D3.9

Design of libraries for HR security and privacy services - V1

ABSTRACT

This deliverable provides the first version of the design of security and privacy services, in particular the components and the functional primitives regarding identity management and consent management. The content of this deliverable derived from the InteropEHRate deliverables D3.3 - Specification of remote and D2D IDM mechanisms for HRs Interoperability - V1 **[D3.3]** and D3.7 - Specification of consent management and decentralized authorization mechanisms for HR Exchange - V1 [**D3.7**] and depict the major features and principles of designing the security libraries addressing identity management, consent management and crypto primitives.

Delivery Date	18th October 2019
Work Package	WP3
Task	T3.4
Dissemination Level	Public
Type of Deliverable	Report
Lead partner	UBIT

InteropEHRate project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826106.

This document has been produced in the context of the InteropEHRate Project which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826106. All information provided in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose.

CONTRIBUTORS

	Name	Partner
Contributors	Entso Veliou, Sofianna Menesidou, Dimitris Papamartzivanos	UBIT
	Salima Houta, Marcel Klötgen	FRAU
Reviewers	Salima Houta, Marcel Klötgen	FRAU
	Chrysostomos Symvoulidis	BYTE

LOGTABLE

Version	Date	Change	Author	Partner
0.1	2019-09-02	First draft of ToC	Sofianna Menesidou	UBIT
0.2	2019-09-17	Internal review	Salima Houta, Marcel Klötgen	FRAU
0.3	2019-09-18	Introduction	Sofianna Menesidou	UBIT
0.4	2019-09-19	Mapping with the user requirements	Sofianna Menesidou	UBIT
0.5	2019-09-22	HR Security and Privacy Fundamentals	Entso Veliou	UBIT
0.6	2019-10-04	Mapping with the user requirements	Sofianna Menesidou	UBIT
0.7	2019-10-07	HR security and privacy service library (d2d)	Entso Veliou, Sofianna Menesidou	UBIT
0.8	2019-10-08	HR security and privacy service library (r2d)	Entso Veliou, Sofianna Menesidou	UBIT
0.9	2019-10-09	HR security and privacy service library (d2d) and hr security and privacy service library (r2d)	Entso Veliou, Sofianna Menesidou, Dimitris Papamartzivanos	UBIT
1.0	2019-10-10	Conclusions	Sofianna Menesidou	UBIT
1.1	2019-10-14	Review	Salima Houta	FRAU

1.2	2019-10-14	Review	Chrysostomos Symvoulidis	BYTE
1.3	2019-10-17	Quality check	Argyro Mavrogiorgou	UPRC
1.4	2019-10-17	Final check	Laura Pucci	ENG
1.5	2019-10-18	Addressed final comments	Sofianna Menesidou	UBIT
vFinal	2019-10-18	Final version for submission	Laura Pucci	ENG

ACRONYMS

Acronym	Term and definition
CA	Certificate Authority
CEF	Connecting Europe Facility
CSR	Certificate Signing Request
D2D	Device to Device protocol
DS	Digital Signature
DSA	Digital Signature Algorithm
eHDSI	e-Health Digital Service Infrastructure
EHR	Electronic Health Record
eID	electronic identification
epSOS	european patient Smart Open Services project
eTS	Electronic Trust Services
HCP	Healthcare Professional
HR	Health Record
IDP	Identity Provider
IoT	Internet of Things
JCA	Java Cryptography Architecture
JKS	Java Key Store
JWE	JSON Web Encryption
JWS	JSON Web Signature
JWT	JSON Web Token
OS	Operating System
M-D2D-SM	Mobile Device to Device Security Management
M-R2D-SM	Mobile Remote to Device Security Management
MAC	Message Authentication Code

NCP	National Contact Point
PKC	Public Key Cryptography
PKI	Public Key Infrastructure
T-D2D-SM	Terminal Device to Device Security Management
TA	Trusted Application
TEE	Trusted Execution Environment
2FA	Two-Factor Authentication

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Scope of the document

The main goal of this document is to describe the initial version of the design of the security libraries offered by the InteropEHRate Framework as a reference implementation of the HR security and privacy services. The current document outlines the most important initial design features addressing identity management, consent management based on crypto primitives. At this stage of project implementation, the deliverable aims at depicting the major features and principles of designing the security libraries. A top-down approach was followed in order to identify the internal interactions starting from the D2D and R2D protocols.

In more detail, this deliverable describes two security libraries for D2D (M-D2D-SM and T-D2D-SM) and one security library for R2D (M-R2D-SM) including the related external components. All security libraries are Java-based in order to provide identity management, authentication and consent management. In general, the security libraries invoked by the S-EHR App, the HCP Web App, the D2D library and the R2D library. Similarly to other reports of the InteropEHRate project, this document presents just a first draft of the design of the libraries offered by the InteropEHRate Framework as a reference implementation of HR security and privacy services.

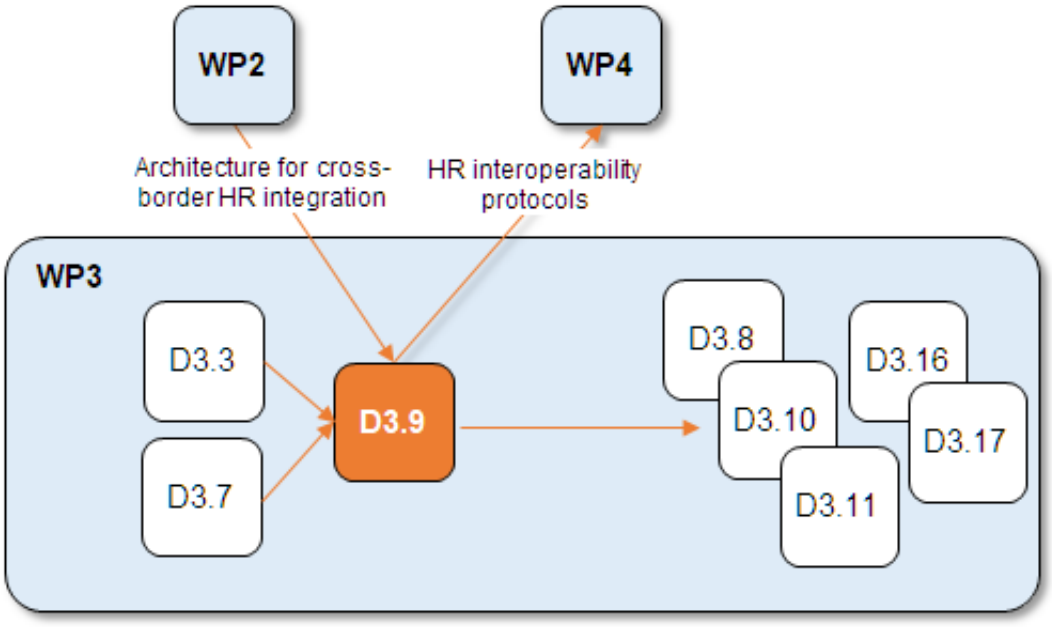

Figure 1 - Relation with other deliverables

1.2. Intended audience

The document is intended to security engineers, developers, architects, and all the InteropEHRate project participants and partners interested to have an overview of how InteropEHRate will support HR security and privacy services. These services will be described as libraries for mobile and web application developers who desire to exploit and reuse the security functionalities offered by the InteropEHRate framework.

