D3.6

Specification of data encryption mechanisms for mobile and web applications - V2

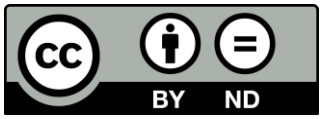
ABSTRACT

This deliverable provides the second and final version of the specification of protocols for encryption mechanisms for both health data storage and health data exchange. This document also provides a detailed technical background for encryption mechanisms, which is a necessary step to move forward for both data in-transit and data at rest. The deliverable includes the encryption aspects of all the involved architectural components (e.g., S-EHR App, HCP Web App, S-EHR Cloud, and Reference Research Center), protocols (e.g., D2D, R2D Access, R2D Backup, R2D Emergency, RDS), and scenarios (e.g. Medical Visit, Emergency and Research) for data at rest and in-transit.

Delivery Date	August 9th 2021
Work Package	WP3
Task	T3.5
Dissemination Level	Public
Type of Deliverable	Report
Lead partner	UBIT

This 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	Sofianna Menesidou, Entso Veliou, Menelaso Giannopoulos, Thanassis Giannetsos	UBIT
Reviewers	Paolo Marcheschi	FTGM
	Christina Kotsiopoulou	HYGEIA

LOGTABLE

Version	Date	Change	Author	Partner
0.1	12-05-21	ToC and cleaning from the v1	Sofianna Menesidou	UBIT
0.2	17-05-21	Section 3 restructuring	Sofianna Menesidou	UBIT
0.3	18-05-21	Section 3	Sofianna Menesidou	UBIT
0.4	25-05-21	Section 3	Sofianna Menesidou	UBIT
0.5	27-05-21	Section 3	Sofianna Menesidou	UBIT
0.6	28-05-21	Section 3	Sofianna Menesidou	UBIT
0.7	31-05-21	Section 3	Sofianna Menesidou	UBIT
0.8	14-06-21	Section 3	Sofianna Menesidou	UBIT
0.9	15-06-21	Section 3	Sofianna Menesidou, Menelaos Giannopoulos	UBIT
1.0	23-06-21	Appendix	Sofianna Menesidou	UBIT
1.1	06-07-21	Quality check	Argyro Mavrogiorgou	UPRC
1.2	03-08-21	Section 3 updates based on comments	Sofianna Menesidou	UBIT
VFInal	06-08-2021	Final technical check and version for submission	Francesco Torelli Laura Pucci	ENG

ACRONYMS

Acronym	Term and definition
ABAC	Attribute Based Access Control
ABE	Attribute-based encryption
AES	Advanced Encryption Standard
BIOS	Basic Input/Output System
BLE	Bluetooth Low Energy
CA	Certificate Authority
CAI	Certificate Authority Interface
CP-ABE	Ciphertext policy Attribute-based encryption
CRL	Certificate Revocation List
CRUD	create, read, update and delete
CSR	Certificate Signing Request
DES	Data Encryption Standard
DH	Diffie-Hellman
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
EMR	Electronic Medical Records
FDE	Full Disk Encryption
HTTPS	Hypertext Transfer Protocol Secure
IBE	Identity-based Encryption
IdP	Identity Provider
IaaS	Infrastructure-as-a-Service
KP-ABE	Key policy Attribute-based encryption
PKG	Private Key Generator

PKI	Public Key Infrastructure
RBAC	Role Based Access Control
RRC	Reference Research Center
RSA	Rivest – Shamir – Adleman
SAML	Security Assertion Markup Language
SEV	Secure Encrypted Virtualisation
SGX	Software Guard Extension
SoC	System on Chip
SQL	Structured Query Language
SSL	Secure Socket Layer
TDE	Transparent Data Encryption
TPM	Trusted Platform Module
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TTP	Trusted Third Party

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

According to OWASP, two out of the top ten mobile risks are a) insecure communications and b) insecure data storage **[owasp2020]**. On one hand, insecure data transmission to and from a mobile app generally takes place through a telecom carrier and/or over the internet. Hackers intercept data either by interfering with the local area network of users through a compromised Wi-Fi network, by tapping into the network through routers, cellular towers, proxy servers, or by exploiting an infected app through malware. Insecure data storage is an easy way in which an adversary can access data in a mobile device. On the other hand, an adversary can either gain physical access to a stolen device or enter into it using malware or a repackaged app.

Encryption is the main technique to mitigate both insecure communications and data storage. Healthcare data encryption has become a popular option for protecting sensitive medical information. The need for encryption has become more prevalent with the rapid increase in the number of practices using Electronic medical records (EMRs) and mobile devices. Encryption is a means to protect patient health information when it is transmitted from one user to another.

In addition, the healthcare industry can benefit from cloud technology to facilitate communication, collaboration, and coordination among different healthcare providers. However, to ensure the patients' control over access to their own health data, it is necessary to encrypt the data before they are transferred and stored in the cloud. In fact, outsourcing to the cloud brings several security risks.

Due to the high value of sensitive health data, third-party storage servers are often the targets of various malicious behaviours which may lead to exposure of the data. That was the case of the famous incident of the stored data in the Department of Veterans Affairs database containing sensitive PHI of 26.5 million military veterans, including their social security numbers and health problems that was stolen by an employee who took the data home without authorization **[La2006]**.

Last but not least, in emergency situations, it is crucial, for sensitive encrypted data, to be able to be decrypted when a specific access control policy on who can decrypt the data applies **[Bethencourt2007]**.

1.1 Scope of the document

The main goal of the present document is to describe the InteropEHRate specification of protocols for encryption mechanics for both a) health data storage on mobile devices, HCP App and cloud services and b) health data exchange. Moreover, the deliverable describes the research conducted regarding encryption mechanisms. In a nutshell, for data encryption in-transit, we propose apart from having enabled the encryption mechanisms that are supported by Bluetooth and HTTPS over the Internet, an application level encryption for encrypted communication. In the same manner, for data encryption in storage apart from full disk encryption based on TEE mechanisms, we propose an application level encryption for encrypted storage. To this end, a detailed symmetric encryption/decryption specification concerning all the InteropEHRate protocols will be provided.

1.2 Intended audience

The document is mainly intended for developers, architects, manufacturers, security engineers, and all the project participants and partners interested to have an overview of how the InteropEHRate supports encryption/decryption mechanisms for data storage and data exchange.

1.3 Structure of the document

This deliverable is structured as follows:

- **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 and reviews the research background regarding encryption mechanisms for both data storage and data exchange including aspects regarding confidential computing.
- **Section 3** introduces the overall encryption/decryption mechanisms in terms of InteropEHRate, where it is analysed in detail for both data storage and data exchange for all the InteropEHRate protocols. This section includes the security models for all the security protocols to highlight the used crypto-primitives.
- **Section 4** concludes the deliverable and highlights the most important aspects of the encryption/decryption algorithms used.
- **Appendix A** summarises all the cryptographic notations used for a better understanding of the modelling of protocols and the JSON schemas for D2D requests.

1.4 Updates with respect to previous version (if any)

Several updates have been made with respect to the previous version. The most important are:

- Background extended with the PKI and the concept of hierarchy of trust among CAs for cross-border trust establishment.
- A summary and table with all the technologies adopted in InteropEHRate is also included as a sub-section.
- Description of all the security models and crypto-primitives regarding data encryption mechanisms per protocol is included and described in the deliverable.
- The structure of Chapter 3 is completely restructured based on the InteropEHRate protocols for a clearer presentation. In addition, all the security protocols are analyzed in comparison with the previous version of the deliverable.
- Specification has been updated with the inclusion of Diffie-Hellman (DH) key agreement APIs, the inclusion of RDS protocol and a clear distinction between the R2D-based protocols namely R2D Access, R2D Backup and R2D Emergency.
- Specification was updated by removing the CP-ABE application level encryption for the R2D Emergency protocol for simplicity.
- Conclusion section was updated, while no next steps have been included since this is the final version of the deliverable.
- An appendix with all the cryptographic notations of the security models included in the deliverable and the JSON schemas for D2D requests.

2 TECHNICAL BACKGROUND

This chapter includes the necessary background and terminology for the encryption mechanisms, starting from the cryptography basics, the state-of-the-art solutions for both data storage and data exchange and a detailed literature review on the challenging aspect of cloud data storage. In addition, the concept of confidential computing is also highlighted, while the most known commercial TEE technologies are compared.

2.1 Cryptography

Cryptography is one of the most used techniques to build security and is an indispensable tool for protecting information in computer systems **[Ghulam2018]**. Cryptography is used to store and transfer data in such a form that only the sender and the receiver can understand or process it. In addition, cryptography depends upon both the algorithm and the key. There are two main types of Cryptography: Symmetric key cryptography and Asymmetric cryptography.

Symmetric Key Cryptography: In symmetric key cryptography, a shared secret key is used between the sender and recipient in order to encrypt and decrypt the data. There are many algorithms that are based on symmetric key cryptography, like Caesar cipher, Block cipher, Stream cipher, DES (Data Encryption Standard), and AES (Advanced Encryption Standard). The main disadvantage of using symmetric key cryptography is the need to exchange the secret key between the sender and the receiver in a secure manner. In addition, symmetric algorithms such as the AES demand only a small amount of computational power **[Lisonek2008]**.

Asymmetric Key Cryptography: In asymmetric key cryptography, also called public key cryptography, two different keys are used for encryption and decryption. These two keys are known as a public key and private key, where one the former is used for encryption and the latter is used for decryption. The private key is a secret key, private key never exposed. There are many algorithms that are based on asymmetric key cryptography, like Diffie-Hellman, RSA (Rivest - Shamir - Adleman) and Elliptic Curve Cryptography (ECC). This method of encrypting data eliminates the need for the existence of a unique shared key between the communicating partners, but requires more computational power to perform manipulations on the data in comparison to symmetric cryptographic techniques **[Lisonek2008]**.

Identity-based encryption: The identity-based encryption is a type of asymmetric key encryption in which a user's public key is a string (can be a user's identity or mail address) combined with a public master key. User obtains his private key from the Private Key Generator (PKG) **[BF03]**.

Attribute-based encryption: Attribute-based encryption (ABE) is a recent promising cryptographic method proposed by Sahai and Waters in 2005 **[SW05]**. The ABE technique extends identity-based encryption (IBE) to enable expressive access policies and fine-grained access to encrypted data. In both schemes IBE and ABE, cryptographic keys are managed by a Trusted Third Party (TTP), usually called Attribute Authority (AA). In ABE, data is encrypted along with an access structure which is the logical expression of the access policy. The encrypted data can be decrypted by any user if his secret key has attributes that satisfy the access policy. The power of ABE is that we do not need to rely on the storage server to avoid unauthorized data access since the access policy is embedded in the ciphertext itself **[Lounis2014]**. The two main variants of ABE are the Key-Policy Attribute-Based Encryption (KP-ABE) **[GPSW06]** and the Ciphertext Policy Attribute-Based Encryption (CP-ABE) **[BSW07]**.

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 for Public Key Infrastructure (PKI) is public key cryptography. 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). In the literature, five PKI trust models are mainly used: the hierarchical trust model, the mesh trust model, the bridge CA trust model, the hybrid trust model, and the trust list trust model **[Uahhabi2014]**. More information regarding each PKI trust model can be found in **[Uahhabi2014]**.

2.2 Encryption for Data in Transit

Security is one of the main challenges when it comes to eHealth services and is a crucial requirement for the transmission of required health data across the network. Data in transit is vulnerable to interception and potentially redirection attacks. InteropEHRate deals with five protocols namely D2D, R2D Access, R2D Backup, R2D Emergency and RDS. D2D is over Bluetooth without Internet usage, while R2D Access, R2D Backup, R2D Emergency and RDS are over the Internet. This section will provide a brief overview of the encryption mechanisms used. In the context of InteropEHRate it's assumed that common best practices, such as HTTPS (Hypertext Transfer Protocol Secure), are enabled, but will also be provided as an extra security layer at the application level encryption.

2.2.1 Data exchange over Bluetooth

Bluetooth devices are used to exchange encrypted data over an encrypted link with the use of a "link key". The creation of that key depends on the pairing methods **[Lecroy]**. These pairing methods help the users to decide whether they exchange no key at all, or if they want to use a 6-digit (randomly or not) generated passcode which is used to authenticate the users **[Loveless2018]** **[Ravikiran]**. In addition, if the devices have enabled out-of-band communication channels, then all the needed information and the key will be exchanged out of the Bluetooth band. If two devices want to share information, for instance a file, then they have to (i) first, exchange device information to establish a secure connection and (ii) through the use of the common key, which they agreed to, encrypt the connection. After the establishment of the secure channel, they can securely exchange their data **[bon2016]** **[Ravikiran]**.

Prior to Bluetooth version 2.1, pairing was not secure at all **[Lecroy]**. A passive eavesdropper was able to crack the user's PIN and then compute the traffic key. Since Bluetooth v2.1 Secure Simple Pairing is used, which uses Elliptic Curve Diffie-Hellman (ECDH) for establishment of the session keys. In this way, a passive eavesdropper is prevented from obtaining the traffic keys. Version 4.0 established Bluetooth Low Energy (BLE), which approached the traffic encryption using the AES algorithm. But even though the encryption is better, the lack of use of ECDH made the encryption keys vulnerable to passive eavesdroppers **[Corella2015]**. In the context of InteropEHRate, the latest AES encryption for Bluetooth will be used, apart from the application level encryption.

