086407 82,
          "y": 240762640992560245766970683092294127383627495764434996821393252879139912347354 69
        }
      },
      {
        "SharePoint": {
          "x": 121156173670578606288697300259084165009501232109803953692541079897502365121297 72,
          "y": 374861082322139944088173142254184144306264588768874068022872677917250300160616 9
        }
      },
      {
        "SharePoint": {
          "x": 159858406646794823307151991617769535196240065741147130267276395415878513200344 27,
          "y": 965014193923924397340741907279005873536520435604689081880541150024144268795642 3
        }
      },
      {
        "SharePoint": {
          "x": 681123772807037120319768650118490065819361782250492029227714829334912398411218 7,
          "y": 200986247800913790852251202239243577812158772249947473654548095612216123030948 89
        }
      }
    ]
  },
  "creationTime": "2021-08-10T18:46:33",
  "modificationTime": "2021-08-10T18:46:35.1145691",
  "links": [
    {
      "rel": "self",
      "href": "/slices/d517092d-ccee-4b4b-8ebf-fcf5b3c11297",
      "type": [
        "GET",
        "PATCH"
      ]
    },
    {
      "rel": "participant",
      "href": "/participant/8844dd34-c836-4060-ba73-c6d86ad1275d",
      "type": [
      ]
    },
    {
      "rel": "keystore",
      "href": "/keystore/5adab38c-702c-4559-8a5f-b792c14b9a43",
      "type": [
        "GET"
      ]
    }
  ]
}
```


3.5.4.5 Slice Representation

Name	Full	Schema	Patchable
id		string	
partitionId		string	
state		string	☑
size		integer	
share	☑	object	☑
creationTime		string	
modificationTime		string	

links		array	

3.6 Test Plan

3.6.1 Automatic Tests

These tests are written in the spirit of testdriven development. There is at least one test case for each use case. Often more than one are needed for each use case to evaluate the robustness of the service. At the same time these tests help to avoid regressions when implementing new features.

3.6.1.1 Unit Tests

These tests are executed by JUnit 5 via the Maven Surefire Plugin during the build and are dependent on the development database. Every test class will perform a setup of the database via JDBC prior to the execution of its test cases. The test cases concentrate on the correctness of the database access routines and therefore bypass the presentation layer. Database modifications done by the JPA routines are crosschecked by native SQL statements via JDBC. Test cases within the same test class aren't strictly isolated from each other and must take into account modifications of the database state done by previous test cases. Where necessary a test order is defined. Scheduled periodic application services are switched off during the test run but a one-time execution can be triggered on demand.

3.6.1.2 Integration Tests

These tests are defined by a separate console application. The actual test classes must be configured within a XML file. The test discovery feature of JUnit 5 is used to identify the set of enabled test cases. Before the execution of each test class the development database is initialized with a test scenario and the service is started automatically by the console application. After the execution of each test class the service is automatically shut down as well via an HTTP endpoint provided by Spring Boot Actuator. Test cases defined within the same test class aren't isolated from each other and must take database changes done by previously run test cases into account. If necessary a test order is defined. The test cases verify the presumed database state prior to their actual execution.

These tests are implemented against the presentation layer and the offered REST API. Additional services that might needed such as a OAuth2 token service, e.g. provided by Keycloak, are booted and initialized as well if required.

Scheduled periodic services are switched on and their impact must be taken into account by the test cases.

3.6.2 Manual Tests

The workflows described in the paragraphs below have been recorded during a hands-on session with a prototype of the service. At this stage integration with an OpenId Connect provider is out of scope. Access to the service is granted by Client-authenticated TLS handshake. Invocations without certificate will result in an HTTP 403 FORBIDDEN error. Invalid or expired certificates will lead to aborted connections. The server certificate of the prototype is ultimately self-signed. An accompanying certificate for checking the service id will be provided by the project site. The required client certificates can be issued on request.

The recorded manual tests have been executed on the command line with curl. curl is available on most Linux distributions, Windows 10 and MacOS out of the box. Other options like Postman or browser plugins like Rester might work as well but please note the remarks about issues regarding JSON libs and big integers.

You will need a tool for pretty printing the JSON output of the service if you use curl on the command line. I have tried jq and yajl. jq is very liberal when parsing and formatting numbers. jq parses all numbers as 64-bit floating point numbers which clashes with the internal representation of the shares. Those shares are currently using arbitrarily big integer numbers for point coordinates and even the slightest deviation would cause havoc when fetching and retransmitting shares. The JSON specification is oblivious to this and speaks only of numbers which might have a fraction (and exponent) or not. I am using Jakarta JSON Processing on both the client and the server side when executing automatic tests. Jakarta JSON Processing explicitly offers big integer support for the interpretation of JSON numbers, hence this problem has been slipped below the radar until I have run into problems when doing manual tests and piping the result to jq. yajl – implemented in C - seems to work well in that regard but is otherwise very limited. However, I expect that most Javascript libraries will run into issues when parsing and serializing the big integers. More libraries for pretty printing JSON are discussed by Format JSON data on Ubuntu.

The following sections are presenting a bunch of features of the service beginning with the intial creation of a keystore, the withdrawal of shares until the keystore is unloadable, the posting of documents for review, the activation of sessions, the processing of documents and the closure of sessions.

The subsequently shown curl commands are all assuming that the certificate to verify the server identity, the private key of the client and its accompanying certificate are located in a subdirectory `pki`.

3.6.2.1 Keystore Generation

We might want to create a keystore with an ECDSA key pair for digital signing and an AES symmetric key for the encryption of electronic documents. The test system provides seven test users (test-user-0, …, test-user-6) which can receive shares. We specify the generation of twelve shares and calling for a threshold of four shares. test-user-0 will be given four shares, test-user-1 and test-user-2 will receive both two shares and the remaining four shares will be distributed among test-user-3, test-user-4, test-user-5 and test-user-6. That means we have to post something like the following JSON document:

```
{
  "shares": 12,
  "threshold": 4,
  "descriptiveName": "my-test-keystore",
  "keyinfos": [
    {
      "alias": "my-secret-key",
      "algorithm": "AES",
      "keySize": 256,
      "type": "secret-key"
    },
    {
      "alias": "my-private-ec-key",
      "algorithm": "EC",
      "type": "private-key",
      "x509": {
        "validity": 100,
        "commonName": "Donald Duck",
        "locality": "Entenhausen",
        "state": "Bayern",
        "country": "Deutschland"
      }
    }
  ],
  "sizes": [
    {
      "size": 4,
      "participant": "test-user-0"
    },
    {
      "size": 2,
      "participant": "test-user-1"
    },
    {
      "size": 2,
      "participant": "test-user-2"
    },
    {
      "size": 1,
      "participant": "test-user-3"
    },
    {
      "size": 1,
      "participant": "test-user-4"
    },
    {
      "size": 1,
      "participant": "test-user-5"
    },
```

```
    {
      "size": 1,
      "participant": "test-user-6"
    }
  ]
}
```

Assuming that the above JSON file is located at `json/keystore-instructions.json` the curl command can be stated as follows:

```
$ curl --request POST --header "Content-Type: application/json" \
--header "Accept: application/json" --silent --data @json/keystore-instructions.json \
--cacert pki/root-ca.pem --cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores
```

which gives something like

```
{
  "id": "eca053b9-90fa-42a1-8483-1b5de4765673",
  "descriptiveName": "my-test-keystore",
  "currentPartitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
  "shares": 12,
  "threshold": 4,
  "creationTime": "2021-09-08T14:39:47.7054843",
  "mofificationTime": "2021-09-08T14:39:47.7054887",
  "links": [
    {
      "rel": "self",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673",
      "type": [
        "GET"
      ]
    }
  ]
}
```

Please note that you will get with virtual certainty other Ids. Following the self link with HTTP GET gives the full representation of the keystore:

```
$ KEYSTORE_ID=eca053b9-90fa-42a1-8483-1b5de4765673
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores/${KEYSTORE_ID}
```

```
{
  "id": "eca053b9-90fa-42a1-8483-1b5de4765673",
  "descriptiveName": "my-test-keystore",
  "currentPartitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
  "shares": 12,
  "threshold": 4,
  "creationTime": "2021-09-08T14:39:47",
  "mofificationTime": "2021-09-08T14:39:47",
  "keyEntries": [
    {
      "alias": "my-secret-key",
      "localKeyID": "54:69:6d:65:20:31:36:33:31:31:30:34:37:38:37:33:39:38",
      "friendlyName": "my-secret-key"
    },
    {
      "alias": "my-private-ec-key",
      "localKeyID": "54:69:6d:65:20:31:36:33:31:31:30:34:37:38:37:36:39:37",
      "friendlyName": "my-private-ec-key"
    }
  ],
  "links": [
```

```
    {
      "rel": "self",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "sessions",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "currentSession",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd",
      "type": [
        "GET",
        "PATCH"
      ]
    }
  ]
}
```


3.6.2.2 View the provisioned Session

Next, we request the sessions of this keystore. Since we have been creating the keystore just now, we expect only one (provisioned) session:

```
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores/${KEYSTORE_ID}/sessions
```


```
{
  "sessions": [
    {
      "id": "9d88c8be-5d7e-4dab-8520-51ef668a22cd",
      "phase": "PROVISIONED",
      "idleTime": 0,
      "creationTime": "2021-09-08T14:39:47",
      "modificationTime": "2021-09-08T14:39:47",
      "expirationTime": "null",
      "links": [
        {
          "rel": "self",
          "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    }
  ]
}
```


Note that the link of the provisioned session were already contained within in the links section of the full representation of the keystore. Now we are requesting the full representation of this session:

```
$ SESSION_ID=9d88c8be-5d7e-4dab-8520-51ef668a22cd
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores/${KEYSTORE_ID}/sessions/${SESSION_ID}
```

```
{
  "id": "9d88c8be-5d7e-4dab-8520-51ef668a22cd",
  "phase": "PROVISIONED",
  "idleTime": 0,
  "creationTime": "2021-09-08T14:39:47",
  "modificationTime": "2021-09-08T14:39:47",
  "expirationTime": "null",
  "links": [
    {
      "rel": "self",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd",
      "type": [
        "GET",
        "PATCH"
      ]
    },
    {
      "rel": "documents",
      "href": "/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd/documents",
      "type": [
        "GET",
        "POST"
      ]
    },
    {
      "rel": "keystore",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673",
      "type": [
        "GET"
      ]
    }
  ]
}
```


Now we can see where to post documents. Before we indeed post a document we might want to fetch some slices such that the keystore becomes unloadable.

3.6.2.3 Fetch Slices

First we need to request the slices for the keystore:

```
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/slices?keystoreId=${KEYSTORE_ID}
```


```
{
  "slices": [
    {
      "id": "223d6d95-882d-456b-a9ed-0ce6d8fa91e3",
      "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
      "state": "CREATED",
      "size": 1,
      "creationTime": "2021-09-08T14:39:47",
      "modificationTime": "2021-09-08T14:39:47",
      "links": [
        {
          "rel": "self",
          "href": "/slices/223d6d95-882d-456b-a9ed-0ce6d8fa91e3",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    },
    {
      "id": "35da371c-063e-449d-b2d5-db3efff50046",
```

```
    "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "state": "CREATED",
    "size": 1,
    "creationTime": "2021-09-08T14:39:47",
    "modificationTime": "2021-09-08T14:39:47",
    "links": [
      {
        "rel": "self",
        "href": "/slices/35da371c-063e-449d-b2d5-db3efff50046",
        "type": [
          "GET",
          "PATCH"
        ]
      }
    ]
  },
  {
    "id": "496f1814-2123-4c32-b3b3-d91167ca1853",
    "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "state": "CREATED",
    "size": 1,
    "creationTime": "2021-09-08T14:39:47",
    "modificationTime": "2021-09-08T14:39:47",
    "links": [
      {
        "rel": "self",
        "href": "/slices/496f1814-2123-4c32-b3b3-d91167ca1853",
        "type": [
          "GET",
          "PATCH"
        ]
      }
    ]
  },
  {
    "id": "997a6ac7-f318-45a6-9ca5-f95d69727181",
    "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "state": "CREATED",
    "size": 2,
    "creationTime": "2021-09-08T14:39:47",
    "modificationTime": "2021-09-08T14:39:47",
    "links": [
      {
        "rel": "self",
        "href": "/slices/997a6ac7-f318-45a6-9ca5-f95d69727181",
        "type": [
          "GET",
          "PATCH"
        ]
      }
    ]
  },
  {
    "id": "b8927aab-9d24-4915-b485-dfc42fab4cb6",
    "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "state": "CREATED",
    "size": 2,
    "creationTime": "2021-09-08T14:39:47",
    "modificationTime": "2021-09-08T14:39:47",
    "links": [
      {
        "rel": "self",
        "href": "/slices/b8927aab-9d24-4915-b485-dfc42fab4cb6",
        "type": [
          "GET",
          "PATCH"
        ]
      }
    ]
  },
  {
    "id": "c0602798-ff42-470a-b69a-57974c94b5ea",
```

```
      "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
      "state": "CREATED",
      "size": 4,
      "creationTime": "2021-09-08T14:39:47",
      "modificationTime": "2021-09-08T14:39:47",
      "links": [
        {
          "rel": "self",
          "href": "/slices/c0602798-ff42-470a-b69a-57974c94b5ea",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    },
    {
      "id": "f229b45b-780d-4953-bcf1-b46458a35cd6",
      "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
      "state": "CREATED",
      "size": 1,
      "creationTime": "2021-09-08T14:39:47",
      "modificationTime": "2021-09-08T14:39:47",
      "links": [
        {
          "rel": "self",
          "href": "/slices/f229b45b-780d-4953-bcf1-b46458a35cd6",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    }
  ]
}
```


Now that we have the Ids of the relevant Slices, we can request the full representation of a single Slice which gives the password shares. Since we have specified a threshold of four shares from altogether twelve shares we need to fetch at least nine shares to make the keystore unloadable, e.g. the slice with four shares, the two slices with two shares and one of the slices with one share. The following command requests the full representation of the Slice with four shares:

```
$ SLICE_ID=c0602798-ff42-470a-b69a-57974c94b5ea
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/slices/${SLICE_ID}
```


```
{
  "id": "c0602798-ff42-470a-b69a-57974c94b5ea",
  "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
  "state": "CREATED",
  "size": 4,
  "share": {
    "PartitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "Prime": 260701976915497503405144147025079973753752554615104557250534393564138723320407 09,
    "Threshold": 4,
    "SharePoints": [
      {
        "SharePoint": {
          "x": 702455815462894565328113099324812345942478619869949337775819925687898633989701 3,
          "y": 231615719641857672468529130265248419630938942336212927352243534367158100435670 18
        }
      },
      {
        "SharePoint": {
```

```
          "x": 129447511417575746468307693992798308971898350468506917384007658398352452690980008,
          "y": 412428685790405502489245374976585082395207589406724960139206186678276571330838 1
        }
      },
      {
        "SharePoint": {
          "x": 117195177362216432130314897662939125655583560339347990292647895906230728623856 11,
          "y": 981235306165521933606642199355944582354653049268336211904109621250019775384671 7
        }
      },
      {
        "SharePoint": {
          "x": 147255834605446611280254821295866358363933939601087611627157045060464180809573 47,
          "y": 325896916452254568049887820788372852967199740294738283950048627165604968158374 9
        }
      }
    ]
  },
  "creationTime": "2021-09-08T14:39:47",
  "modificationTime": "2021-09-08T14:39:47",
  "links": [
    {
      "rel": "self",
      "href": "/slices/c0602798-ff42-470a-b69a-57974c94b5ea",
      "type": [
        "GET",
        "PATCH"
      ]
    },
    {
      "rel": "participant",
      "href": "/participant/7a8a75d9-e4ac-4f88-9bbd-75d6582c4577",
      "type": [