1.3. Structure of the document

The current document is organized in the following Sections:

Section 1 (the current section) introduces the overall concept of the document, defining its scope, intended audience, and relation to the other project tasks and reports.

Section 2 describes the mapping between the security libraries and the user requirements introduced in the Architecture of the InteropEHRate project.

Section 3 focuses on describing and explaining some key concepts of security, which will be used in InteropEHRate framework.

Section 4 focuses on the design of the security libraries of the D2D case including the identity management, consent management and cryptographic primitives.

Section 5 focuses on the design of the security libraries of the R2D case including the identity management, consent management and cryptographic primitives.

Section 6 concludes the document, including future developments and updates for the two security libraries.

1.4. Updates with respect to previous version (if any)

This is the first version of the deliverable. Not Applicable.

2. MAPPING WITH THE USER REQUIREMENTS

This section describes the mapping between the designed libraries for HR security and privacy services and the user requirements. The user requirements have already identified in the Architecture of the InteropEHRate project. Table 1 below presents all the security-related user requirements and the corresponding implementation API provided by the security libraries to successfully satisfy the requirement. Next sections will provide more details on the implementation aspects.

The security implementations targeting the Citizen as a main actor is for the M-D2D-SM and M-R2D-SM components, while the security implementations targeting the HCP as a main actor is for the T-D2D-SM components.

#	User Requirement	Main Actor	SW Application	Implementation
1	Enabling of HCP identification from HCP app	HCP	HCP App	fetchCertificate, createPayload, signPayload
2	Enabling of healthcare organization identification from HCP app	HCP	HCP App	fetchCertificate, createPayload, signPayload
3	Consent to S-EHR data management	Citizen	S-EHR Mobile App	verifyAPPCConsent, signAPPCConsent
4	Enabling of Citizen identification from S-EHR	Citizen	S-EHR Mobile App	fetchCertificate, fetchHCPCertificate, verifySignature
5	D2D authorization to download and upload S-EHR data from HCP App	HCP	HCP App	getAuthenticationMeans, getAuthattributes, get2FAmeans, authenticate, bindUserWith2FA, authenticate2FA
6	Consent to store Citizen's data	HCP	HCP App	generateAPPCConsent, signAPPCConsent, verifyAPPCConsent

7	Data provenance tracking	Data user	S-EHR Mobile & HCP App	Focus on the next year
8	Integrity of medical information	Data user	S-EHR Mobile & HCP App	Focus on the next year
9	Confidentiality of medical information	Data user	S-EHR Mobile & HCP App	Focus on the next year

Table 1 - User Requirements and Security Libraries Mapping

3. HR SECURITY AND PRIVACY FUNDAMENTALS

This section presents the terminology and all the main security aspects of the InteropEHRate project. This section is necessary in order to justify the security-oriented solutions that InteropEHRate will provide through the security libraries.

3.1. Cross-border health interoperability - epSOS/eHDSI

Smart Open Services for European Patients (epSOS) is the main European electronic Health (eHealth) interoperability project co-funded by the European Commission and the partners [**epSOS2014**]. Results of epSOS project have been used in its successor project called eHealth Digital Service Infrastructure (eHDSI or eHealth DSI). eHDSI is focused on health data exchange and implemented by the Commission and the Member States through the Connecting Europe Facility (CEF) Programme. The eHDSI connects eHealth national contact points (NCP) allowing them to exchange two sets of health data: patient summaries and ePrescriptions.

Despite InteropEHRate and epSOS/eHDSI projects are both focused on cross border health data exchange, the context in which they operate is very different, especially for what concerns authentication. In epSOS there is no authentication mechanism for the citizen, there is no app given to the citizen and the only user that performs an electronic authentication is the HCP, but using proprietary authentication mechanism provided by his country. In the context of InteropEHRate, we will further extend the ability for a citizen to be able to authenticated and exchange his health data.

3.2. eIDAS Electronic Identification and Trust Services

The Regulation (EU) N°910/2014 on electronic identification and trust services for electronic transactions in the internal market (eIDAS Regulation) adopted by the co-legislators on 23 July 2014 is a milestone to provide a predictable regulatory environment to enable secure and seamless electronic interactions between businesses, citizens and public authorities [**eIDAS2014**].

The eIDAS Regulation:

1. Ensures that people and businesses can use their own national electronic identification schemes (eIDs) to access public services in other EU eID are available.
2. Creates a European internal market for eTS - namely electronic signatures, electronic seals, time stamp, electronic delivery service and website authentication - by ensuring that they will work across borders and have the same legal status as traditional paper based processes. Only by providing certainty on the legal validity of all these services, businesses and citizens will use the digital interactions as their natural way of interaction [**eIDAS2014**].

3.3. Public Key Infrastructure (PKI)

The Public key infrastructure (PKI) is the set of hardware, software, policies, processes, and procedures required to create, manage, distribute, use, store, and revoke digital certificates and public-keys [PKI]. The foundation of PKI is Public Key Cryptography (PKC). PKC, or asymmetric cryptography, is a cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. In PKC, the two related keys, work together to provide encryption/decryption and signing/verification functionalities. The public key — as the name suggests — is publicly available to anyone, while the private key should never be shared.

The PKI is required to deliver the public keys to existing systems or users securely. The public key is exchanged digitally in the form of digital certificates having a certain period of validity. The most known standard defining the format of a certificate is the X.509, while the entity that issues a digital certificate is the Certificate Authority (CA).

The Root CA is always a self-signed certificate. The root certificate, often called a trusted root, is at the centre of the trust model that undergirds PKI. Every device includes something called a root store. A root store is a collection of pre-downloaded root certificates (and their public keys) that live on the device itself. Generally, the device will use whatever root store is native to its Operating System (OS), otherwise it might use a third-party root store via an app like a web browser [**THESSLSTORE2019**].