2.2.2 Data exchange over the Internet

Traditionally, a secure socket layer (SSL) is used to establish secure communications. However, the IETF deprecated SSL in 2015, with Transport Layer Security (TLS) 1.0 supplanting SSL 3.1, but the 'SSL' tag has stuck, often representing both standards. A website that has implemented these cryptographic protocols is marked Secure HTTPS (HTTP within SSL/TLS), which should be table stakes for any mobile app.

HTTPS is an extension of the Hypertext Transfer Protocol and the letter "S" is referred to as Security. HTTPS is used to establish secure communication over a computer network **[Sullivan2018]**. Clients and servers can communicate the same way as they did by using HTTP, but in this case, they communicate over a secure SSL or TLS connection, which encrypts and decrypts the messages that both client and server exchange. As HTTPS is the secure version of HTTP, it adds encryption in HTTP in order to increase the security of the data being transferred. In practice, this provides an assurance that no one can possibly alter the communications between two parties **[Kothari2019]**.

Transport Layer Security (TLS) is a widely used security protocol, which protects the data that is transmitted online, between a web browser and a website through HTTPS. TLS also provides confidentiality and data integrity through encryption and it ensures that the other party in a connection is who he says that he is **[Lake2019]**. By using both symmetric and asymmetric encryption a secure connection is established and so the data are transmitted between client and server. The client and the server should agree to the algorithms that they will use for both symmetric and asymmetric encryption. Negotiation for the agreement on the utilised algorithms is handled internally by the protocol. The most frequent algorithm for symmetric encryption is the Advanced Encryption Standard (AES) and for asymmetric encryption is Diffie-Hellman **[Prodromou2019]**.

2.3 Encryption for Data in Storage

An end user device is a personal computer (desktop or laptop), a consumer device (e.g., personal digital assistant, smart phone), or a removable storage media (e.g., USB flash drive, memory card, external hard drive, writable CD or DVD) that can store information. Storage security is the process of allowing only authorized parties to access and use stored information **[nist800-111]**. Data at rest is extremely vulnerable, and thus, in the context of InteropEHRate we will focus on mobile, desktop and cloud data storage since they are the main involved devices in the InteropEHRate architecture. According to **[nist800-111]** the common types of storage encryption are:

- **Full Disk Encryption (FDE)** - For a computer that is not booted, all the information encrypted by FDE is protected, assuming that pre-boot authentication is required. When the device is booted, then FDE provides no protection; once the OS is loaded, the OS becomes fully responsible for protecting the unencrypted information. FDE can be achieved with a Trusted Platform Module (TPM).
- **Virtual Disk and Volume Encryption** - When virtual disk encryption is employed, the contents of containers are protected until the user is authenticated. If single sign-on is being used for authentication to the solution, this usually means that the containers are protected until the user logs onto the device. If single sign-on is not being used, then protection is typically provided until the user explicitly authenticates to a container.
- **File/Folder Encryption** - File/folder encryption protects the contents of encrypted files (including files in encrypted folders) until the user is authenticated for the files or folders. If single sign-on is being used, this usually means that the files are only protected until the user logs onto the device. If single sign-on is not being used, then protection is typically provided until the user explicitly authenticates to a file or folder.

2.3.1 Mobile Data Storage

This section describes the storage encryption techniques that are used in both known mobile devices Android and iOS. In order to provide confidentiality, medical data must be encrypted before it is stored on

the mobile phone or any other device. As aforementioned, symmetric encryption enables the data to be securely stored in an efficient manner.

- **Android Data Storage** - Android supports two major categories for storage encryption: full-disk encryption (FDE) and file-based encryption (FBE). In Android versions 5.0 up to 9.0 FDE is supported and is enabled by default with the use of Advanced Encryption Standard (AES) algorithm **[androidd2020]**. For Android version 7.0 or later, FBE is supported too. FBE has the ability to encrypt different files with different keys and hence each file can be decrypted independently **[androidf2020]**. FBE keys, which are 512-bit keys, are stored encrypted by another key (a 256-bit AES-GCM key) held in the Trusted Execution Environment (TEE) **[androidf2020]**.
- **iOS Data Storage** - Apple automates by default the FBE encryption process of an iPhone from version 8 and above **[kaspersky]** with a 256-bit AES encryption **[applesec]**. The data stored on the phone is automatically encrypted through a unique identifier built into the device's hardware. In addition, all personal data are encrypted by default whenever the phone is locked, and it is necessary for the user to have a passcode or Touch ID enabled (i.e. their fingerprint) in order to prevent unauthorized access to data **[nield2020] [appledev]**.

2.3.2 Desktop Data Storage

This section describes the storage encryption techniques that are used for both database and disk storage. The first subsection describes the technologies that are used for the encryption of data in databases, both Structured Query Language (SQL) and NoSQL, since both HCP Apps and Cloud services use databases to store their data, and the second describes disk encryption techniques. In the context of InteropEHRate, we assume that common best practices, such as full disk encryption are enabled, but we will also provide application level encryption.

- **Database Encryption** - Structured Query Language (SQL) supports Transparent Data Encryption (TDE). TDE encrypts both the data and log files **[microsoftder2019]**. The encryption process is using either AES or Triple DES algorithm **[microsofttde2019]**. The process of encryption and decryption are real time and they are completely transparent to the applications that have access to these databases **[microsoftder2019]**. NoSQL databases, and specifically MongoDB, support data-in-motion encryption and the data-at-rest encryption **[Townsend]**. For data-in-motion encryption, both Transport Layer Security (TLS) and Secure Socket Layer (SSL) protocols are supported. For data-at-rest encryption, an AES 256-bit symmetric key encryption at the file level is used.
- **Full-Disk Encryption** - FDE is encryption at the hardware level, where the data is automatically written in encrypted form. When it is read, it is automatically decrypted. However, such an approach has the disadvantage of additional time overhead for accessing data.

2.3.3 Cloud Data Storage and Break-glass Encryption

Three types of cryptography are commonly used to secure EHRs: a) symmetric key cryptography, b) public key cryptography, and c) attribute-based encryption **[Madnani2013]**. "Break-glass" is a term used in IT healthcare systems in order to denote an emergency access to private information without having the credentials to do so **[Scafuro2019]**. Several works in the literature deal with the concept of break-glass encryption for cloud storage **[Scafuro2019] [Oliveira2020]**. Cloud services emerge as a promising solution to this problem by allowing ubiquitous access to information. However, Electronic Medical Records (EMR) storage and sharing through clouds raise several concerns about security and privacy.

Several studies propose to send the EMR to a cloud service provider, where it is stored and encrypted with an encryption key known by the cloud provider [Abbas2014]. However, this approach does not protect the medical data against internal attacks [Abbas2014]. The storage of sensitive data over the cloud requires cryptography techniques in order to keep data confidential and preserve patients' privacy. Moreover, various solutions, based on symmetric or public cryptography, have been proposed to provide cryptographic access controls that allow storage and sharing of data on untrusted servers [KRS+03] [GSMB03] [BCHL09] [dVFJ+07] [WLOB09]. However, these techniques do not support fine grained access control required by medical applications and are not scalable with the number of users and introduce high complexity in key distribution and management.

The work in [Li2010] proposes a unique authority to authenticate the medical staff to access the data. Other research works suggest encrypting the EMR with a secret key before storing it in the cloud [Zhang2010] [Mashima2012]. However, this means that the secret key needs to be pre-shared with all the legitimate users that need to access the EMR throughout the treatment, while in case of revoking the treatment process, the EMR must be re-encrypted with a new key and re-distributed to the legitimate users making the whole process not efficient [Oliveira2020]. Moreover, several works attempt to address access control of encrypted data by using secret sharing schemes combined with identity-based encryption [Benaloh1988] [Brickell1989]. However, such schemes do not address resistance to collusion attacks. A break-glass solution based on a password-based encryption and a master secret key-based encryption proposed in [Zhang2016]. The work in [Scafuro2019] proposed a solution where the security of the ciphertexts stored on a cloud can be violated exactly once, in a way that is detectable and without relying on a trusted third party, in case of secret keys lost.

Another approach is to use attribute-based encryption (ABE) techniques to control access to patients' data. In [6] [Brucker2010], the authors present an ABE-based break-glass access control. However, their solution does not enable revoking access after it is granted [Oliveira2020]. The authors in [Li2013] propose a patient-centric framework and a suite of mechanisms for data access control to PHRs stored in semi-trusted servers based on attribute-based encryption (ABE) techniques to encrypt each patient's PHR file. Their work also enables dynamic modification of access policies or file attributes, supports efficient on-demand user/attribute revocation and break-glass access under emergency scenarios.

Several works leverage techniques, such as Role Based Access Control (RBAC) and Attribute Based Encryption (ABE), to provide fine-grained access control required by personal medical systems. In research work [IAP09], applied Ciphertext Policy ABE (CP-ABE) is used to enable patients to securely store and share their health record on external third-party servers. In [LYRL10], authors proposed a novel practical framework for fine-grained data access control to medical data in Cloud. To avoid high key management complexity and overhead, they organized the system into multiple security domains where each domain manages a subset of users [Lounis2014]. The work in [Yang2019] presents a self-adaptive access control scheme for healthcare by combining attribute-based encryption (ABE) and a password-based break-glass key, which is pre-set by the patient. A contact holds this key for emergency situations when break-glass access has to be activated. More recently, the work in [Oliveira2020] proposes the usage of the ciphertext-policy ABE (CP-ABE) associated with policies defined for emergency situations, based on the research [Bethencourt2007], as well as the usage of an authentication token to grant and revoke access dynamically without the need to re-encrypt the patient EMR. In the context of InteropEHRate, we will combine symmetric key encryption for medical data and CP-ABE for symmetric key encryption.

2.3.3.1 *Confidential Computing*

Public cloud systems are the de facto platform of choice to deploy online services. As a matter of fact, all the major IT players provide some form of "infrastructure-as-a-service" (IaaS) commercial offerings, including Microsoft, Google and Amazon **[Göttel2019]**. IaaS infrastructures allow customers to reserve and use (virtual) resources to deploy their own services and data. These resources are eventually allocated in the form of virtual machines, containers or bare metal instances over the cloud provider's hardware infrastructure **[Göttel2019]**. However, privacy concerns have greatly limited the deployment of systems over public clouds. The recent introduction of new hardware-assisted memory protection mechanisms inside x86 processors paves the way to overcome the limitations **[Göttel2019]**.

Confidential computing refers to performing computations with additional data confidentiality and integrity guarantees. TEEs have recently emerged as one of the most flexible and mature technologies, which can enable confidential computing. Many of today's leading technology companies are actively developing and promoting confidential computing technologies **[CCC2020]**. Different TEE implementations vary in terms of features. The two most well-known TEE technologies are Intel SGX and AMD SEV.

- **The Intel Software Guard Extension (SGX) [Pires2019]** is primarily conceived for shielding micro-services, so that the trusted code base would be minimised. Automatic memory encryption and integrity protection are performed by hardware over a reserved memory area fixed at booting time, defined in the basic input/output system (BIOS) and limited to 128MiB (usable 93.5MiB). Whatever is kept in this area is automatically encrypted and integrity checked by hardware. The trust boundary is the CPU package, which holds hardware keys upon which attestation and sealing services are built. Applications are partitioned into trusted and untrusted parts, while the OS is considered untrusted.
- **AMD secure encrypted virtualisation (SEV) [Pires2019]** provides automatic inline encryption and decryption of memory traffic, granting confidentiality for data in use by virtual machines. Cryptographic operations are performed by hardware and are transparent to applications, which do not need to be modified. Keys are generated at boot time and secured in a coprocessor integrated into the System on Chip (SoC). It was conceived for cloud scenarios, where guest VMs might not trust the hypervisor. Apart from including the whole guest OS in the trusted code base, it does not provide memory integrity and freshness guarantees as Intel SGX.

In general, Intel SGX focuses on micro services, while AMD SEV is designed for cloud. AMD SEV offers better performance for intensive workloads and is transparent to the software running in an SEV-enabled VM. Both Microsoft and Amazon offer confidential computing based on Intel SGX, while Google very recently announced such a feature based on the AMD SEV **[Google2020]**. Table 1 below summarizes these two known TEE technologies used for confidential computing **[Pires2019]**.