      ]
    },
    {
      "rel": "keystore",
      "href": "/keystore/eca053b9-90fa-42a1-8483-1b5de4765673",
      "type": [
        "GET"
      ]
    }
  ]
}
```

If your tool gives you floating point numbers for the prime and the coordinate fields you must unfortunately work with the raw data coming over the wire. Now store this response locally and repeat the procedure with the other Slices until you have stored nine share points. Next, we will patch the Slices by deleting the shares on the server and transitioning the status to FETCHED. You have to prepare the following Patch document for each of your Slices:

```
{
  "id": "c0602798-ff42-470a-b69a-57974c94b5ea",
  "state": "FETCHED",
  "share": {
  }
}
```

Now you can set the status of each of the chosen Slices to FETCHED with the curl statement below assuming your Patch documents are located within subdirectory `json`:

```
$ SLICE_ID=c0602798-ff42-470a-b69a-57974c94b5ea
$ curl --request PATCH --header "Content-Type: application/json" \
```

```
--header "Accept: application/json" --silent --data @json/slice-fetched.json \
--cacert pki/root-ca.pem --cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/slices/${SLICE_ID}
```


```
{
  "id": "c0602798-ff42-470a-b69a-57974c94b5ea",
  "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
  "state": "FETCHED",
  "size": 4,
  "share": {},
  "creationTime": "2021-09-08T14:39:47",
  "modificationTime": "2021-09-08T14:57:31.5095269",
  "links": [
    {
      "rel": "self",
      "href": "/slices/c0602798-ff42-470a-b69a-57974c94b5ea",
      "type": [
        "GET",
        "PATCH"
      ]
    },
    {
      "rel": "participant",
      "href": "/participant/7a8a75d9-e4ac-4f88-9bbd-75d6582c4577",
      "type": []
    },
    {
      "rel": "keystore",
      "href": "/keystore/eca053b9-90fa-42a1-8483-1b5de4765673",
      "type": [
        "GET"
      ]
    }
  ]
}
```


We might request all of the Slices of the particular keystore again:

```
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/slices?keystoreId=${KEYSTORE_ID}
```


```
{
  "slices": [
    {
      "id": "223d6d95-882d-456b-a9ed-0ce6d8fa91e3",
      "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
      "state": "FETCHED",
      "size": 1,
      "creationTime": "2021-09-08T14:39:47",
      "modificationTime": "2021-09-08T15:01:50",
      "links": [
        {
          "rel": "self",
          "href": "/slices/223d6d95-882d-456b-a9ed-0ce6d8fa91e3",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    },
    {
      "id": "35da371c-063e-449d-b2d5-db3efff50046",
      "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
      "state": "CREATED",
      "size": 1,
      "creationTime": "2021-09-08T14:39:47",
      "modificationTime": "2021-09-08T14:39:47",
```

```
    "links": [
      {
        "rel": "self",
        "href": "/slices/35da371c-063e-449d-b2d5-db3efff50046",
        "type": [
          "GET",
          "PATCH"
        ]
      }
    ]
  },
  {
    "id": "496f1814-2123-4c32-b3b3-d91167ca1853",
    "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "state": "CREATED",
    "size": 1,
    "creationTime": "2021-09-08T14:39:47",
    "modificationTime": "2021-09-08T14:39:47",
    "links": [
      {
        "rel": "self",
        "href": "/slices/496f1814-2123-4c32-b3b3-d91167ca1853",
        "type": [
          "GET",
          "PATCH"
        ]
      }
    ]
  },
  {
    "id": "997a6ac7-f318-45a6-9ca5-f95d69727181",
    "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "state": "FETCHED",
    "size": 2,
    "creationTime": "2021-09-08T14:39:47",
    "modificationTime": "2021-09-08T15:00:07",
    "links": [
      {
        "rel": "self",
        "href": "/slices/997a6ac7-f318-45a6-9ca5-f95d69727181",
        "type": [
          "GET",
          "PATCH"
        ]
      }
    ]
  },
  {
    "id": "b8927aab-9d24-4915-b485-dfc42fab4cb6",
    "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "state": "FETCHED",
    "size": 2,
    "creationTime": "2021-09-08T14:39:47",
    "modificationTime": "2021-09-08T15:00:58",
    "links": [
      {
        "rel": "self",
        "href": "/slices/b8927aab-9d24-4915-b485-dfc42fab4cb6",
        "type": [
          "GET",
          "PATCH"
        ]
      }
    ]
  },
  {
    "id": "c0602798-ff42-470a-b69a-57974c94b5ea",
    "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "state": "FETCHED",
    "size": 4,
    "creationTime": "2021-09-08T14:39:47",
    "modificationTime": "2021-09-08T14:57:31",
```

```
      "links": [
        {
          "rel": "self",
          "href": "/slices/c0602798-ff42-470a-b69a-57974c94b5ea",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    },
    {
      "id": "f229b45b-780d-4953-bcf1-b46458a35cd6",
      "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
      "state": "CREATED",
      "size": 1,
      "creationTime": "2021-09-08T14:39:47",
      "modificationTime": "2021-09-08T14:39:47",
      "links": [
        {
          "rel": "self",
          "href": "/slices/f229b45b-780d-4953-bcf1-b46458a35cd6",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    }
  ]
}
```


Now we have only three share points left. Make sure that the keystore has become unloadable:

```
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores/${KEYSTORE_ID}
```


```
{
  "id": "eca053b9-90fa-42a1-8483-1b5de4765673",
  "descriptiveName": "my-test-keystore",
  "currentPartitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
  "shares": 12,
  "threshold": 4,
  "creationTime": "2021-09-08T14:39:47",
  "mofificationTime": "2021-09-08T14:39:47",
  "keyEntries": "unloadable",
  "links": [
    {
      "rel": "self",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "sessions",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "currentSession",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd",
      "type": [
        "GET",
        "PATCH"
      ]
    }
```

```
  ]
}
```

Next, try to activate the session with a Patch document like that below:

```
{
  "id": "9d88c8be-5d7e-4dab-8520-51ef668a22cd",
  "phase": "ACTIVE",
  "idleTime": 900
}
```

Assuming the Patch document located at `json/session-activation.json` you might use:

```
$ curl --request PATCH --header "Content-Type: application/json" \
--header "Accept: application/json" --silent --data @json/session-activation.json \
--cacert pki/root-ca.pem --cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores/${KEYSTORE_ID}/sessions/${SESSION_ID}
```

```
{
  "status": 400,
  "reason": "Bad Request",
  "message": "Cannot activate session for Keystore[id=eca053b9-90fa-42a1-8483-1b5de4765673].",
  "hint": "requirement failed: Too few sharepoints."
}
```

3.6.2.4 Post documents for review

Suppose we want to review and subsequently sign a digital payment order such like that one below:

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
    <timestamp>2021-09-06T14:30:44</timestamp>
    <amount currency="EUR">
        100.000.000,00
    </amount>
    <debtor>
        Donald Drump
    </debtor>
    <creditor>
        Teutsche Bank
    </creditor>
</payment>
```

Assuming the payment order located at `xml/payment-order.xml` we can schedule the XML document for digital signing by posting it (please note that you must put the URL between quotes otherwise curl doesn't process the parameters correctly) like that:

```
$ curl --request POST --header "Content-Type: application/xml" \
--header "Accept: application/json" --header "doc-title: payment-order" \
--silent --data-binary @xml/payment-order.xml \
--cacert pki/root-ca.pem --cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
"https://localhost:8443/shamir/v1/sessions/${SESSION_ID}/documents?action=SIGN&alias=my-private-ec-key"
```

```
{
  "id": "903cb911-0ab5-4281-add5-f92d25df844b",
  "title": "payment-order",
  "state": "PENDING",
  "action": "SIGN",
  "alias": "my-private-ec-key",
  "mediaType": "application/xml",
  "creationTime": "2021-09-08T15:08:55",
```

```
  "modificationTime": "2021-09-08T15:08:55",
  "links": [
    {
      "rel": "self",
      "href": "/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd/metadata/903cb911-0ab5-4281-add5-f92d25df844b",
      "type": [
        "GET"
      ]
    }
  ]
}
```


We may retrieve the full metadata with the following curl statement:

```
$ DOCUMENT_ID=903cb911-0ab5-4281-add5-f92d25df844b
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/sessions/${SESSION_ID}/metadata/${DOCUMENT_ID}
```


```
{
  "id": "903cb911-0ab5-4281-add5-f92d25df844b",
  "title": "payment-order",
  "state": "PENDING",
  "action": "SIGN",
  "alias": "my-private-ec-key",
  "mediaType": "application/xml",
  "creationTime": "2021-09-08T15:08:55",
  "modificationTime": "2021-09-08T15:08:55",
  "links": [
    {
      "rel": "self",
      "href": "/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd/metadata/903cb911-0ab5-4281-add5-f92d25df844b",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "content",
      "href": "/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd/documents/903cb911-0ab5-4281-add5-f92d25df844b",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "session",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd",
      "type": [
        "GET",
        "PATCH"
      ]
    }
  ]
}
```


Since the session is not activated the processing of the document is still pending. You might prove that by downloading it from the server:

```
$ curl --header "Accept: application/octet-stream" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
--output xml/payment-order-unsigned.xml \
https://localhost:8443/shamir/v1/sessions/${SESSION_ID}/documents/${DOCUMENT_ID}
$ cat xml/payment-order-unsigned.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<payment>
    <timestamp>2021-09-06T14:30:44</timestamp>
```

```
    <amount currency="EUR">
        100.000.000,00
    </amount>
    <debtor>
        Donald Drump
    </debtor>
    <creditor>
        Teutsche Bank
    </creditor>
</payment>
```

3.6.2.5 Retransmit shares

Since three shares are remaining on the server it suffices to retransmit a single share to reach the threshold. Prepare a JSON document like that below:

```
{
  "id": "223d6d95-882d-456b-a9ed-0ce6d8fa91e3",
  "state": "POSTED",
  "share": {
    "PartitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "Prime": 260701976915497503405144147025079973753752554615104557250534393564138723320407 09,
    "Threshold": 4,
    "SharePoints": [
      {
        "SharePoint": {
          "x": 690863955458811237716937156029872932035776945258749829107055786499141663411 1639,
          "y": 192044776911687030924643054662534117409938249671584532616346017965800896666633 67
        }
      }
    ]
  }
}
```