In InteropEHRate, we follow the hierarchical approach and under the root CA we will create two more singing certificate sub-authorities, namely the HCP's CA and Citizen's CA. In this way, the root CA is protected Sub CAs. The public and private keys which will be issued by each sub CA will be stored to the acquirer's TEE (Trusted Execution Environment). The TEE is a secure area of the main processor of a connected device which ensures sensitive data is stored, processed and protected in an isolated and trusted environment. The TEE's ability to offer safe execution of authorized security software, known as 'trusted applications' (TAs), enables it to provide end-to-end security by protecting the execution of authenticated code, confidentiality, authenticity, privacy, system integrity and data access rights [**GLOBALPLATFORM2018**]. A typical example of hierarchy of trust depicted in Figure 2 below. More information on how the certificates are created is provided in Appendix A.

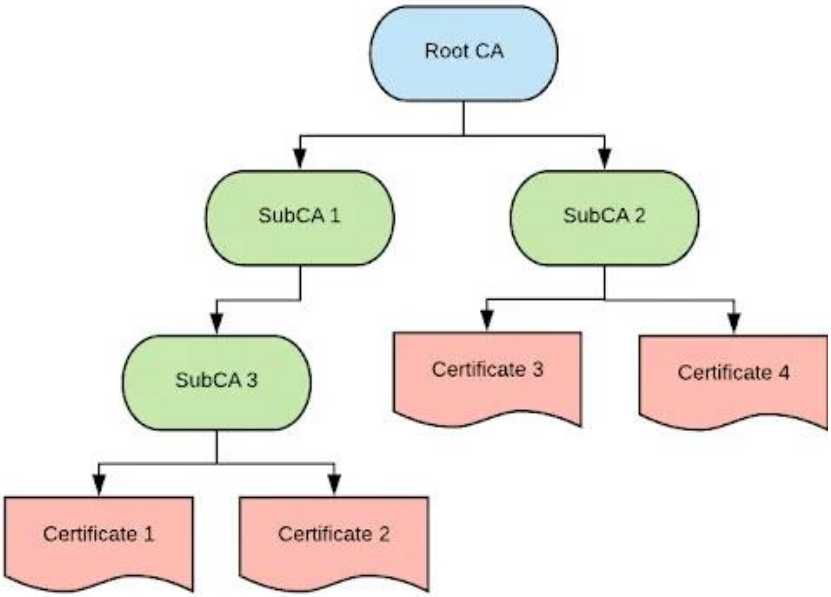

Figure 2 - Hierarchy of Trust

3.4. EJBCA

EJBCA is a free software public key infrastructure certificate authority software package maintained and sponsored by the Swedish for-profit company PrimeKey Solutions AB, which holds the copyright to most of

the codebase [**EJBCA2019**]. EJBCA offers a multipurpose PKI software that supports multiple CAs and levels of CAs to enable one to build a complete infrastructure (or several) for multiple use cases within one instance of the software. EJBCA enables multiple integration and automation possibilities and issues certificates to persons, infrastructure components and IoT (Internet of Things) devices. In addition, EJBCA is flexible, scalable and secure and is installed at numerous eIDAS applications. In the context of InteropEHRate, EJBCA will be used for the non-qualified certificates regarding the Variant 1 introduced in deliverable D3.3 [**[D3.3]**].

3.5. Digital Signature (DS)

The term digital signature is used to refer to a category of e-signatures which are created using the PKC. The terminology is often confusing, and the EU eIDAS Regulation has used terms such as “advanced electronic signatures” and “qualified electronic signatures” in an effort to be technology-neutral. However, practically the only way to implement them is to use digital signature based on PKI [**DS2019**]. The process of digital signing requires that the signature generated by both the fixed message and the private key can then be authenticated by its accompanied public key. At the time of verification the signer’s public key is used to unwrap the digital signature code and compare it with the document to ensure a match. Figure 3 below presents the procedure to sign a document:

- Signing — This algorithm produces a signature upon receiving a private key and the message that is being signed.
- Verification — This algorithm checks for the authenticity of the message by verifying it along with the signature and the public key.

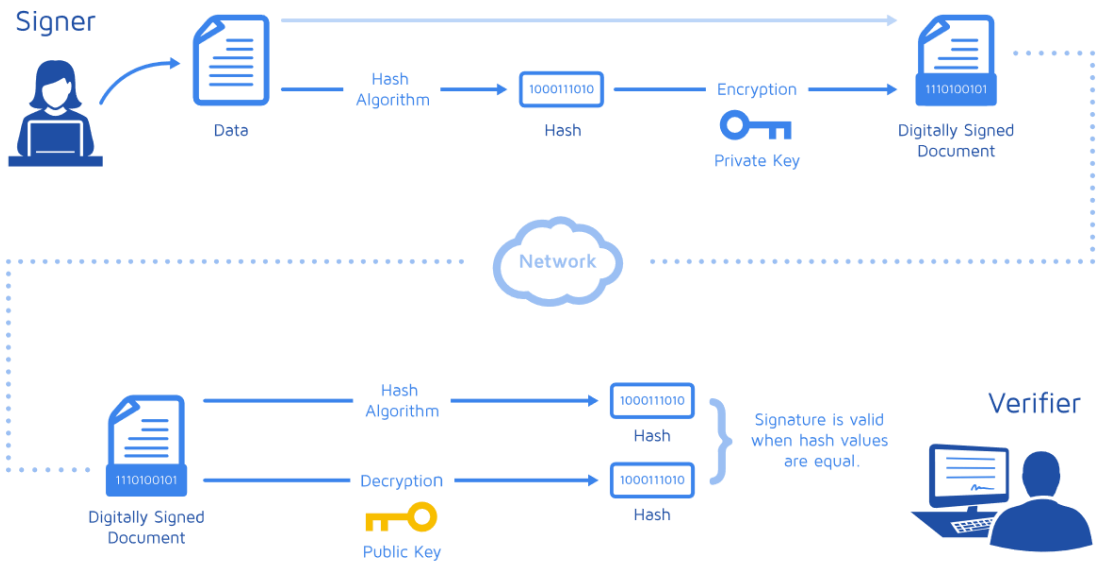

Figure 3 - Digital Signature Process

3.6. JSON Web Token (JWT)

JSON Web Tokens are an open, industry standard RFC 7519 [**RFC7519**] method for representing claims securely between two parties. JWT is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a

JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted. The JWT consists of three parts structured as JSON objects:

- Header – identification of the algorithm used to encrypt the token
- Payload – information stored in the token
- Signature – encrypted signature of header and payload

The three parts are Base64Url encoded and separated by a dot delimiter (.), which enables the token to be easily exchanged among systems and applications as a string value [**TREDER2019**]. In the context of InteropEHRate, JWTs will be used for authentication in the remote scenario. An authentication proxy will be responsible to authenticate the citizen and provide this token. The authorization service will store a citizen's authorization claims in the payload of a JWT. The token will be evaluated by a service receiving an API call bearing the token, enabling it to determine if the caller has access to the service's data or methods. The JWT will be sent to the citizen's side to enable the authenticated interaction with the server. Whenever a request is sent to the authentication proxy, the token is sent along with the request. Typically this is done in the authorization header like "Authorization: Bearer xxxxxx.yyyyy.zzzzzz". However, it could also be passed in a POST body or in the URL itself as a query parameter. When the server sees the token, it decodes it and compares the signature with the secret it has stored which would have been used to generate the token in the first place. If everything matches, the request is authentic, and it responds with data, otherwise it sends back an error message. The aforementioned steps are depicted in Figure 4 below.