Commercial TEE Technologies	Intel SGX	AMD SEV
Public Cloud provider announcements	Microsoft Azure Confidential Computing (2018) & Amazon AWS Nitro Enclaves (2019)	Google Confidential Virtual Machines (2020)
Released	2015	2016

Target devices	Client PCs	Servers
Running mode	User-level	Hypervisor
Executes arbitrary code	Yes	yes
Secret hardware key	Yes	yes
Attestation and Sealing	Yes	yes
Memory encryption	Yes	yes
Memory integrity	Yes	no
Resilient to wiretap	Yes	yes
I/O from TEE	No	no
TEE usable memory limit	93.5MiB	system RAM
Trusted Computing Base	Trusted app partition	Entire VMs

*Table 1 - Comparison of TEEs [**Pires2019**]*

2.4 InteropEHRate Technologies

This section is in alignment with **[D3.1]** summarises the technologies used for encryption for both data at rest and data in transit. As already referred to **[D3.1]** Data-at-rest SHOULD be symmetrically encrypted using a military-grade NIST-compliant algorithm (e.g. AES with 256bit key) and the symmetric Key SHOULD be stored and retrieved by a local Keystore. In addition, encryption in transit SHALL be used with both secure-key-exchange (e.g. Diffie-Hellman key exchange) and strong network-level encryption. Table 2 below summarises the aforementioned used security enablers.

	Data in Storage	Data in Transit
D2D	**S-EHR App**: Symmetric Encryption / Database Encryption	Diffie-Hellman Key Agreement / Symmetric Encryption
R2DAccess	**S-EHR App**: Symmetric Encryption / Database Encryption	TLSv1.2[1]

[1] https://datatracker.ietf.org/doc/html/rfc5246

R2D Backup	**S-EHR Cloud**: Symmetric Encryption / File/Folder Encryption	Symmetric Encryption
R2D Emergency	**S-EHR Cloud**: Symmetric Encryption / File/Folder Encryption	Symmetric Encryption
RDS	**S-EHR App**: Symmetric Encryption / Database Encryption	Diffie-Hellman Key Agreement / Symmetric Encryption

Table 2 - InteropEHRate technologies

3 INTEROPEHRATE SPECIFICATION OF ENCRYPTION MECHANISMS

The purpose of this section is to show how the InteropEHRate project will handle encryption/decryption mechanisms for data storage and data exchange in the context of InteropEHRate protocols and use cases. The following subsections include crypto models for data encryption mechanisms for all communication channels and involved applications. An overview of how the different actors and organizations involved in the InteropEHRate architecture in **[D2.6]** interact with each other in the context of the encryption mechanisms depicted in Figure 1. More specifically, in the context of interoperate data data-at-rest should be symmetrically encrypted using a military-grade NIST-compliant algorithm (e.g. AES with 256bit key), while the symmetric-encryption key (that is used for data-at-rest) should be stored and retrieved by a local KeyStore (password-protected or biometric protected). In addition, apart from application-level encryption, transport-level encryption shall be used such as TLS v1.2 which incorporates both secure-key-exchange and strong network-level encryption (e.g. Diffie-Hellman key exchange and RSA-based encryption).

InteropEHRate architecture involves the following communication protocols: the device-to-device (D2D), the remote-to-device Access (R2D Access), the remote-to-research Access (R2R-Access) which is similar to R2D Access as an optional extension of the RDS protocol, the remote-to-device Backup (R2D Backup), the remote-to-device Emergency (R2D Emergency) and the research data sharing (RDS). In the context of R2D Access, R2D Backup, R2D Emergency, R2R Access and RDS Research, TLS 1.2 should be enabled for encrypted communication. In the context of D2D, the AES Bluetooth encryption should be enabled. In addition, application level symmetric encryption will be used in cases where the TLS 1.2 and Bluetooth encryption are missing or not enabled. This deliverable will focus on application level encryption specifications. The following sections describe the specified APIs for the data encryption mechanisms. In addition, from the Figure 1 below, we can have an overview all the different communication channels and the involved applications where sensitive medical data are transferred and stored.

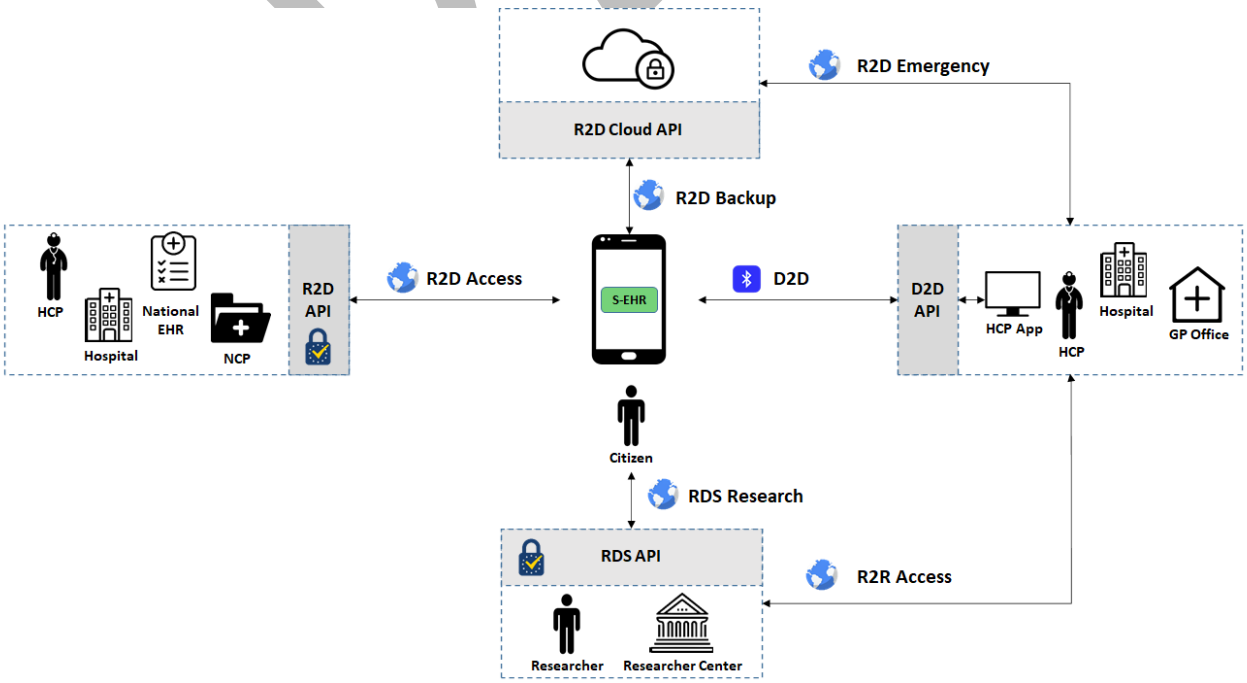


Figure 1 – InteropEHRate protocols

3.1 D2D Security Architecture and Models

The D2D protocol defines the set of operations that allow the exchange of health data between a S-EHR app and an HCP app in short-range distance over Bluetooth, without the usage of Internet connection **[D4.3]**. This section describes the security models in the context of D2D. Prior to any security operation, the bootstrap phase will take place in order for all the participants in the protocol to agree on the necessary elements and acquire the needed Certificates as the necessary step to all the Public Key Infrastructure (PKI) frameworks. This prerequisite phase applies to all scenarios if certificates are missing and an Internet connection is necessary. These steps will be described in the current section and will not be described again in the rest of the scenarios since they are the same. In addition, section 3.11 below summarises all the security common APIs including the interaction with the CA in order to retrieve the necessary certificates, certificate chain and validate a certificate.

Even though the interfaces are not depicted in the security models we refer to them for easier reference on the architecture of the reader. The name of the interface that is offered to the HCP app regarding the D2D protocol is named D2D. This interface contains the operations for letting the HCP app to perform tasks related to the S-EHR app, by invoking these operations, while the D2DServerSecurity and the D2DClientSecurity interfaces contain the operations for letting the HCP app and the S-EHR app establish a secure Bluetooth Connection **[D2.6]**. Both APIs will be used by the S-EHR and HCP app to agree on a shared secret key for the encrypted communication. The reader can also refer to Figure 13 of **[D4.3]**.

In the D2D protocol, we have two principals, the $S-EHR\ App$ and the $HCP\ App$, that agree publicly on an element g that generates a multiplicative group G. The group G is a subgroup ofZ_p^* of prime order q, p is a large prime and $g$is a generator of the group Z_p^* of order m. Typical sizes in use today are 1024 bits for the length of p and 160 bits for the length of q. The two principals select random values, r_A and r_B respectively, in the range between 1 and the order of G. $S-EHR\ App$ calculates $t_A = g^{r_A}$and $HCP\ App$ calculates $t_B = g^{r_B}$ and they exchange these values (public keys) as included in the corresponding Certificates. In order to generate the Certificates, both parties share their public keys to the CA, in order to issue the Certificates (i.e. C_A, C_B). Each issued Certificate (in our case the X.509) contains information regarding the identity of each party, the corresponding public keys t , while it is digitally signed by the CA's Certificate (i.e.C_{CA}). Each party can verify the Certificate signature (when it is necessary) with the root CA's Certificate. In this deliverable, we omit the steps of the identity and consent management that happen before the encryption mechanisms, since they are part of other specification deliverables (**[D3.4]** and **[D3.8]**), and we will focus only on the encryption/decryption aspects.

The encryption (i.e. Enc function) in the communication channel is performed with the symmetric key Z_{AB}. Each party can calculate the symmetric key with the private values, r_A and $r_B$and the public values, t_B and t_A. It has to be noted here that t_B is already contained in the Certificate of $HCP\ App$ and hence the $S-EHR\ App$ has access and $t_A$is already contained in the Certificate of $S-EHR\ App$ and hence $HCP\ App$has access. More specifically, $S-EHR\ App$ calculates $Z_{AB} = {t_B}^{r_A}$ and $HCP\ App$ calculates $Z_{AB} = {t_A}^{r_B}$.This is the last step of the well known Diffie Hellman key agreement protocol. For storage each party encrypts/decrypts (i.e. Enc and Dec functions) the stored data with a newly generated symmetric key $Z_A$and $Z_B$respectively from a Key Derivation Function (KDF) for a high-entropy key. In other words, the shared secret for encrypted communication is $Z_{AB} = g^{r_A r_B}$. This value can be calculated as described earlier by both $S-EHR\ App$ and $HCP\ App$ due to the homomorphic property of exponentiation: $Z_{AB} = t_A^{r_B} = t_B^{r_A}$. Figure 2 below depicts the described crypto model.

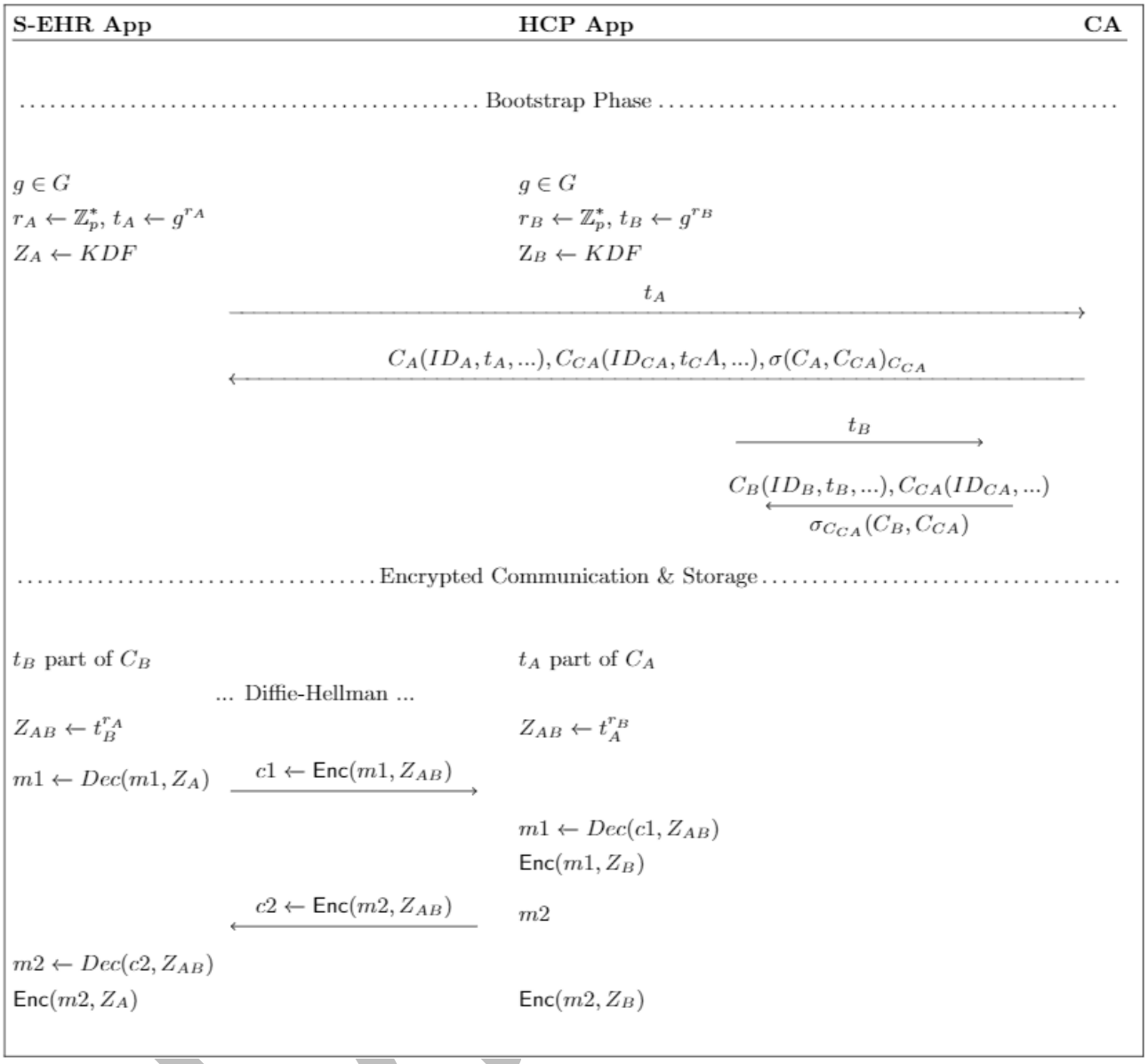


Figure 2 – D2D crypto-model

The conceptual sequence diagram that provides a high-level overview of the D2D is depicted in Figure 3, where the Diffie-Hellman key agreement protocol is hidden behind the exchanged steps. Following a detailed description of the sequence diagram of D2D to achieve encrypted communication:

- **Step 1**: This step is part of the Diffie-Hellman key agreement protocol. More specifically, the HCP app is the initiator of the protocol to agree both the HCP and the S-EHR in secret keys for symmetric key encryption/decryption. The step demonstrates the transfer of the HCP's public part of the agreement.
- **Step 2**: This step is part of the Diffie-Hellman key agreement protocol. The step demonstrates the transfer of the S-EHR's app public part of the agreement. After this step both parties calculate and

agree on a common symmetric key and hence are able to encrypt/decrypt the communication channel.

