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

Contents

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 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.

HTTP
: Hypertext Transfer Protocol.

JSON
: A lightweight data-interchange format.

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

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.

URI
: Uniform Resource Identifier.

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 according to submitted instructions and will never leave the server. 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 it must be assigned a session which in turn can only be created 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 a new password 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 is 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.

1.4 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

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

https://<host>/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

https://<host>/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

https://<host>/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 separete 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 PUTting the appropriating session instructions as JSON object.

URI

https://<host>/shamir/v1/keystores/<keystore_id>/sessions/<session_id>

Method

PUT

Preconditions

(1) The present session belonging to the given keystore has the state PROVIDED.
(2) A 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.

2.6 Session Closure

2.6.1 Automatic Closure

During the session activation a maximum idle period had been transmitted. A concurrent thread 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 PROVIDED.
(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.

2.6.2 On Demand

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

URI

https://<host>/shamir/v1/keystores/<keystore_id>/sessions/<session_id>

Method

PUT

Preconditions

(1) The actor PUTting 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 PROVIDED.

(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 server has responded with `HTTP 204 No Content`.

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

https://<host>/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 Digital Signature and Encryption

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 additional 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. Encrypted, decrypted or digitally signed document will be stored on the server. Depending on the size of the transferred document the service might decide to process the request asynchronously.

URI

https://<host>/shamir/v1/sessions/<session_id>?alias=<alias>&action=<action>

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 has been stored if applicable.

(2) The server has responded either with `HTTP 201 Created` or `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.