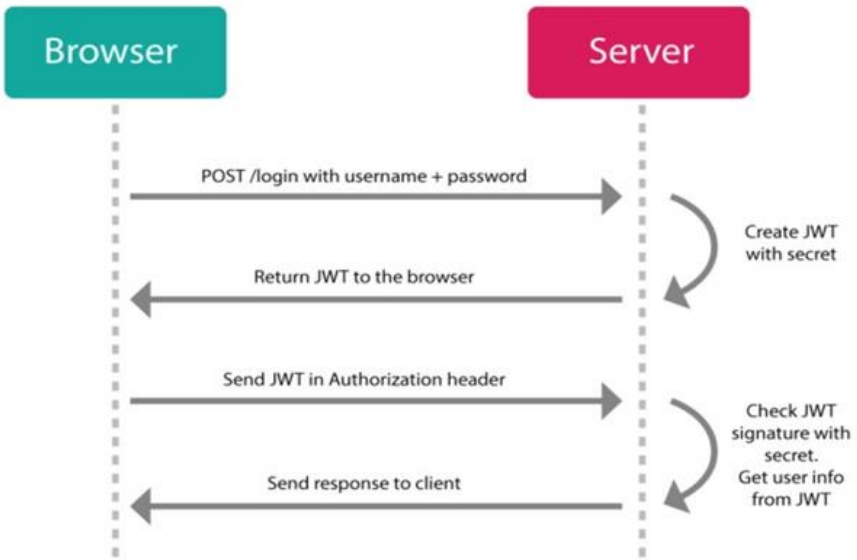

Figure 4 - JSON Web Token

4. HR SECURITY AND PRIVACY SERVICE LIBRARY (D2D)

This section emphasizes on the calls of the security library focused on D2D and describes the way they operate, their outputs and implementation details.

4.1. Zero-day operation (HCP)

On the first run of the HCP App, internet connection will be required. The HCP will generate a public and private key. This pair of keys will be stored on the HCP's TEE (Trusted execution environment) of their PC. The next step is for the HCP to generate a Certificate Signing Request (CSR). The CSR will be signed by an intermediate singing CA and a digital certificate will be delivered to the HCP's device. The CSR will include the HCP's sufficient details needed to specify her/his identity.

4.2. Zero-day operation (Citizen)

On the first run of the S-EHR App, an Internet connection will be required. The citizen will generate a public and private key. This pair of keys will be stored on the user's TEE (Trusted execution environment) of their mobile phones (e.g. Android keystore). The next step is for the citizen to generate a Certificate Signing Request (CSR). The CSR will be signed by an intermediate singing CA and a digital certificate will be delivered to the citizen's device. The CSR will include the citizen's sufficient details needed to specify her/his identity.

4.3. Bluetooth Pairing

As already introduced in D4.1 [**D4.1**], the main idea for the D2D scenario is the lack of Internet connection. The following section will analyse step by step this communication and present thoroughly where the security libraries is necessary to be invoked by the D2D protocol, the S-EHR app and HCP app.

4.3.1. Communication steps on pairing

We will present all the steps and calls that will take place between a citizen and HCP D2D pairing:

Physical World	S-EHR App / M-D2D-SM	HCP App / T-D2D-SM	Security Library Calls
Zero Day Operation completed for citizen	Stored Certificate for citizen	-	- fetchCertificate
Zero Day Operation completed for HCP	-	Stored Certificate for HCP	- fetchCertificate
Creation of QR code and Signed payload	MAC Address of HCP endpoint and the digital signature of the MAC has been exposed to S-EHR App	Generates QR code that includes a payload with the MAC address concatenated with its digitally signature	- signPayload - createPayload

User scans QR code	S-EHR App fetches QRcode and signature for verification	-	-
After connection established	After the connection is established we fetch the HCP's certificate to validate the signature	After the connection is established we fetch the citizen's certificate	- fetchHCPCertificate - fetchSEHRCertificate - verifySignature

Table 2 - Bluetooth pairing and security libraries step by step

4.4. Consent

In order for the processing to be lawful, personal data should be processed and used on the basis of the consent of the data subject concerned or some other legitimate basis. Consent of the data subject means any freely given, specific, informed and unambiguous indication of the data subject's wishes by which he or she, by a statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her. Initially the S-EHR app generates and signs a consent to store the user's data. After the connection has been established between the two devices, a consent request will be sent from the HCP to the citizen for data exchange (e.g. upload/download data). This consent needs to be double signed from both parties before it is sent back to the HCP App.

4.4.1. Communication steps on consent

The following table presents all the steps and calls that take place between a citizen and HCP D2D consent exchange:

Physical World	S-EHR App / M-D2D-SM	HCP App / T-D2D-SM	Security Library Calls
Citizen gives his consent store data to S-EHR app	User receives a pop up with necessary information to store his data	-	- generateAPPCConsent - signAPPCConsent
The HCP sends a consent request upload/download data to the citizen	User receives a pop up with necessary information to upload/download his data	HCP sends a consent request	- generateAPPCConsent - signAPPCConsent
Citizen accepts the consent	User accepts and proceeds	-	- verifyAPPCConsent - signAPPCConsent
HCP accepts the consent	-	HCP is notified and accepts	- verifyAPPCConsent

Table 3 - Consent and security libraries step by step

The HCP will generate the consent in her/his HCP App and sign it with the already existing private key. The consent will be sent to the citizen in order to be verified and signed by this entity. Finally, the double-signed consent will return back to the HCP for verification.

4.5. Security Libraries in D2D

This section describes the functionalities of security libraries in the context of D2D.

4.5.1. Security Library for S-EHR App / M-D2D-SM

Security in Android devices, builds on the Java Cryptography Architecture (JCA), that provides API for digital signatures, certificates, encryption, key generation and management. The Android keystore will be used to store all the necessary keys and certificates. The KeyStore class is an engine class that supplies well-defined interfaces to access and modify the information in a keystore. An example of Android KeyStore is depicted in Figure 5 below. This class represents an in-memory collection of keys and certificates. KeyStore manages two types of entries [**JCA2018**]:

- **Key Entry** - This type of keystore entry holds very sensitive cryptographic key information, which is stored in a protected format to prevent unauthorized access. Typically, a key stored in this type of entry is a secret key, or a private key accompanied by the certificate chain authenticating the corresponding public key.
- **Trusted Certificate Entry** - This type of entry contains a single public key certificate belonging to another party. It is called a trusted certificate because the keystore owner trusts that the public key in the certificate indeed belongs to the identity identified by the subject (owner) of the certificate.