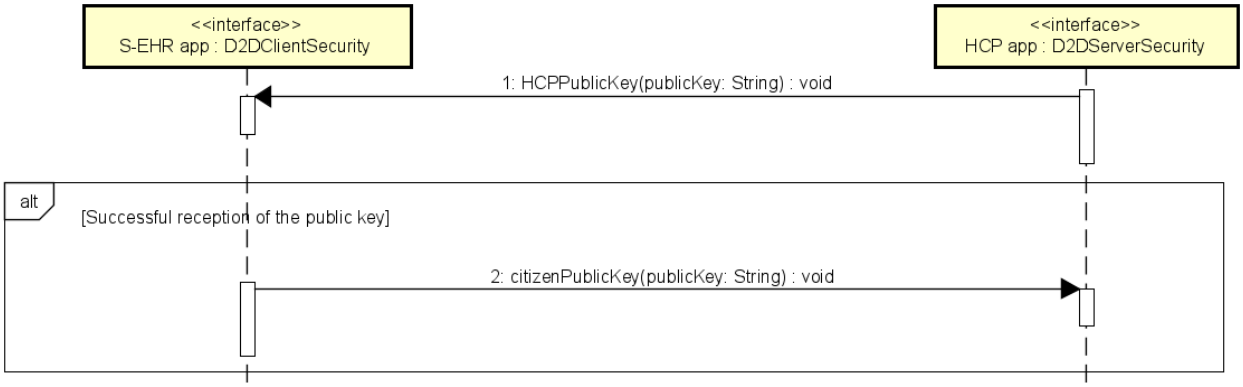


Figure 3 – D2D sequence diagram

3.2 D2D Security APIs

This section includes the security APIs for the secure communication in the D2D protocol. The HCP Web App initiates the procedure of the Diffie-Hellman key agreement, while both the S-EHR App and HCP Web App provide the functionalities of symmetric encryption and decryption for the D2D protocol.

3.2.1 S-EHR App Security APIs

Operation HCPPublicKey

Name	**HCPPublicKey**
Description	HCP sends his/her public key to the S-EHR App.
Arguments	• String publicKey
Return Value	• void
Exceptions	• Exception
Preconditions	• Successful acquisition of public key

3.2.2 HCP App Security APIs

Operation citizenPublicKey

Name	**citizenPublicKey**
Description	S-EHR app sends his/her public key to the HCP.

Arguments	• String publicKey
Return Value	• void
Exceptions	• Exception
Preconditions	• Successful acquisition of public key

3.3 R2D Access Security Architecture and Models

The R2D Access protocol is used for importing health records stored within an EHR of a healthcare organization to a smart mobile device **[D4.3]**. As already written, prior to any security operation a bootstrap phase is necessary in order for all the participants in the protocol to acquire the necessary elements. Regarding R2D Access, all the involved parties should have acquired the necessary Certificates from the CA, including the Certificates of the eIDAS Node C_E, the Certificate of the S-EHR App $C_A$and the Certificate of the Healthcare EHR C_L. Each certificate is associated with a private and a public key. For instance, in the S-EHR app with the private $Pr_A$and the public Pub_A. eIDAS defines citizens as persons and organisations that seek online services from any EU member state using their domestic eID with assured security, cost- and time-efficiencies, and usability **[KENNEDY2016]**. The proprietary national input is mapped and conditioned through the eIDAS Node in Country-A to an interoperable transport form, the eIDAS SAML Assertion. Such assertions can be requested during an authentication request by a Service Provider (SP) through an eIDAS Connector in Country-B **[ESENS2017]**. More details regarding eIDAS and eIDAS-based identification are provided in **[D3.4]**.

Both the Certificates and a successful eIDAS-based authentication must be retrieved in order to achieve all the necessary crypto operations. In section 3.1 above is explained how the Certificates acquired from a CA, while the detailed eIDAS authentication steps are included in **[D4.3].** After a successful cross-border authentication with the eIDAS infrastructure an authentication token with a certain validity period is stored in the mobile for future usage. Figure 4 depicts the R2D Access crypto-model for data encryption mechanism. After successful authentication with the utilisation of the eIDAS Infrastructure, and a TLS handshake for key agreement, the encrypted communication is achieved symmetrically using the HTTPS protocol. The symmetric key established form the TLS handshake is depicted as $Z_{AB}$and the symmetric key used for data storage is depicted as Z_A. The latter is generated with a KDF function in the S-EHR app to achieve high entropy. Since encryption will be used at a network layer, no security APIs and sequence diagram will be provided for the R2D Access encryption mechanisms.

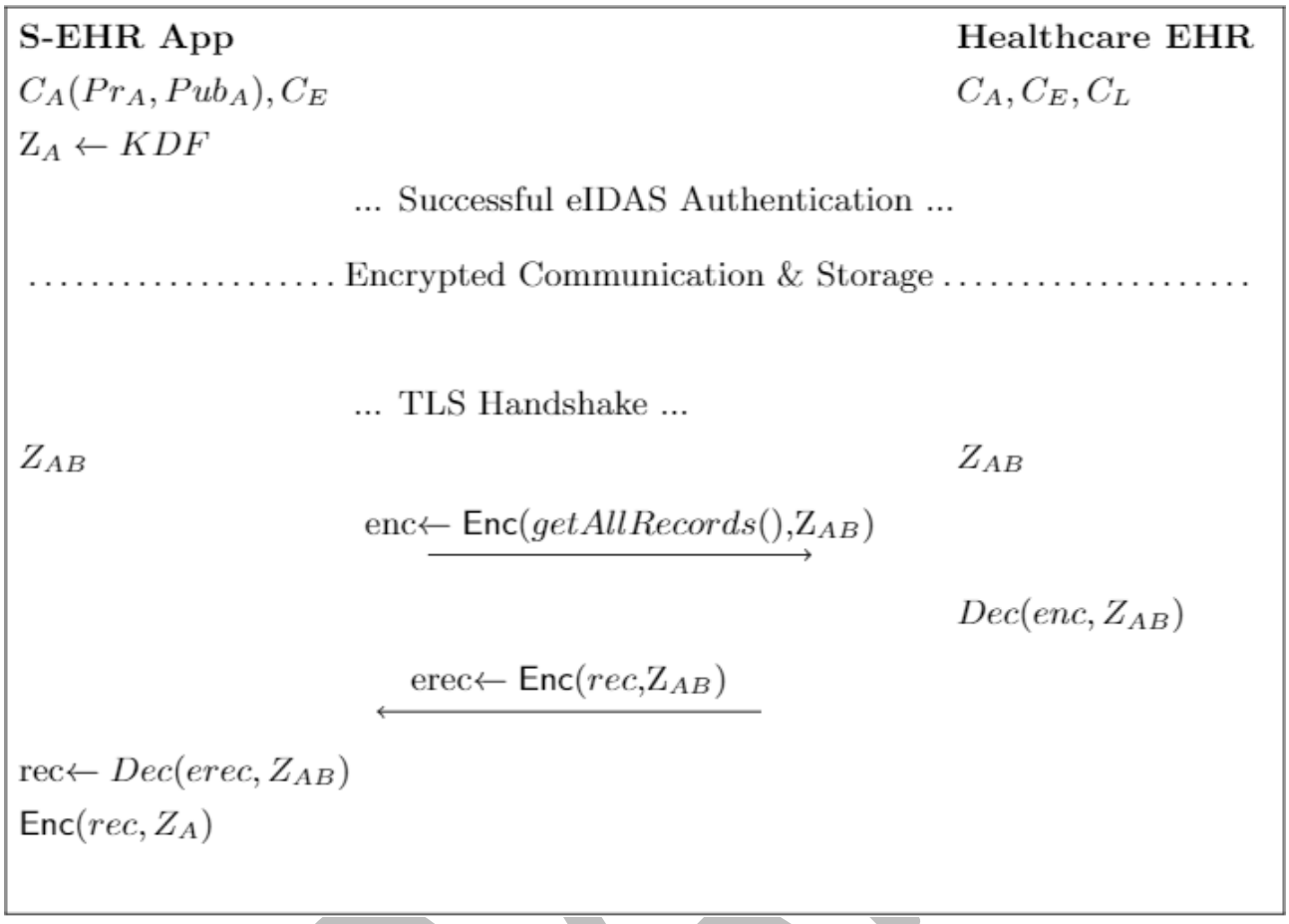


Figure 4 – R2D Access crypto-model

The TLS handshake[2] is depicted in Figure 5 and involves a series of steps, which accomplish the three main tasks a) exchanging encryption capabilities, b) authenticating the certificate, and c) exchanging/generating a session key. Handshake is a necessary step prior to the encryption in order to agree on a set of keys for both parties. The handshake can currently use 5 different algorithms to do the key exchange: RSA, Diffie-Hellman, Elliptic Curve Diffie-Hellman and the ephemeral versions of the last two algorithms. Following a detailed description of the TLS handshake steps:

- **Step 1**: This step is called the "Client Hello" and lists the client's capabilities so that the server can pick the cipher suite that the two will use to communicate. It also includes a large, randomly picked prime number called a client random or nonce.
- **Step 2**: The server responds with a "Server Hello" message, where it tells the client what connection parameters it has selected from the provided list and returns its own randomly selected prime number called a server random or nonce. If the client and server do not share any capabilities in common, the connection terminated unsuccessfully.
- **Step 3**: In the "Certificate" message, the Server sends its certificate chain to the client. To provide authentication to the connection certificate is signed by a CA, which allows the client to verify that the certificate is legitimate. Upon receipt, the client checks the certificate's digital signature,

[2] https://datatracker.ietf.org/doc/html/rfc524

verifying the certificate chain, and any other potential problems with the certificate data (expired certificate, wrong domain name, etc).

- **Step 4**: In this step, the "ServerHelloDone" message informs the client that it has sent over all its messages.
- **Step 5**: The client then provides its contribution to the session key. The specifics of this step depend on the key exchange method that was decided on in the initial "Hello" messages.
- **Step 6**: The "ChangeCipherSpec" message lets the other party know that it has generated the session key and is going to switch to encrypted communication.
- **Step 7**: The "Finished" message is then sent to indicate that the handshake is complete on the client side. The Finished message is encrypted, and is the first data protected by the session key. The message contains the message authentication code (MAC) that allows each party to make sure the handshake was not tampered with.
- **Step 8**: Server decrypts the pre-master secret and computes the session key. Then it sends its "Change Cipher Spec" message to indicate it is switching to encrypted communication.
- **Step 9**: In the last step, the server sends its "Finished" message using the symmetric session key it just generated, it also performs the same check-sum to verify the integrity of the handshake.