Assuming the patch document is located at `json/slice-posted.json` we can use the curl statement below:

```
$ SLICE_ID=223d6d95-882d-456b-a9ed-0ce6d8fa91e3
$ curl --request PATCH --header "Content-Type: application/json" \
--header "Accept: application/json" --silent --data @json/slice-posted.json \
--cacert pki/root-ca.pem --cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/slices/${SLICE_ID}
```

```
{
  "id": "223d6d95-882d-456b-a9ed-0ce6d8fa91e3",
  "partitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
  "state": "POSTED",
  "size": 1,
  "share": {
    "PartitionId": "a146fedc-0ae4-447b-8db8-c710cc72155d",
    "Prime": 260701976915497503405144147025079973753752554615104557250534393564138723320407 09,
    "Threshold": 4,
    "SharePoints": [
      {
        "SharePoint": {
          "x": 690863955458811237716937156029872932035776945258749829107055786499141663411 1639,
          "y": 192044776911687030924643054662534117409938249671584532616346017965800896666633 67
        }
      }
    ]
  },
  "creationTime": "2021-09-08T14:39:47",
```

```
  "modificationTime": "2021-09-08T15:15:28.2359195",
  "links": [
    {
      "rel": "self",
      "href": "/slices/223d6d95-882d-456b-a9ed-0ce6d8fa91e3",
      "type": [
        "GET",
        "PATCH"
      ]
    },
    {
      "rel": "participant",
      "href": "/participant/48ef6c98-0e04-49bc-9f7f-01f2cec3ccac",
      "type": [

      ]
    },
    {
      "rel": "keystore",
      "href": "/keystore/eca053b9-90fa-42a1-8483-1b5de4765673",
      "type": [
        "GET"
      ]
    }
  ]
}
```


Now try to activate the session again:

```
$ curl --request PATCH --header "Content-Type: application/json" \
--header "Accept: application/json" --silent --data @json/session-activation.json \
--cacert pki/root-ca.pem --cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores/${KEYSTORE_ID}/sessions/${SESSION_ID}
```


```
{
  "id": "9d88c8be-5d7e-4dab-8520-51ef668a22cd",
  "phase": "ACTIVE",
  "idleTime": 900,
  "creationTime": "2021-09-08T14:39:47",
  "modificationTime": "2021-09-08T15:17:14.0637243",
  "expirationTime": "2021-09-08T15:32:14.0637243",
  "links": [
    {
      "rel": "self",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd",
      "type": [
        "GET",
        "PATCH"
      ]
    }
  ]
}
```


This has worked. The document should be processed by now:

```
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/sessions/${SESSION_ID}/metadata/${DOCUMENT_ID}
```


```
{
  "id": "903cb911-0ab5-4281-add5-f92d25df844b",
  "title": "payment-order",
  "state": "PROCESSED",
  "action": "SIGN",
  "alias": "my-private-ec-key",
  "mediaType": "application/xml",
  "creationTime": "2021-09-08T15:08:55",
  "modificationTime": "2021-09-08T15:17:14",
```

```
  "links": [
    {
      "rel": "self",
      "href": "/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd/metadata/903cb911-0ab5-4281-add5-f92d25df844b",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "content",
      "href": "/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd/documents/903cb911-0ab5-4281-add5-f92d25df844b",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "session",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd",
      "type": [
        "GET",
        "PATCH"
      ]
    }
  ]
}
```


Now download the signed document:

```
$ curl --header "Accept: application/octet-stream" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
--output xml/payment-order-signed.xml \
https://localhost:8443/shamir/v1/sessions/${SESSION_ID}/documents/${DOCUMENT_ID}
$ cat xml/payment-order-signed.xml
```


```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<payment>
    <timestamp>2021-09-06T14:30:44</timestamp>
    <amount currency="EUR">
        100.000.000,00
    </amount>
    <debtor>
        Donald Drump
    </debtor>
    <creditor>
        Teutsche Bank
    </creditor>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
            <SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256"/>
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <DigestValue>ukNx9wzzzFxj5omYdRO3nrSNbqDOlp5X1Yg5xNIgL7I=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>ATtsIOhG3OVkPM/SLo8LrdNHIryJtemw173q4Fevt7N0/Ub4bumbinVzfe8r6+ly0ybYgBmI4xbH&#13;
            rkk8OOepgAoxAHfnzsHMo23SurAJhnYnCazRV0MhVyU+fSIQFqY/7lbd719eSnYdzbPWgr86UqZZ&#13;
            C6WeWF/a5qvGqO8prJEVejz7</SignatureValue>
    </Signature>
</payment>
```


The above XML document has been pretty printed after the fact for clarity – CAUTION: that makes the signature indeed invalid. Note that you also get escaped carriage returns (
) within the Base64 encoded sections after every block of 76 characters as specified by MIME.