Each entry in a keystore is identified by an "alias" string. In the case of private keys and their associated certificate chains, these strings distinguish among the different ways in which the entity may authenticate itself. For example, the entity may authenticate itself using different certificate authorities, or using different public key algorithms. Android support PKCS#12 key store files with .pfx or .p12 extensions.

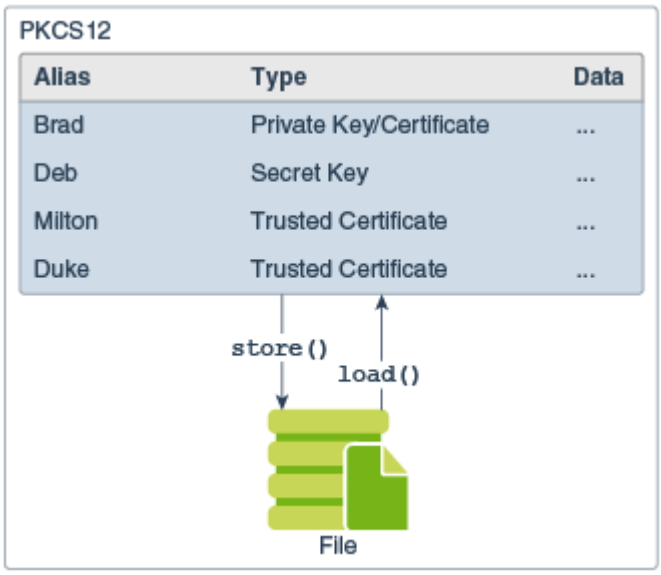

Figure 5 - Android KeyStore Class

A high-level components diagram is presented in Figure 6 below.

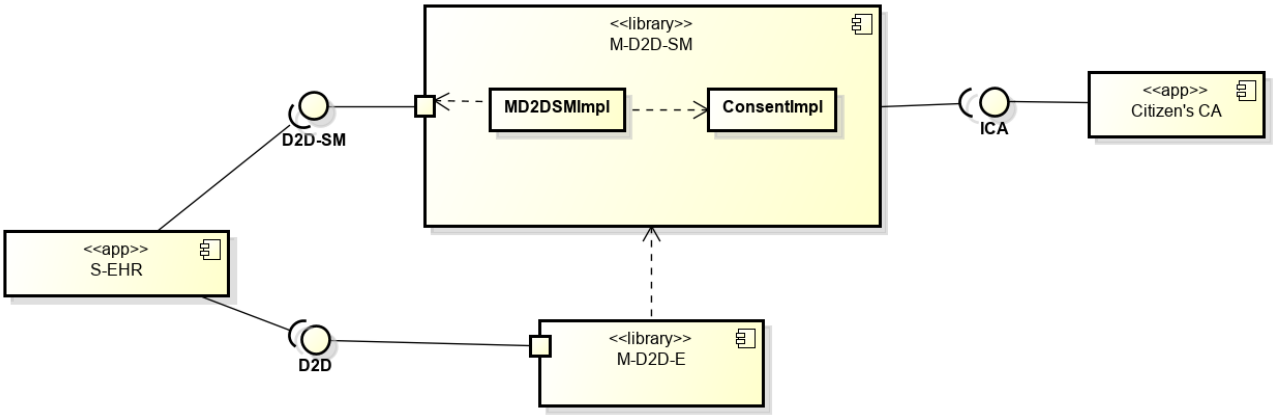

Figure 6 - Security library components in D2D (M-D2D-SM

(a) Prerequisite

Operation fetchCertificate

Name	fetchCertificate
Description	This call generates an X.509 certificate signed by the citizen's CA upon a CSR request. This certificate is received and stored to the Android keystore of the mobile device. When generating or importing a key into the Android keystore the key will be used if the user has been authenticated first in the device. Once keys are in the keystore, they can be used for cryptographic operations with the key material remaining non-exportable [**ANDROID2019**]. The Android keystore in allows to access certificate and keys from PKCS12 files. This operation is invoked by S-EHR App.
Arguments	• No arguments
Return Value	• PublicKey
Exceptions	• NoSuchKeyException • KeyStoreException • CertificateException • NoSuchAlgorithmException • IOException

Preconditions	• Internet Connection • Public/Private key generation and storage in Android keystore

Table 4 - fetchCertificate

(b) After pairing / Identity Management

Operation fetchHCPCertificate

Name	fetchHCPCertificate
Description	After the bluetooth pairing establishment, the first message should be the transfer of HCP public key (certificate). Such a message is necessary in order the S-EHR App to be able to validate the HCP signature for identification purposes. This operation is invoked by D2D library to transfer the HCP public key.
Arguments	• No arguments
Return Value	• HCP's PublicKey encoded in base64
Exceptions	• Network exceptions related to Bluetooth state.
Preconditions	• Bluetooth Pairing has taken place

Table 5 - fetchHCPCertificate

Operation verifySignature

Name	verifySignature
Description	The S-EHR App scanned the QR code and receives the MAC address along with the signature. This operation is invoked by S-EHR App.
Arguments	• scanned QR code
Return Value	• Boolean - true if the signature was verified, false if not.

Exceptions	• SignatureException - if this signature object is not initialized properly, or the passed-in signature is improperly encoded or of the wrong type, etc.
Preconditions	• QR code scanned successfully

Table 6 - verifySignature

(c) After pairing / Consent Management

Operation generateAPPCConsent

Name	generateAPPCConsent
Description	S-EHR app creates and stores the consent, in XML format. This operation is invoked by S-EHR App.
Arguments	• String From • String To • Date timestamp • Date Duration • String Purpose • String Type • String Reference text
Return Value	• Consent in XML format
Exceptions	• IOException
Preconditions	• Citizen gives his consent to store the data.