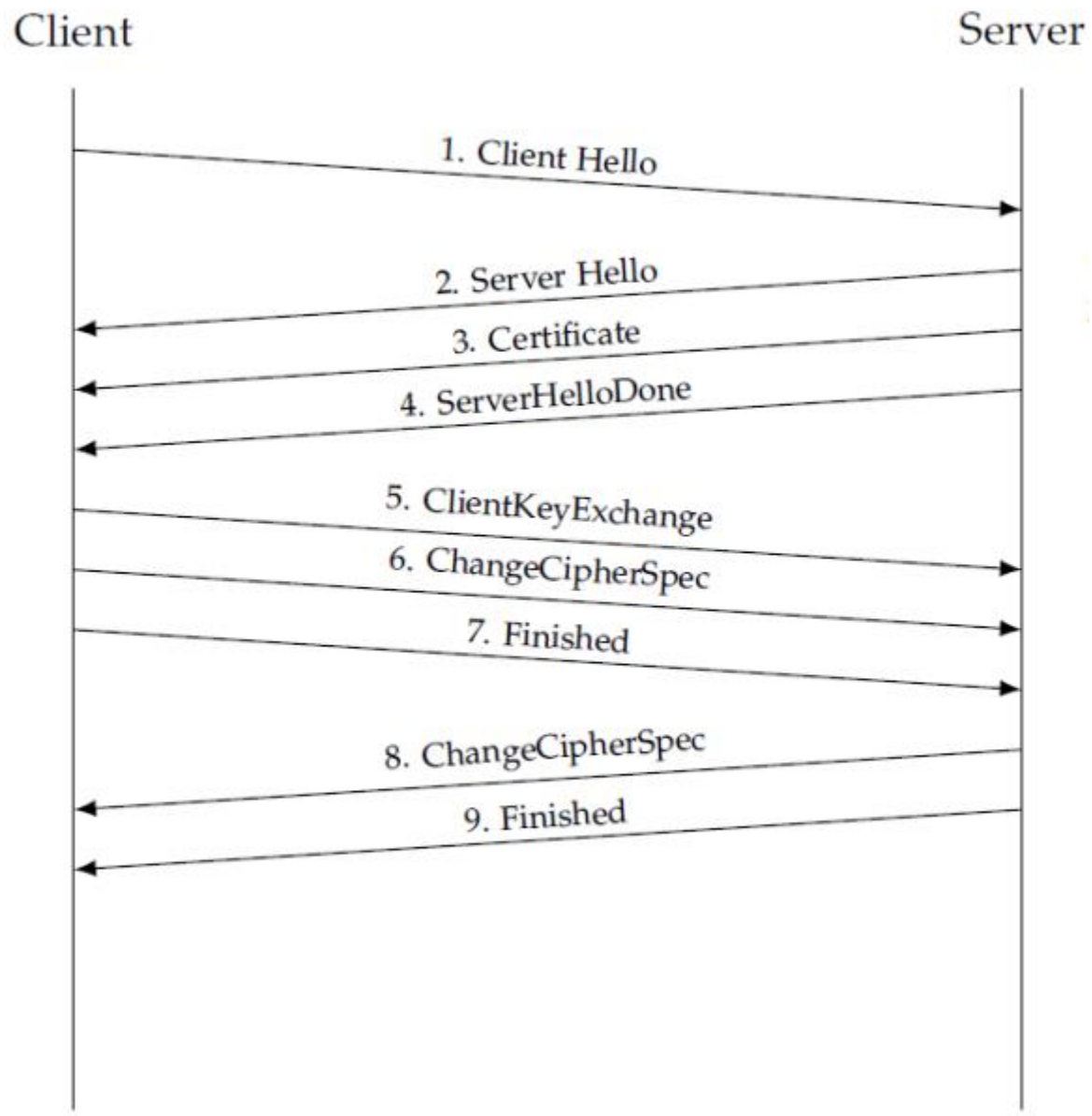


Figure 5 – TLS handshake protocol

3.4 R2D Backup Security Architecture and Models

The R2D Backup protocol, as its name suggests, defines the set of operations that allow backup of encrypted health data that is stored in a S-EHR mobile app to a S-EHR Cloud over the Internet **[D4.3]**. Prior to the encrypted communication and storage phase, a successful authentication of the S-EHR App to the S-EHR Cloud is needed. In this scenario, a simple username/password authentication mechanism is performed that returns a JWT authentication token. More information regarding the authentication aspects will be provided in the **[D3.4]**. In the S-EHR Cloud symmetric encryption will be used for secure transport and storage. More specifically, S-EHR App encrypts the data for backup with AES 256 and uploads them to the cloud for storage, after a successful authentication. The symmetric key is added in a QR code, in order the HCP could access it for emergency cases. The R2D Backup scenario requires the involved parties to have

acquired the necessary Certificates from the CA (i.e. C_A, C_C) for the necessary crypto operations. Figure 6 depicts the R2D Backup crypto model. The S-EHR App decrypts the encrypted stored data (i.e.Dec) with the key Z_A, re-encrypts (i.e.Enc) them with a different symmetric key generated from a KDF the Z_{AC} and stores the data in the selected S-EHR Cloud. Last but not least, this new generated key will be added in the printed QR code (i.e. $QR(Z_{AC}, \sigma_A(Z_{AC}))$ along with S-EHR App's signature for emergency cases, where authorised HCPs can have access and download the encrypted data and based on the scanned QR code to decrypt them.

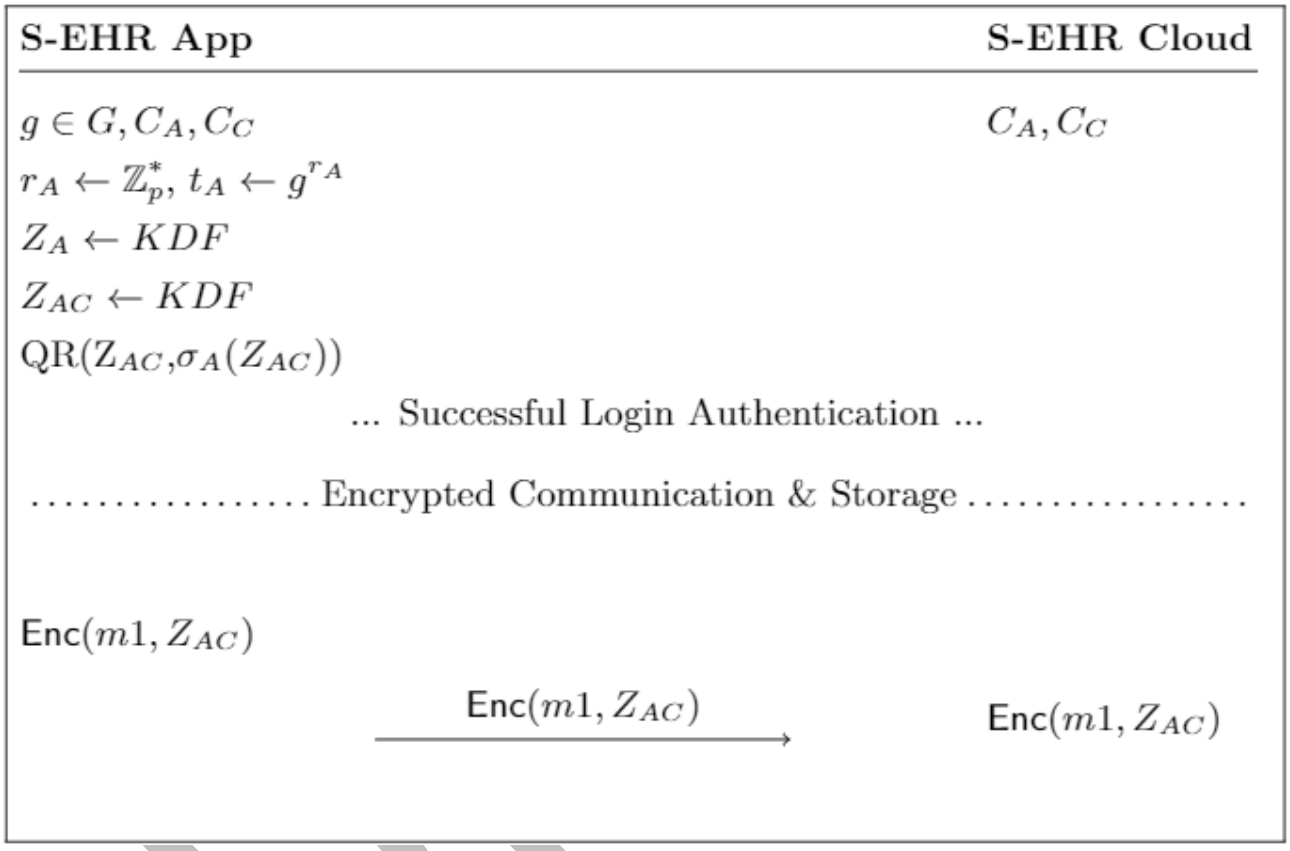


Figure 6 – R2D Backup crypto-model

The conceptual sequence diagram that provides a high-level overview R2D Backup is presented in Figure 7. The S-EHR App generates a symmetric key and a QR-code that includes the symmetric key used to encrypt his/her medical data prior to backup to the S-EHR Cloud. Following a detailed description of the sequence diagram of R2D Backup encryption mechanism:

- **Step 1-2**: These steps demonstrate secure upload of the health records. The encryption of the health records, registration and secure upload are performed on the selected S-EHR cloud by providing the encrypted data and the JWT token.
- **Step 3-4**: These steps demonstrate secure download of the health records. After the download of the health records decryption with the same symmetric key is needed to access the health records.

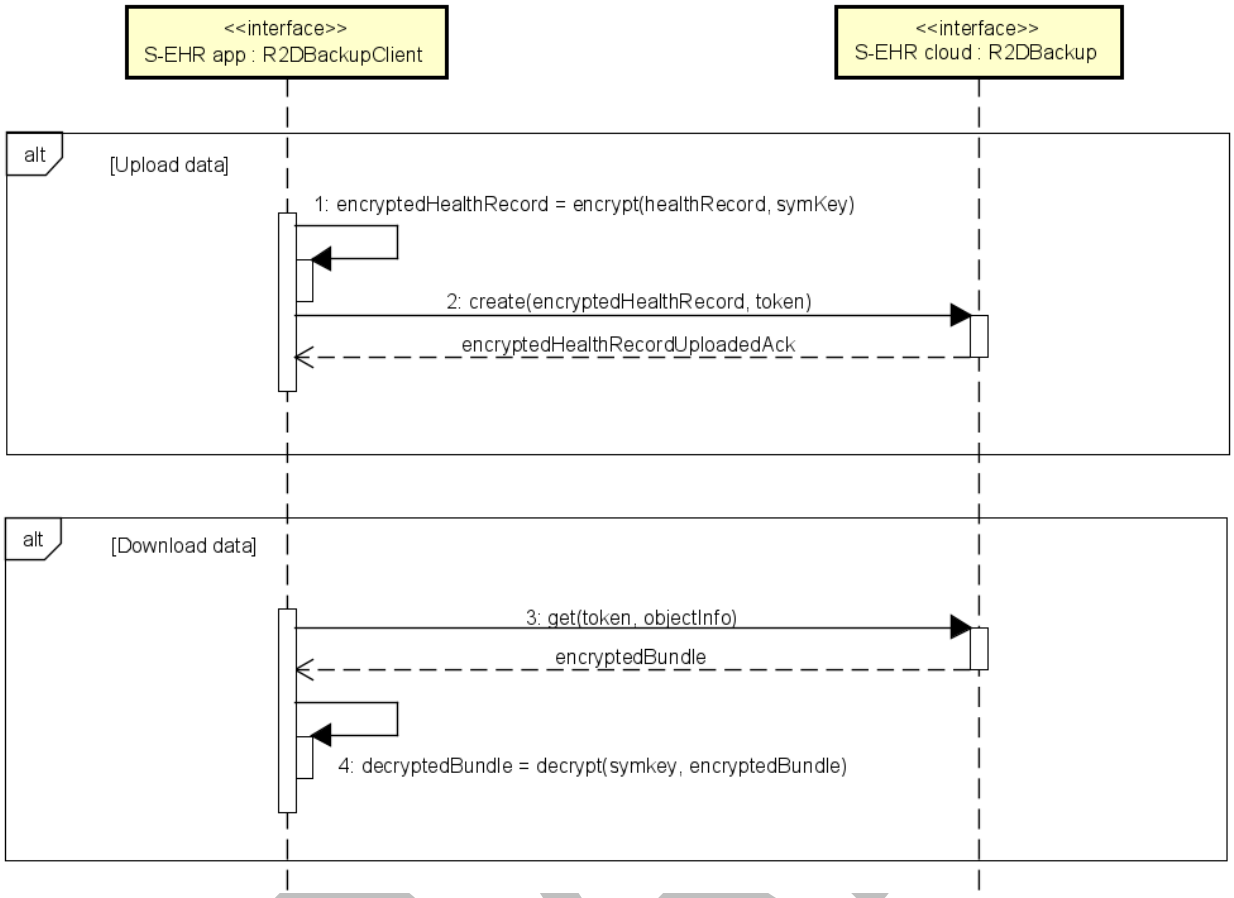


Figure 7 – R2D Backup sequence diagram

3.5 R2D Backup Security APIs

This section includes the security APIs for the encryption mechanism in the R2D Backup protocol. The S-EHR App provides the functionality of symmetric encryption for data in transit and in storage.