3.6.2.6 Session Closure

Now prepare a patch document to close the session:

```
{
  "id": "9d88c8be-5d7e-4dab-8520-51ef668a22cd",
  "phase": "CLOSED"
}
```

Assuming the patch document at location `json/session-closure.json` the curl statement below can be used to close the session:

```
$ curl --request PATCH --header "Content-Type: application/json" \
--header "Accept: application/json" --silent --data @json/session-closure.json \
--cacert pki/root-ca.pem --cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores/${KEYSTORE_ID}/sessions/${SESSION_ID}
```

```
{
  "id": "9d88c8be-5d7e-4dab-8520-51ef668a22cd",
  "phase": "CLOSED",
  "idleTime": 900,
  "creationTime": "2021-09-08T14:39:47",
  "modificationTime": "2021-09-08T15:22:35",
  "expirationTime": "2021-09-08T15:32:14",
  "links": [
    {
      "rel": "self",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd",
      "type": [
        "GET"
      ]
    }
  ]
}
```

Now request the keystore view again:

```
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores/${KEYSTORE_ID}
```

```
{
  "id": "eca053b9-90fa-42a1-8483-1b5de4765673",
  "descriptiveName": "my-test-keystore",
  "currentPartitionId": "3cf39f07-385c-4e26-a49c-b25268e2b3cd",
  "shares": 12,
  "threshold": 4,
  "creationTime": "2021-09-08T14:39:47",
  "mofificationTime": "2021-09-08T15:22:35",
  "keyEntries": [
    {
      "alias": "my-secret-key",
      "localKeyID": "54:69:6d:65:20:31:36:33:31:31:30:37:33:35:35:39:32:33",
      "friendlyName": "my-secret-key"
    },
    {
      "alias": "my-private-ec-key",
      "localKeyID": "54:69:6d:65:20:31:36:33:31:31:30:37:33:35:35:39:34:30",
      "friendlyName": "my-private-ec-key"
    }
  ],
  "links": [
    {
      "rel": "self",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673",
```

```
      "type": [
        "GET"
      ]
    },
    {
      "rel": "sessions",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions",
      "type": [
        "GET"
      ]
    },
    {
      "rel": "currentSession",
      "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/737be9f9-48f9-47c2-9469-9af284e27f2e",
      "type": [
        "GET",
        "PATCH"
      ]
    }
  ]
}
```

Note that we have a different current session. Indeed the old session has been closed and a new one has been provisioned:

```
$ curl --header "Accept: application/json" --silent --cacert pki/root-ca.pem \
--cert pki/test-user-0-id.pem --key pki/test-user-0-id-pk.pem \
https://localhost:8443/shamir/v1/keystores/${KEYSTORE_ID}/sessions
```

```
{
  "sessions": [
    {
      "id": "737be9f9-48f9-47c2-9469-9af284e27f2e",
      "phase": "PROVISIONED",
      "idleTime": 0,
      "creationTime": "2021-09-08T15:22:35",
      "modificationTime": "2021-09-08T15:22:35",
      "expirationTime": "null",
      "links": [
        {
          "rel": "self",
          "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/737be9f9-48f9-47c2-9469-9af284e27f2e",
          "type": [
            "GET",
            "PATCH"
          ]
        }
      ]
    },
    {
      "id": "9d88c8be-5d7e-4dab-8520-51ef668a22cd",
      "phase": "CLOSED",
      "idleTime": 900,
      "creationTime": "2021-09-08T14:39:47",
      "modificationTime": "2021-09-08T15:22:35",
      "expirationTime": "2021-09-08T15:32:14",
      "links": [
        {
          "rel": "self",
          "href": "/keystores/eca053b9-90fa-42a1-8483-1b5de4765673/sessions/9d88c8be-5d7e-4dab-8520-51ef668a22cd",
          "type": [
            "GET"
          ]
        }
      ]
    }
  ]
}
```

3.7 Deployment

Deploying both the service and its database as containers on the same linux host connected by a bridge network is the favoured deployment option, see Figure 4. The development environment can already be configured to use either a native MariaDB 10 installation or a docker container running the latest MariaDB image (presently mariadb:10.6.5-focal). Shell scripts are provided which setup and start the database instance by mounting the data directionary from the host system. Certain Maven build profiles create images for the service itself and for the integration test client.