Table 7 - generateAPPCConsent

Operation signAPPCConsent

Name	signAPPCConsent

Description	The signature algorithm is the NIST standard Digital Signature Algorithm (DSA), using DSA and SHA-256. The call will use the private key stored in the Android keystore to initialize the signing operation. This operation is invoked by S-EHR App.
Arguments	• Consent
Return Value	• byte[] - Returns the signature bytes of all the Consent
Exceptions	• SignatureException - if an error occurs or length is less than the actual signature length.
Preconditions	• HCP has successfully fetched his credentials

Table 8 - signAPPCConsent

Operation verifyAPPCConsent

Name	verifyAPPCConsent
Description	This verifies that the HCP has signed with their digital signature the consent and the citizen verify and acknowledges the data transfer. This operation is invoked by S-EHR App.
Arguments	• Consent • byte[] signature
Return Value	• Boolean - true if the signature was verified, false if not.
Exceptions	• SignatureException - if this signature object is not initialized properly, or the passed-in signature is improperly encoded or of the wrong type, etc.
Preconditions	• HCP has requested for the consent

Table 9 - verifyAPPCConsent

Operation signAPPCConsent

Name	signAPPCConsent
Description	The signature algorithm is the NIST standard Digital Signature Algorithm (DSA), using DSA and SHA-256. The call will use the private key stored in the keystore to initialize the signing operation. This operation is invoked by S-EHR App.
Arguments	• Consent
Return Value	• byte[] - Returns the signature bytes of all the Consent
Exceptions	• SignatureException - if an error occurs or length is less than the actual signature length.
Preconditions	• HCP has successfully fetched his credentials

Table 10 - signAPPCConsent

4.5.2. Security Library for HCP App / T-D2D-SM

Java Keytool is a key and certificate management tool that is used to manipulate Java Keystores, and is included with Java. A Java Keystore is a container for authorization certificates or public key certificates, and is often used by Java-based applications for encryption, authentication, and serving over HTTPS. Its entries are protected by a keystore password. A keystore entry is identified by an alias, and it consists of keys and certificates that form a trust chain [**ANICAS2014**].

A high-level components diagram is presented in Figure 7 below.

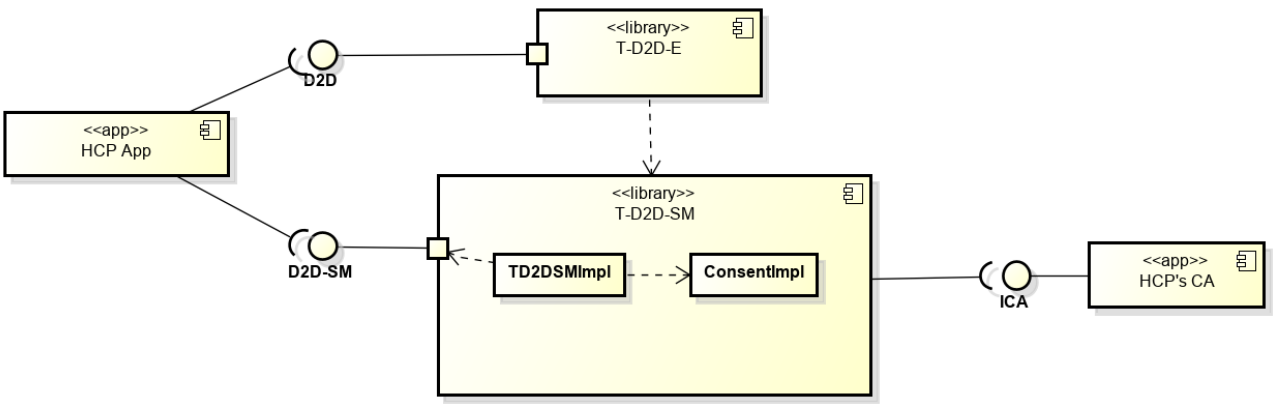

Figure 7 - Security library components in D2D (T-D2D-SM)

(a) Before pairing / Identity Management

Operation fetchCertificate

Name	fetchCertificate
Description	This call generates an X.509 certificate from the HCP's CA upon a CSR request. This certificate is received and stored to the keystore of the HCP's device. In Java 9, the default keystore type will be changed to PKCS12, while in earlier versions was the Java Key Store (JKS). PKCS12 is a file format to store certificates and private keys. The KeyStore API in Java also allows to access certificate and keys from PKCS12 files. This operation is invoked by HCP App.
Arguments	• No arguments
Return Value	• PublicKey
Exceptions	• NoSuchKeyException • KeyStoreException • CertificateException • NoSuchAlgorithmException • IOException
Preconditions	• Internet Connection • Public/Private key generation and storage in keystore

Table 11 - fetchCertificate

Operation signPayload

Name	signPayload
Description	The signature algorithm is the NIST standard Digital Signature Algorithm (DSA), using DSA and SHA-256. The call will use the private key stored in keystore to initialize the signing operation. This operation is invoked by the HCP App.

Arguments	• MAC address
Return Value	• byte[] - Returns the signature bytes of all the MAC address
Exceptions	• SignatureException - if an error occurs or length is less than the actual signature length.
Preconditions	• HCP has successfully fetched his credentials

Table 12 - signPayload

Operation createPayload

Name	createPayload
Description	The HCP creates the payload with their attributes (mac address, etc..) and the signature to be used for the QR code. This operation is invoked by the HCP App.
Arguments	• MAC address • signed MAC address
Return Value	• concatenated payload to feed the QR code
Exceptions	• No exceptions
Preconditions	• HCP has signed the MAC address

Table 13 - createPayload

(b) After pairing / Identity Management

Operation fetchSEHRCertificate

Name	fetchSEHRCertificate
Description	After the Bluetooth pairing is established, the first message should be the transfer of citizen's public key (certificate). The HCP app should fetch the

	Certificate of S-EHR app in order to verify it's public key. This operation is invoked by the D2D library in order to transfer the citizen's public key.
Arguments	• No arguments
Return Value	• Citizen's public key encoded in base64
Exceptions	• Network exceptions related to Bluetooth state.
Preconditions	• Bluetooth pairing has taken place

Table 14 - fetchSEHRCertificate

(c) After pairing / Consent Management

Operation generateAPPCConsent

Name	generateAPPCConsent
Description	The HCP app creates the consent, in XML format, to upload/download data to/from S-EHR app. This operation is invoked by HCP App.
Arguments	• String From • String To • Date timestamp • Date Duration • String Purpose • String Type • String Reference text
Return Value	• Consent in XML format
Exceptions	• IOException

Preconditions	• Citizen gives their consent to upload/download data. • HCP gives their consent to upload/download data.

Table 15 - generateAPPCConsent

Operation signAPPCConsent

Name	signAPPCConsent
Description	The signature algorithm is the NIST standard Digital Signature Algorithm (DSA), using DSA and SHA-256. The call will use the private key stored in the keystore to initialize the signing operation. This operation is invoked by HCP App.
Arguments	• Consent
Return Value	• byte[] - Returns the signature bytes of all the Consent
Exceptions	• SignatureException - if an error occurs or length is less than the actual signature length.
Preconditions	• The HCP has successfully fetched their credentials

Table 16 - signAPPCConsent

Operation verifyAPPCConsent

Name	verifyAPPCConsent
Description	This verifies that the citizen has signed with their digital signature the consent and the HCP verify and acknowledges the data transfer. This operation is invoked by the HCP App.
Arguments	• Consent • byte[] signature

Return Value	• Boolean - true if the signature was verified, false if not.
Exceptions	• SignatureException - if this signature object is not initialized properly, or the passed-in signature is improperly encoded or of the wrong type, etc.
Preconditions	• The Citizen gives his consent to upload/download data.