3.5.1 S-EHR Cloud Security APIs

Operation create

Name	**Create**
Description	This API is invoked by the S-EHR app to register and encrypted upload his/her health records. The citizen encrypts a health data resource on the S-EHR app and uploads it on the S-EHR cloud. This is a POST request to http://[baseurl]/citizen/upload?objectName={$ResourceCategory}
Arguments	• String encryptedHelathRecord: the encrypted payload. • String token: the JWT authorization token Header: "Authorization": Authentication token - JSON Web token

Return Value	• Acknowledgement: String HTTP Return Codes: 200 Successful: request was successfully processed. 400 Bad request: request could not be processed. 404 Not Found: User with that username is not found. 500 Internal Server Error: server encountered an unexpected internal error, the request could not be processed.
Exceptions	• Exception
Preconditions	• Successful JWT authorization • Symmetric key agreement established • Health Records are successfully encrypted

Operation get

Name	**Get**
Description	This API is invoked by the S-EHR app to download his/her encrypted health records. The citizen downloads an encrypted health data resource from the S-EHR cloud and decrypts it locally on the S-EHR app. This is a GET request to http://[baseurl]/citizen/{$bucketName}/{$objectName}
Arguments	• String token: the JWT authorization token • String objectInfo: information related to the requested record Header: "Authorization": Authentication token - JSON Web token
Return Value	• Encrypted Health Data Resource: EncryptedBundle HTTP Return Codes: 200 Successful: request was successfully processed. 400 Bad request: request could not be processed. 404 Not Found: User with that username is not found. 500 Internal Server Error: server encountered an unexpected internal error, the request could not be processed.
Exceptions	• Exception
Preconditions	• Successful JWT authorization

3.6 R2D Emergency Architecture and Models

The R2D Emergency protocol defines the set of operations that allow authorized HCPs to access the encrypted health data that is backed up on a S-EHR Cloud of a citizen in need during an emergency situation over the Internet **[D4.3]**. In this protocol, only authorised HCPs are allowed to access a citizen's health data. An Attribute Based Access Control (ABAC) mechanism will be utilised to make an access control decision based on the HCPs assigned attributes of the requester, the assigned attributes of the object (e.g.

health data), environment conditions (e.g. location, temperature etc.), and a set of policies that are specified in terms of those attributes and conditions. More details regarding the authorization phase are included in the **[D3.8]**. Figure 8 depicts the R2D Emergency crypto model. The HCP App will be able to acquire the symmetric key Z_{AC} after scanning the QR code (generated in R2D Backup protocol - Section 3.5). Finally, the HCP App downloads the encrypted data and decrypt (i.e. Dec) them using the acquired symmetric key Z_{AC}.

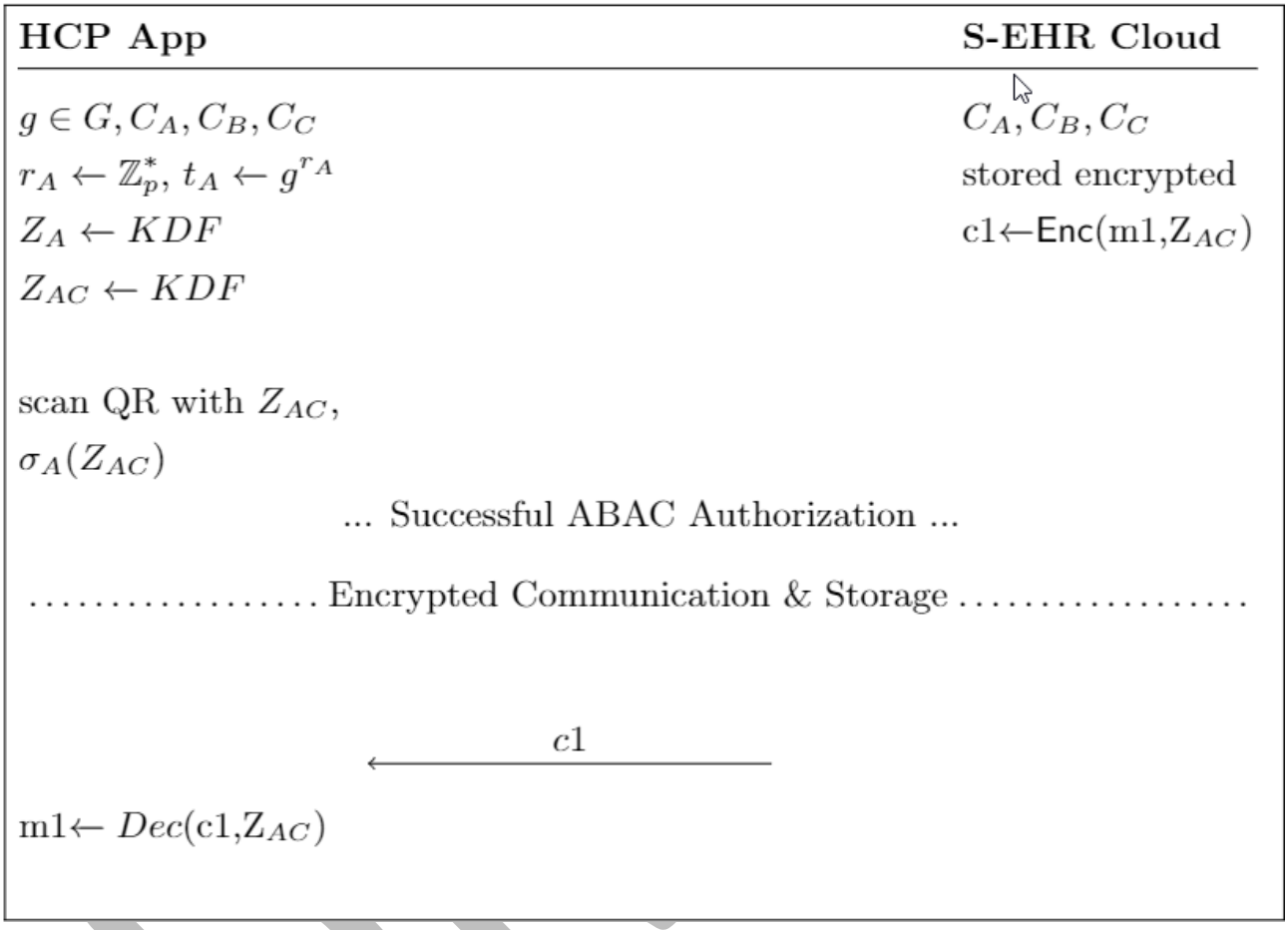


Figure 8 – R2D Emergency crypto-model

The conceptual sequence diagram that provides a high-level overview R2D Emergency is presented in Figure 9. In cases of emergency the HCP scans the QR-code to retrieve the symmetric key and after successful authorization to the cloud and downloads the encrypted data. Once the emergency occurs, the citizen is transferred to a healthcare facility. With the phone being unreachable the HCP that cures the citizen (from now on called patient), uses their HCP app to connect to the S-EHR Cloud service that the patient uses in order to access their health data. More details for the authorization aspects will be provided in **[D3.8]**. Following a detailed description of the sequence diagram of R2D Emergency encryption mechanism:

- **Step 1-2**: These steps demonstrate secure upload of the newly generated health records. The encryption of the health records and secure upload are performed with the symmetric key scanned from the QR.

- **Step 3-4**: These steps demonstrate secure download of the health records. After successful authorization and download of the encrypted data, the HCP can use the already scanned symmetric key to decrypt the data and get access to the needed patient's medical data.

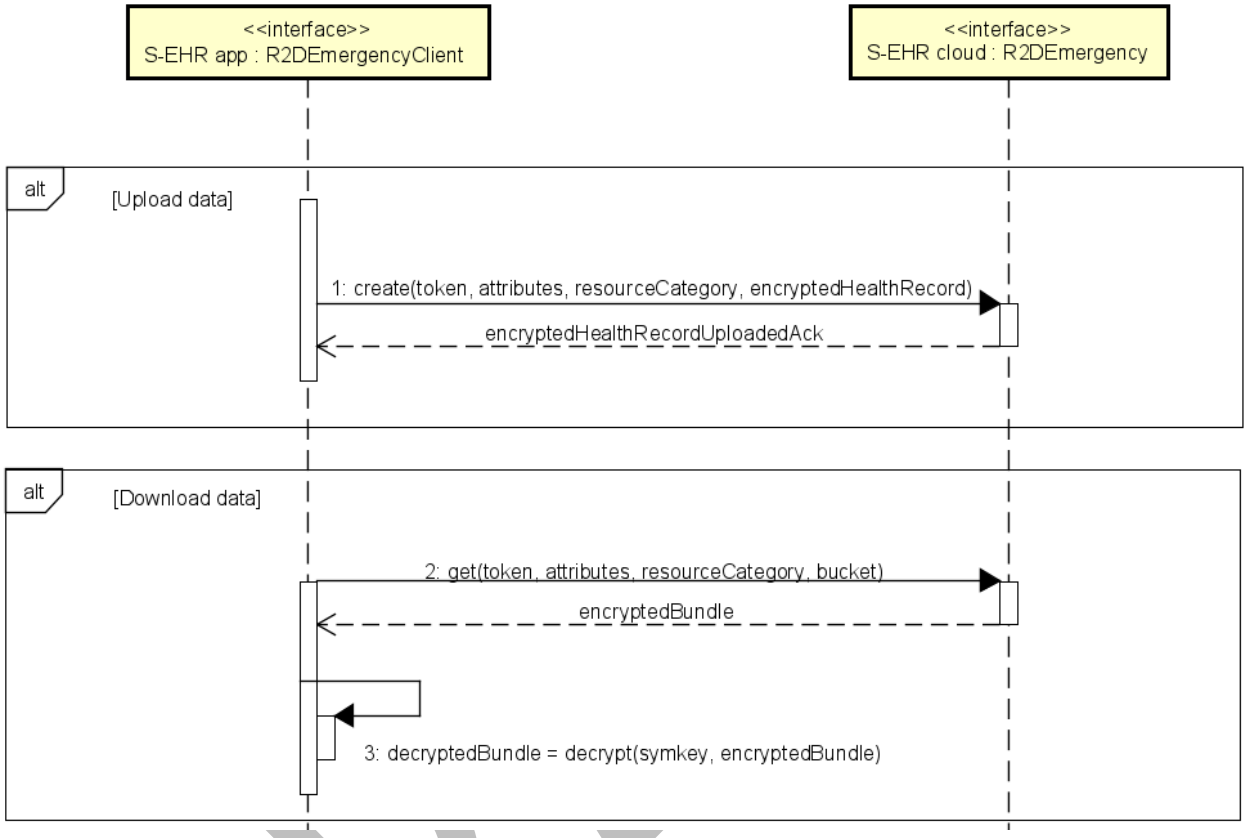


Figure 9 – R2D Emergency sequence diagram

3.7 R2D Emergency Security APIs

This section includes the security APIs for the encryption mechanism in the R2D Emergency protocol. The HCP Web App provides the functionality of decryption for the encrypted data.

3.7.1 S-EHR Cloud Security APIs

Operation create

Name	**create**
Description	This API is invoked by the HCP app to upload newly generated health records. The HCP encrypts a health data resource on the HCP app and uploads it on the S-EHR cloud using the scanned symmetric key. This is a POST request to http://[base url]/hcp/upload?objectName={$ResourceCategory}
Arguments	• HCP attributes: String • Encrypted health data resource: EncryptedBundle • Health data type: ResourceCategory Header: "Authorization": Health care institution authentication token: JSON Web token

Return Value	• Acknowledgement: String HTTP Return Codes: 200 Successful: request was successfully processed. 400 Bad request: request could not be processed. 404 Not Found: User with that username is not found. 500 Internal Server Error: server encountered an unexpected internal error, the request could not be processed.
Exceptions	• Missing header: Health care institution authentication token • Missing argument: HCP attributes • Missing argument: Encrypted health data resource • Missing argument: Metadata
Preconditions	• Symmetric key agreement established • Health Records are successfully encrypted • The citizen should have agreed to share their health data with authorized HCPs during emergency situations. • The health care institution should have already been granted access.

Operation get

Name	**get**
Description	This API is invoked by the HCP app to download citizen's encrypted health records for emergency purposes. The authorized HCP downloads an encrypted health data resource from the S-EHR cloud and decrypts it locally on the HCP app. This is a GET request to http://[baseurl]/hcp/{$bucketName}/{$objectName}
Arguments	• HCP attributes: String • Health data type: ResourceCategory • Bucket containing the object: String Header: "Authorization": Health care institution authentication token: JSON Web token
Return Value	• EncryptedBundle: The health data resource requested HTTP Return Codes: 200 Successful: request was successfully processed. 400 Bad request: request could not be processed. 404 Not Found: User with that username is not found. 500 Internal Server Error: server encountered an unexpected internal error, the request could not be processed.
Exceptions	• Missing header: Health care institution authentication token • Missing argument: HCP attributes • Missing argument: Encrypted health data type

	• Missing argument: Bucket name
Preconditions	• The citizen should have agreed to share their health data with authorized HCPs during emergency situations. • The health care institution should have already been granted access.