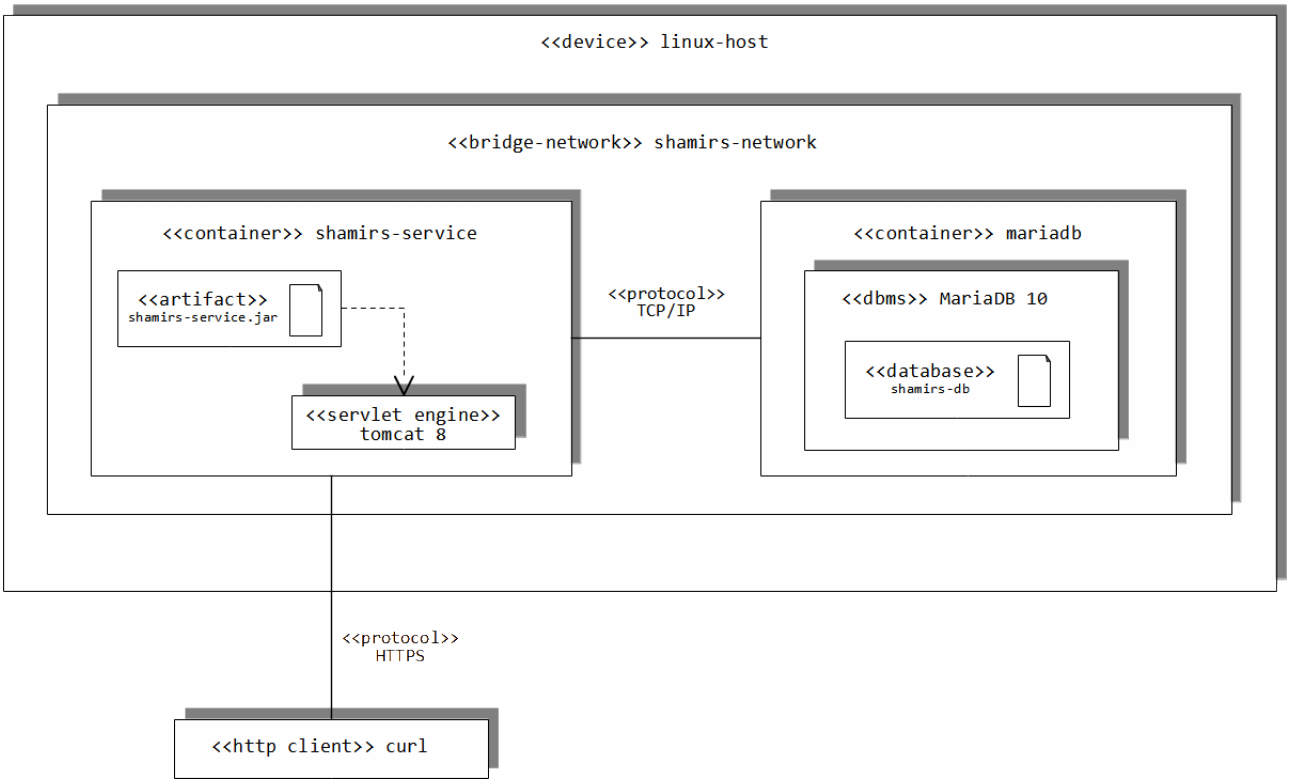

Figure 4: Production System

The installation can optionally be verified with the test client connecting to the bridge network as well, see Figure 5.

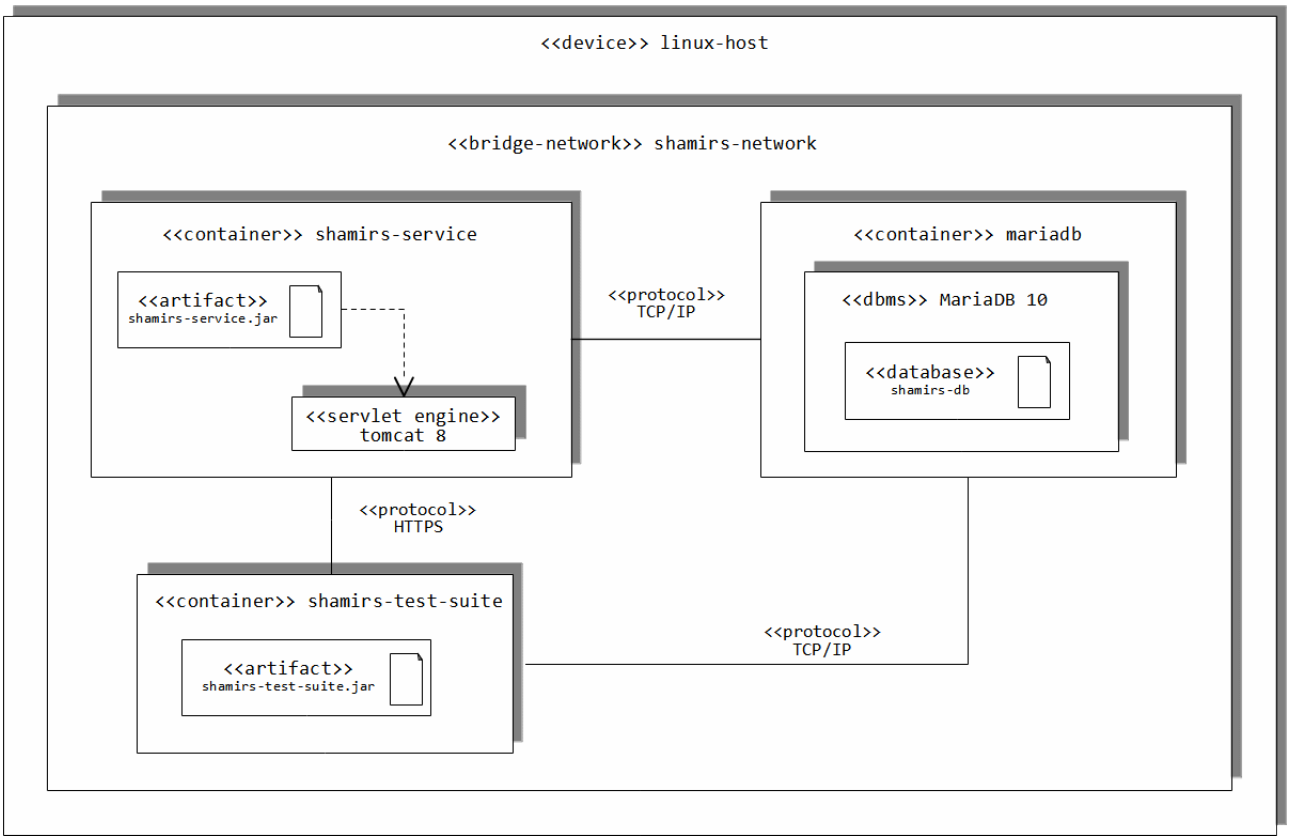

Figure 5: Integration Test System

4 Security Considerations

5 Privacy