Table 17 - verifyAPPCConsent

5. HR SECURITY AND PRIVACY SERVICE LIBRARY (R2D)

This section emphasizes on the calls of the security library focused on R2D and describes the way they operate, their outputs and implementation details.

5.1. Authentication through Authentication Proxy

Since each nation employs its own different authentication mechanisms, in the context of InteropEHRate, we will use pluggable authentication with different modalities. For that reason, an authentication proxy will be used for InteropEHRate users to be authenticated. Authentication is an interaction between the S-EHR app and Identity provider through the Authentication proxy. In our case, in order to test and be more flexible throughout the project's development period, we will create our own identity provider and verify some test users like any real-life use case. Upon the verification of user's identity from the identity provider. Authentication proxy provides the persistent JWT token.

Two-factor authentication (2FA) adds an additional layer of security by introducing a second step to citizen login. It takes something you know (i.e. your password), and adds a second factor, typically something you physically have (such as your phone). Since both are required to log in, in the case where an attacker obtains your password, the two-factor authentication would stop them for accessing your account. There are a variety of methods that can be used for two-factor authentication. Some of these methods are text messages, time-based tokens, fingerprint, face recognition and a more extreme one is using external hardware like yubico. As a reference implementation of authentication mechanism, the FIDO U2F mechanism will be provided. Figure 8, below depicts how the abstract R2D authentication mechanism.

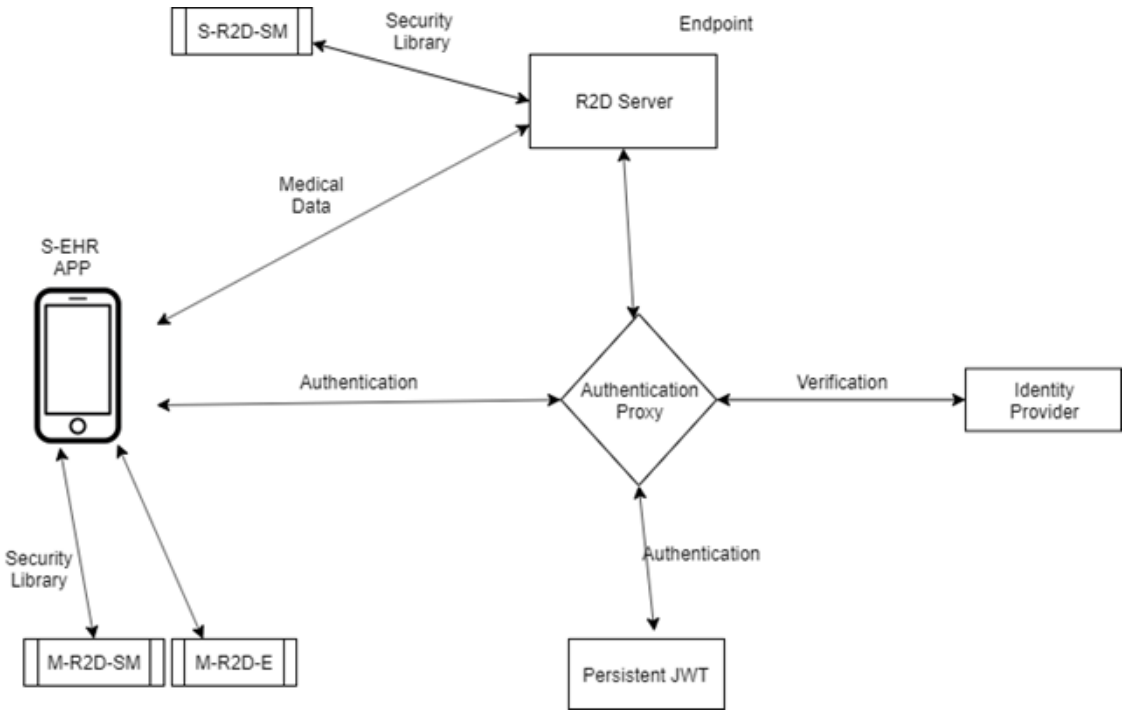

Figure 8 - Abstract R2D authentication

To sum up, the Authentication proxy will:

- Authenticate the user requests from the S-EHR app;
- Interact with epSOS for the authentication and authorization requests from the S-EHR app;
- Evaluate the authentication and authorization by utilizing the JWT token.

5.2. Security Libraries in R2D

This section describes the functionalities of security libraries in the context of R2D.

5.2.1. Security Library for S-EHR App / M-R2D-SM

A high-level components diagram is presented in Figure 9 below.

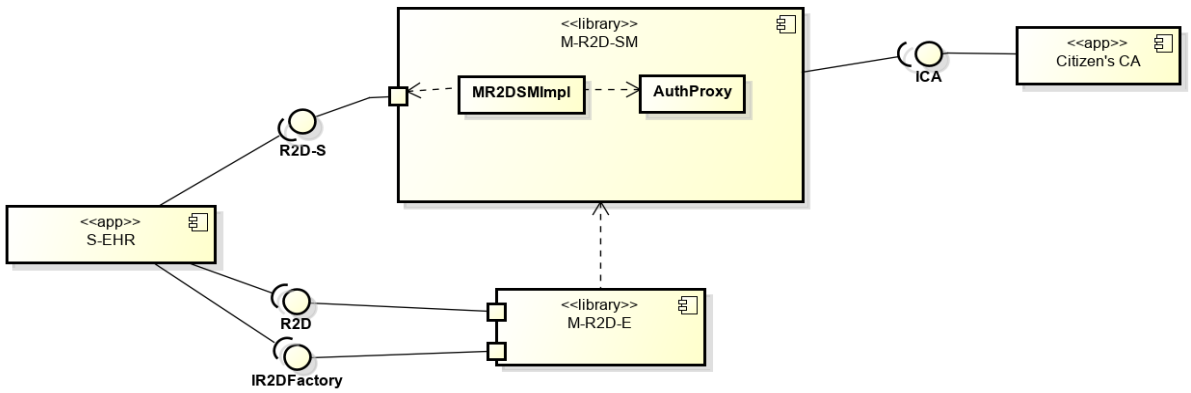

Figure 9 - Security library components in R2D (M-R2D-SM)

Operation getAuthenticationMeans

Name	getAuthenticationMeans
Description	This call asks from the Identity Provider (IDP) the authentication means that are necessary for authentication. This operation is invoked by S-EHR App.
Arguments	• No arguments
Return Value	List <AuthMeans>
Exceptions	• Network exceptions

Preconditions	• Citizen has already registered to a national Identity Provider (IDP)