3.8 RDS Security Architecture and Models

The RDS protocol specifies the technical means to citizens for the sharing of their health data for the purposes of cross-border medical research, in a cross-border interoperable manner **[D4.9]**, while in some cases (e.g. data not able to be stored in the mobile) gives the ability to the research centers to download in a secure and anonymous manner. In terms of privacy preservation, two variants are used **[D6.8]**, one standardized with the state-of-the-art crypto primitives for enriching privacy and one with the currently adopted mechanisms by the end-users. More specifically, the first variant is the pseudo-identity, which is generated at the RRC if the citizen gives his/her consent to participating in the study. The second variant is the pseudonym which is generated by the Pseudonym Provider and leverages an eIDAS-based architecture for cross-border identification/authentication to the PP. More information regarding these aspects will be provided in **[D6.8]** and **[D3.4]**. As already stated in **[ENISA2021]**, there is no fit-for-all pseudonymisation technique and a detailed analysis of the case is necessary. The usage of the second variant does not enhance the applicability of the InteropEHRate framework but allows to perform a detailed investigation of new privacy-preserving enablers that can extend the state of the art and be potentially considered as a new standard. Pseudonymisation can go beyond hiding real identities and data minimisation into supporting the unlinkability **[ENISA2021]** making high entropy pseudonyms necessary.

As aforementioned, the encrypted communication between the S-EHR app and RRC is achieved with AES symmetric encryption, after the successful Diffie-Helman key agreement, while the encrypted communication between the RRC and the HCP After is achieved via TLS using the HTTPS protocol, after successful authentication. This section describes the security models in the context of RDS for encrypted communication between the S-EHR app and the RRC. As with the previous protocols, S-EHR App and RRC already have access to the needed Certificates (C_A, C_E, C_R). On demand prior to the RDS protocol, S-EHR App and RRC run the Diffie-Hellman key agreement phase to establish a shared key Z_{AR}(the same used in Figure 2). One of the two pid (pseudo-id - variant 1) or $pseu$(pseudonym - variant 2) based on the variant will be used to anonymize the data (i.e. An- anonymous signing) and the encrypted (i.e. Enc) with the agreed key is transferred to the RRC. The RRC decrypts (i.e. Dec) with the same key the data, verifies the anonymous signature (currently this is not depicted for space reasons - the certificate is assumed that is sent along with the anonymised data) and retrieves the anonymised data for further process and survey.

S-EHR App	RRC
C_A, C_E, T_{ID}	C_R
$g \in G, Z_A \leftarrow KDF$	$g \in G, Z_R \leftarrow KDF$
$r_A \leftarrow \mathbb{Z}_p^*$, $t_A \leftarrow g^{r_A}$	$r_R \leftarrow \mathbb{Z}_p^*$, $t_R \leftarrow g^{r_R}$
t_R part of C_R	t_A part of C_A
$Z_{AR} \leftarrow t_R^{r_A}$	$Z_{AR} \leftarrow t_A^{r_R}$
... Successful Pseudonymization ...	
..... Encrypted Communication & Storage	
... Diffie-Hellman ...	
$m1 \leftarrow \mathsf{Enc}(An(\text{data, pid or pseu})), Z_{AR})$ $\longrightarrow$	
	$An(\text{data, pid or pseu}))$ $\leftarrow Dec(m1, Z_{AR})$

Figure 10 – RDS crypto-model

The conceptual sequence diagram that provides a high-level overview of RDS in Figure 11, where the encryption communication flow is depicted. Following a detailed description of the sequence diagram of RDS encryption mechanism:

- **Step 1**: The step demonstrates the transfer of encrypted data from the S-EHR app to the RRC. The decryption process is taking place on the RRC side.

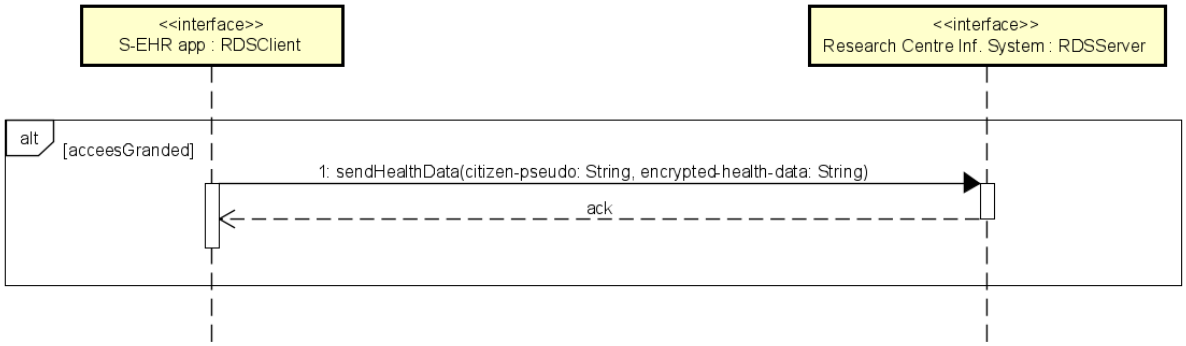


Figure 11 – RDS sequence diagram

3.9 RDS Security APIs

This section includes the security APIs for the encryption mechanism in the RDS protocol. The RRC initiates the procedure of the Diffie-Hellman key agreement, while the S-EHR App and RRC provide the functionalities of symmetric encryption and decryption for data in transit respectively.

3.9.1 RRC Security APIs

Operation sendHealthData

Name	**sendHealthData**
Description	This API allows a S-EHR App to send encrypted citizen health data to a Research Centre. The receiving RC verifies and decrypts the encrypted and signed payload healthData and retrieves the FHIR bundle contained within. This is a POST request to http://<BASE_ADDR>/sendHealthData?studyID=<studyID>

Arguments	URL params:
	studyID: the ID of the study in which the Citizen is enrolling;
	The POST body content is a JSON file defined as follows:
	{
	"citizen-pseudo": <citizen-pseudo>,
	"health-data": <health-data>
	}
	where:
	<citizen-pseudo>: the study-specific pseudonym or pseudo-identity of the Citizen;
	<health-data>: a FHIR bundle containing the health data (resources, attributes, values) necessary for the study, in an encrypted form, as well as the responses to research questionnaire(s) provided by the Citizen if available
	
	Header:
	Content-Type: application/fhir+json

Return Value	HTTP return codes: 200 Successful: request was successfully processed. 400 Bad Request: search could not be processed or failed basic FHIR validation rules. 401 Not Authorized: authorisation is required for the interaction that was attempted. 403 Forbidden: client is not allowed to access requested resources due to security policy. 404 Not Found: resource type not supported, or not a valid FHIR endpoint. 406 Not Acceptable: client requested a not supported content-type format. 500 Internal Server Error: server encountered an unexpected internal error, the request could not be processed.
Exceptions	The call's exceptions returned are added as text messages within the HTTP response body which is defined as follows: { "timestamp":<timestamp>,

	"status":<http-status-code>, "error":<code-description>, "message":<exception message>, "path":<request-path> } where: ● <timestamp> : response timestamp; ● <http-status-code> : one of the http codes listed in the previous row; ● <code-description> : description of http code; ● <request-path> : http request's URL; ● <exception message> : the following text message: ○ invalid content (study ID, pseudo-identity, healthData)
Preconditions	● The Citizen must have enrolled into the study previously. ● The S-EHR App must have access to the Citizen's private key to encrypt the health data, and the called Research Centre must have access to the Citizen's public key to be able to decrypt it.

3.10 Security Commons Architecture and Models

As already mentioned, prior to any security operation, the bootstrap phase will take place in order for all the participants in the protocol to agree on the necessary elements and acquire the needed X.509 Certificates. This prerequisite phase applies to all the scenarios if certificates are missing and an Internet connection is necessary. This section describes the security commons for all the protocols and more specifically the interaction with the CA and the corresponding APIs. The Certificate Authority provides the services exposed through the Certificate Authority Interface (CAI) **[EJBCA 2021]**, where CAI is a web service interface. In addition, since the eIDAS-based authentication will be adopted in the InteropEHRate project, a cooperation between these two services will benefit for a seamless Certificate generation based on the eIDAS acquired attributes of the eIDAS token $eidtkn$.

The Root CA is always a self-signed certificate C_{CA}. The root certificate, often called a trusted root, is at the center 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. As already introduced in **[D3.9]**, 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. The high-level steps in order for an entity (e.g. the citizen, the healthcare professional etc.) to retrieve a certificate is the following:

1. Entity (e.g. the citizen, the healthcare professional) generates a private/public key pair r_A/t_A, keeping the private key secret.
2. Entity crafts a certificate signing request (CSR) and submits it to the CA. CSR usually contains the public key t_A for which the certificate should be issued, identifying information $eidtkn\ and\ attr$ and integrity protection (e.g., a digital signature) σ_{r_A}. The most common format for CSRs is the PKCS #10 specification. The CSR may be accompanied by other credentials or proofs of identity required by the certificate authority.

3. If the request is successfully verified Ver, the CA issues a certificate that has been digitally signed using the private key of the CA and records it in the CA's database.
4. Entity verifies Verits own Certificate and presents the certificate to the another entity for identification purposes (e.g. S-EHR App to HCP App)
5. The other entity presumably has the signing certification authority's certificate or can get it if Internet connection is available. Then verifiesVer the validity of the Certificate.
6. The other entity checks that the certificate does not appear on the Certificate Revocation List (CRL) if Internet connection is available.
7. If 4, 5, and 6 all check out, the client will accept the certificate.

In case of D2D steps that require Internet connection will happen later in time, when Internet will be available. The reasoning behind separating the eIDAS token (Certificate form eIDAS) from the CA Certificate attributes, is that the eIDAS token includes a predefined attributes list, however, in the context of R2D Emergency more attributes are needed for access control purposes (e.g. the hospital name and the role). The eIDAS token will be used for identification purposes to verify and validate the identity of the requestor and the additional attributes will be used as extra information for the CA's X.509 Certificate generation.

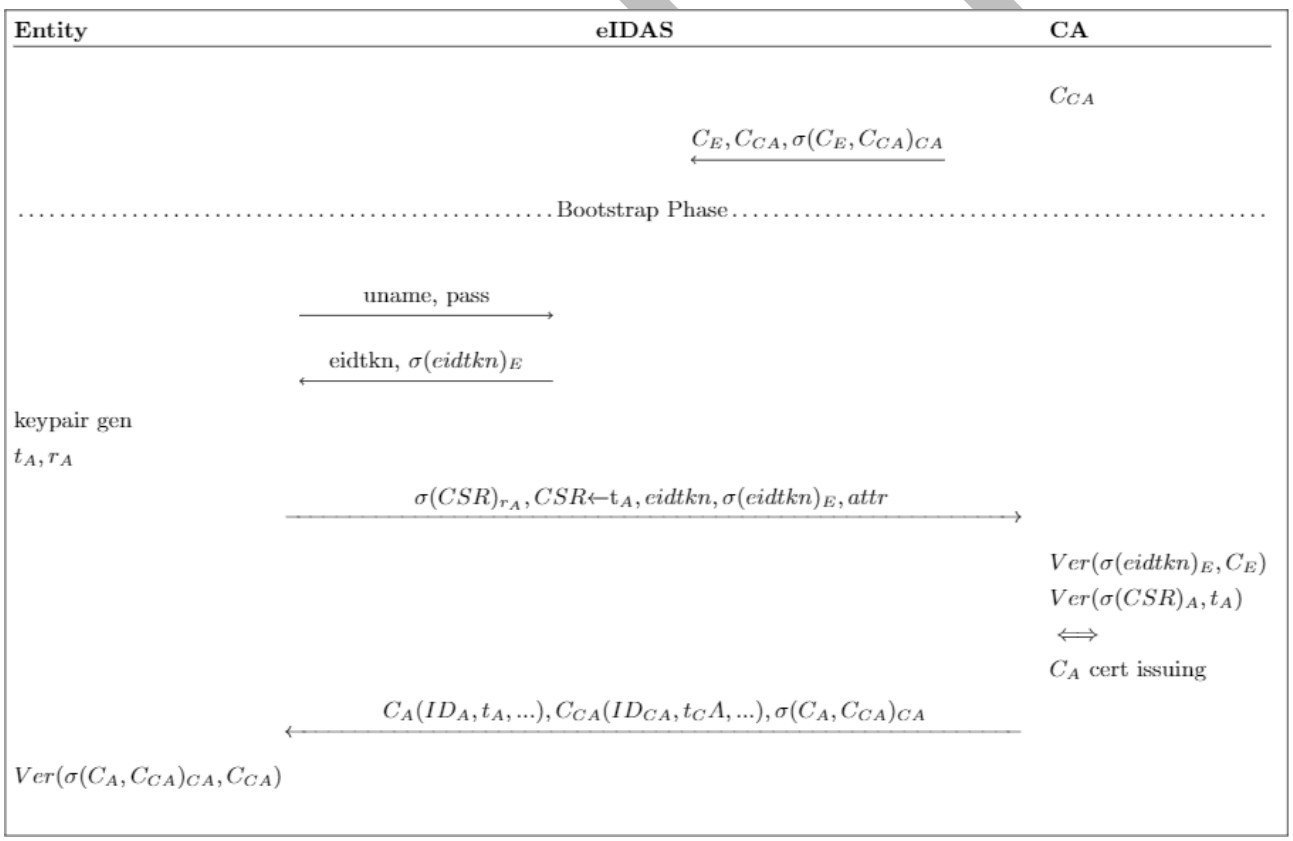


Figure 12 – Security commons crypto-model

The conceptual sequence diagram that provides a high-level overview of RDS in Figure 13, where common security flows are depicted. Following a detailed description of the sequence diagram:

- **Step 1:** An entity (e.g. S-EHR App, HCP App etc.) request from the CA to issue a certificate. In cases where the entity is an eIDAS registered, CA verifies the identity of the entity through the eIDAS infrastructure.