Table 18 - getAuthenticationMeans

Operation getAuthattributes

Name	getAuthattributes
Description	This call returns from the IDP through the authentication proxy the attributes and validators (e.g. first name, last name, etc.) that are necessary for authentication. The landing page on the S-EHR App will change based on the country. This operation is invoked by S-EHR App.
Arguments	• authentication mean
Return Value	• List <Attributes, Validators> ○ returns a distinct set of attributes and regular expression (regex) validators per action that will be displayed in a form in S-EHR App.
Exceptions	• Network exceptions
Preconditions	• Citizen has already registered to a national Identity Provider (IDP)

Table 19 - getAuthattributes

Operation get2FAmeans

Name	get2FAmeans
Description	This call gets the available means of two-factor authentication (2FA) activation between the citizen and the authentication proxy. This operation is invoked by S-EHR App.
Arguments	• No arguments

Return Value	List <2FAMeans>
Exceptions	• Network exceptions
Preconditions	• Citizen has already registered to authentication proxy.

Table 20 - get2FAmeans

Operation authenticate2FA

Name	authenticate2FA
Description	This call is the 2FA challenge/response against the existing proxy application. Part of this AuthResponse is the JWT persistence token which is stored in the authentication proxy. This token is necessary to authenticate the citizen and is appended to all requests in the system. In every request the token will be re-evaluated. This operation is invoked by S-EHR App.
Arguments	• challengeBasedOnMeans
Return Value	• AuthResponse ○ JWT token
Exceptions	• Network exceptions
Preconditions	• Citizen has already registered to authentication proxy.

Table 21 - authenticate2FA

Operation bindUserWith2FA

Name	bindUserWith2FA
Description	This call binds a specific user with a specific 2FA mean (e.g. SMS, FIDO 2FA,

	hardware-based etc.). This operation is invoked by S-EHR App.
Arguments	• 2FA means
Return Value	• BindingResponse
Exceptions	• Network exceptions
Preconditions	• Citizen has already registered to authentication proxy.

Table 22 - bindUserWith2FA

Operation authenticate

Name	Authenticate
Description	This call is the actual authentication process. This operation is invoked by S-EHR App.
Arguments	• challengeBasedOnMeans
Return Value	• AuthResponse
Exceptions	• Network exceptions
Preconditions	• Citizen has already authenticated to the proxy

Table 23 - authenticate

6. CONCLUSIONS AND NEXT STEPS

The objective of this report was to deliver the initial version of the design of the security libraries offered by the InteropEHRate Framework as a reference implementation. In the same notion as the other reports of the InteropEHRate project, this document presents a first draft of the intended content of the security libraries and their further functionality purposes.

However, it should be mentioned that other two updated versions of this report are planned to be released. The one is planned to be released on December 2020, whilst the final one is planned to be released on December 2021, both of them including the relevant updates, of all the security libraries. In the next version of the Design of libraries for HR security and privacy services, based on the current implementation, the needs as well as the additional functionalities that will be required, a new version of the libraries' design will be released for the intended audience.

REFERENCES

- **[D3.3]** InteropEHRate Consortium, Specification of remote and D2D IDM mechanisms for HRs Interoperability - V1, 2019. www.interopehrate.eu/resources
- **[D3.7]** InteropEHRate Consortium, Specification of consent management and decentralized authorization mechanisms for HR Exchange, 2019. www.interopehrate.eu/resources
- **[D4.1]** InteropEHRate Consortium, Specification of remote and D2D protocol and APIs for HR exchange - V1, 2019. www.interopehrate.eu/resources
- **[epSOS2014]** Digital Single Market, Cross-border health project epSOS: What has it achieved?, 2014 Website:https://ec.europa.eu/digital-single-market/en/news/cross-border-health-project-epsos-what-has-it-achieved
- **[eIDAS2014]** Digital Single Market, Trust Services and Electronic identification (eID), 20104 Website:https://ec.europa.eu/digital-single-market/en/trust-services-and-eid
- **[PKI]** Thales, What is Public Key Infrastructure (PKI)? Website:https://www.thalesesecurity.com/faq/public-key-infrastructure-pki/what-public-key-infrastructure-pki
- **[RFC7519]** Internet Engineering Task Force (IETF) , JSON Web Token (JWT), 2015 Website:https://tools.ietf.org/html/rfc7519
- **[TREDER2019]** Treder, M., Protecting JavaScript Microservices on Node.js with JSON Web Tokens and Twilio Authy, 2019 Website:https://www.twilio.com/blog/protecting-javascript-microservices-node-js-json-web-tokens-twilio-authy
- **[DS2019]** Digital Signatures, What are digital signatures?, 2019 Website:https://www.signinghub.com/digital-signatures/
- **[ALDEY2018]** Adley, J., Understanding the Role of Certificate Authorities in PKI, 2018, Website:https://dzone.com/articles/understanding-the-role-of-certificate-authorities
- **[THESSLSTORE2019]** The Difference Between Root Certificates and Intermediate Certificates, 2019 Website:https://www.thesslstore.com/blog/root-certificates-intermediate/
- **[GLOBALPLATFORM2018]** GLOBALPLATFORM, Introduction to Trusted Execution Environments, 2018 Website:https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf
- **[ANDROID2019]** Google, Android keystore system, Website: https://developer.android.com/training/articles/keystore
- **[EJBCA2019]** EJBCA Enterprise from PrimeKey, Website: https://www.primekey.com/products/software/ejbca-enterprise/
- **[JCA2018]** Oracle, Java Cryptography Architecture (JCA) Reference Guide, 2018. Website: https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html
- **[ANICAS2014]** Anicas, M., Java Keytool Essentials: Working with Java Keystores, 2014 Website:https://www.digitalocean.com/community/tutorials/java-keytool-essentials-working-with-java-keystores

APPENDIX A

This Appendix provides information on how the EJBCA CA and Sub CAs issued the certificates using openssl.

Root CA

Create Root CA key

$ openssl req -new -x509 -sha256 -days 3650 -key (root name).key -reqexts v3_req -extensions v3_ca -out (root name).crt

Sub CA

Create Sub CA key

$ openssl genrsa -out (sub ca name).key 4096

Create Sub CA request

$ openssl req -new -key (sub ca name).key -reqexts v3_req -extensions v3_ca -out (sub ca name).csr

Sign Intermediate CA request

$ openssl x509 -req -days 730 -in (sub ca name).csr -CA (root name).crt -CAkey (root key).key -set_serial 01 -out (sub ca name).crt

Create CA chain file

$ cat (sub ca name).crt (root name).crt > (chain name).crt

Create p12 file

$ openssl pkcs12 -export -out (name).p12 -inkey (sub ca name).key -in (sub ca name).crt -chain -CAfile (root name).crt