- **Step 2:** An entity (e.g. S-EHR App, HCP App etc.) request from the CA to reissue a lost certificate. In cases where the entity is an eIDAS register, CA verifies the identity of the entity through the eIDAS infrastructure.
- **Step 3:** An entity (e.g. S-EHR App, HCP App etc.) requests a certificate based on the alias and other optional attributes.
- **Step 4:** An entity (e.g. S-EHR App, HCP App etc.) checks the validity of a certificate.

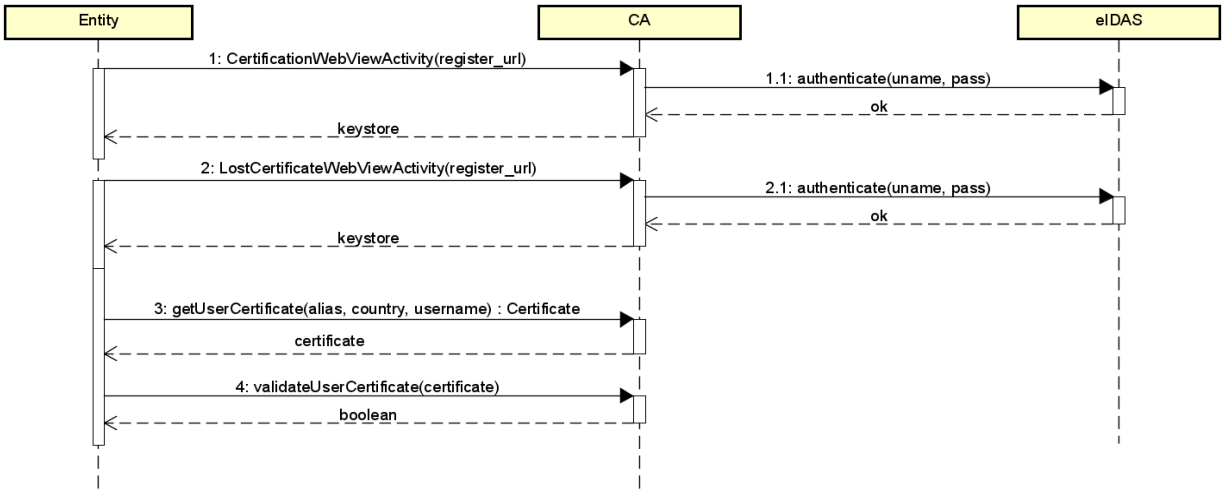


Figure 13 – Security commons sequence diagram

3.11 Security Commons APIs

Operation getUserCertificate

Name	**getUserCertificate**
Description	Retrieves a valid certificate generated for a user from the CA server.
Arguments	• String alias: User alias (e.g. GRxavi) • String country (optional) • String username (optional)
Return Value	• String Certificate data
Exceptions	• Exception, in case of error.
Preconditions	• CA server is available • An existing user and a server-generated keystore.

Operation validateUserCertificate

Name	**validateUserCertificate**
Description	Checks if certificate is valid
Arguments	• String certificateData

Return Value	• A boolean value, "true" meaning that the signature was successfully verified.
Exceptions	• Exception, in case of error.
Preconditions	• CA server is available

Operation CertificationWebViewActivity

Name	**CertificationWebViewActivity**
Description	This activity handles the redirection of the user to the eIDAS registration page hosted by the Trusted Proxy Server. The Trusted Proxy Server is responsible for managing all the communication between the eIDAS infrastructure and the S-EHR App.
Arguments	• String register_url: the register url of the Trusted Proxy Server. The user is redirected to this url in order to add his/her eIDAS credentials
Return Value	• String keystore: a keystore with eidas authentication
Exceptions	• N/A
Preconditions	• Trusted Proxy Server is available

Operation LostCertificateWebViewActivity

Name	**LostCertificateWebViewActivity**
Description	This activity handles the redirection of the user to the eIDAS lost certificate page hosted by the Trusted Proxy Server. The Trusted Proxy Server is responsible for managing all the communication between the eIDAS infrastructure and the S-EHR App.
Arguments	• String lost_certificate_url: the lost certificate url of the Trusted Proxy Server. The user is redirected to this url in order to add his/her eIDAS credentials
Return Value	• String keystore: a keystore with eidas authentication
Exceptions	• N/A
Preconditions	• Trusted Proxy Server is available

4 CONCLUSIONS

In this report, it's defined the second and final version of the specification of data encryption mechanisms for mobile and web applications. A technical background with state-of-the-art encryption mechanisms and crypto primitives are also provided. More specifically, this deliverable includes the detailed crypto models and encryption/decryption aspects of all the involved architecture components (e.g., S-EHR App, HCP Web App, S-EHR Cloud, Central Node and Reference Research Center), protocols (e.g., D2D, R2D Access, R2D Backup, R2D Emergency, RDS), and scenarios (e.g., Medical Visit, Emergency and Research) for data at rest and in-transit. Last but not least, the deliverable includes the detailed crypto models for the security common for all scenarios functionalities (e.g. interaction with a CA for certificate generation). This final version of the deliverable acts as the detailed specification of encryption and decryption purposes defined in the context of InteropEHRate.

REFERENCES

- **[Benaloh1988]** J. Benaloh and L. J. Generalized Secret Sharing and Monotone Functions. In Advances in Cryptology – CRYPTO, volume 403 of LNCS, pages 27–36. Springer, 1988
- **[Brickell1989]** E. F. Brickell. Some ideal secret sharing schemes. Journal of Combinatorial Mathematics and Combinatorial Computing, 6:105–113, 1989
- **[Bethencourt2007]** Bethencourt J, Sahai A, Waters B (2007) Ciphertext-policy attribute-based encryption. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy, SP'07. IEEE Computer Society, Washington, DC, pp 321–334.
- **[Zhang2010]** Zhang R, Liu L (2010) Security models and requirements for healthcare application clouds, in: 2010 IEEE 3rd International Conference on cloud Computing, IEEE, pp 268–275
- **[Li2010]** Li M, Yu S, Ren K, Lou W (2010) Securing personal health records in cloud computing: patient-centric and fine-grained data access control in multi-owner settings. In: International conference on security and privacy in communication systems. Springer, pp 89–106
- **[Brucker2010]** Brucker AD, Petritsch H, Weber SG (2010) Attribute-based encryption with break-glass. In: IFIP International Workshop on Information Security Theory and Practices. Springer, pp 237–244
- **[Mashima2012]** Mashima D, Ahamad M (2012) Enhancing accountability of electronic health 660 record usage via patient-centric monitoring, in: Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium, ACM, pp 409–418
- **[Abbas2014]** Abbas A, Khan SU (2014) A review on the state-of-the-art privacy-preserving approaches in the e-health clouds. IEEE J Biomed Health Inform 18(4):1431–1441
- **[Zhang2016]** Zhang T, Chow SS, Sun J (2016) Password-controlled encryption with accountable break-glass access. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. ACM
- **[Scafuro2019]** Scafuro A, Break-glass encryption. In: IACR International Workshop on Public Key Cryptography. Springer, pp 34–62, 2019.
- **[Yang2019]** Yang, Y.; Zheng, X.; Guo, W.; Liu, X.; Chang, V. Privacy-preserving smart IoT-based healthcare big data storage and self-adaptive access control system. Inf. Sci. 2019, 479, 567–592.
- **[Oliveira2020]** T. de Oliveira, M., Bakas, A., Frimpong, E. et al. A break-glass protocol based on ciphertext-policy attribute-based encryption to access medical records in the cloud. Ann. Telecommun. 75, 103–119 (2020).
- **[La2006]** Los Angeles Times, "At Risk of Exposure - in the Push for Electronic Medical Records, Concern Is Growing About How Well Privacy Can Be Safeguarded," 2006.

- **[Lisonek2008]** Lisonek, D. AND Drahansky, M. 2008. SMS Encryption for Mobile Communication. In SECTECH '08: Proceedings of the 2008 International Conference on Security Technology. IEEE Computer Society, Washington, DC, USA, pp. 198-201.
- **[Li2013]** Li, M., Yu, S., Zheng, Y., Ren, K., & Lou, W. (2013). Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption. IEEE Transactions on Parallel and Distributed Systems, 24(1), 131–143. doi:10.1109/tpds.2012.97
- **[androidd2020]** Android Open Source Project, "Full-disk Encryption", 2020. Web site: https://source.android.com/security/encryption/full-disk
- **[androidf2020]** Android Open Source Project, "File-based Encryption", 2020 Web site: https://source.android.com/security/encryption/file-based
- **[kaspersky]** Kaspersky, "iPhone Encryption: How to Encrypt Your iPhone". Web site: https://usa.kaspersky.com/resource-centerz/preemptive-safety/iphone-encryption
- **[applesec]** Apple Platform Security, "Encryption and Data Protection overview". Web site: https://support.apple.com/guide/security/encryption-and-data-protection-overview-sece3bee0835/1/web/1
- **[nield2020]** David Nield, "How to Get the Most Out of Your Smartphone's Encryption", 2020. Web site: https://www.wired.com/story/smartphone-encryption-apps/
- **[appledev]** Apple Developer Documentation, "Encrypting Your App's Files". Web site: https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files
- **[microsoftder2019]** Microsoft, "Data Encryption at Rest", 2019. Web site: https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/security/transparent-data-encryption
- **[microsofttde2019]** Microsoft, "Transparent Data Encryption (TDE)", 2019. Web site: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?view=sql-server-ver15
- **[Townsend]** Townsend Security, "The Definitive Guide to MongoDB Encryption & Key Management". Web site: https://info.townsendsecurity.com/mongodb-encryption-key-management-definitive-guide
- **[Lecroy]** Teledyne Lecroy, "How Encryption Works in Bluetooth". Web site: http://www.fte.com/webhelp/bpa500/Content/Documentation/WhitePapers/BPA600/Encryption/HowEncryptionWorks.htm
- **[Loveless2018]** Mark Loveless, "Understanding Bluetooth Security", 2018. Web site: https://duo.com/decipher/understanding-bluetooth-security
- **[Corella2015]** Francisco Corella, "Has Bluetooth Become Secure?", 2015. Web site: https://pomcor.com/2015/06/03/has-bluetooth-become-secure

- **[bon2016]** Matthew Bon, "A Basic Introduction to BLE Security", 2016. https://www.digikey.com/eewiki/display/Wireless/A+Basic+Introduction+to+BLE+Security
- **[Ravikiran]** Ravikiran HV, "Security Considerations For Bluetooth Smart Devices". Web site: https://www.design-reuse.com/articles/39779/security-considerations-for-bluetooth-smart-devices.html
- **[Prodromou2019]** Agathoklis Prodromou, "TLS Security 5: Establishing a TLS Connection", 2019. Web site: https://www.acunetix.com/blog/articles/establishing-tls-ssl-connection-part-5/
- **[Lake2019]** Josh Lake, "What is TLS and how does it work?", 2019. Web site: https://www.comparitech.com/blog/information-security/tls-encryption/
- **[Sullivan2018]** Nick Sullivan, "A Detailed Look at RFC 8446 (a.k.a. TLS 1.3)", 2018. Web site: https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/
- **[Kothari2019]** Kewal Kothari, "How does SSL/TLS make HTTPS secure?", 2019. Web site: https://hackernoon.com/how-does-ssl-tls-make-https-secure-d247bd4e4cae
- **[SW05]** Amit Sahai and Brent Waters. Fuzzy Identity-Based encryption. In Ronald Cramer, editor, EUROCRYPT, volume 3494 of Lecture Notes in Computer Science, pages 457473. Springer, 2005.
- **[BF03]** Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. SIAM J. Comput., 32(3) :586615, March 2003.
- **[KRS+03]** Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus, Scalable secure file sharing on untrusted storage. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pages 2942, Berkeley, CA, USA, 2003. USENIX Association.
- **[GSMB03]** Eu-jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. Sirius : Securing remote untrusted storage. Network and distributed systems security, NDSS'03, pages 131145, 2003.
- **[BCHL09]** Josh Benaloh, Melissa Chase, Eric Horvitz, and Kristin Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In Proceedings of the 2009 ACM workshop on Cloud computing security, CCSW '09, pages 103114, New York, NY, USA, 2009
- **[dVFJ+07]** Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Over-encryption : management of access control evolution on outsourced data. In Proceedings of the 33rd international conference on Very large data bases, VLDB '07, pages 123134, 2007.
- **[WLOB09]** Weichao Wang, Zhiwei Li, Rodney Owens, and Bharat Bhargava. Secure and efficient access to outsourced data. In Proceedings of the 2009 ACM workshop on Cloud computing security, CCSW '09, pages 5566, New York, NY, USA, 2009.
- **[Lounis2014]** Ahmed Lounis. Security in cloud computing. Other. Université de Technologie de Compiègne, 2014. English.: 2014COMP1945ff. Fftel-01293631f https://tel.archives-ouvertes.fr/tel-01293631/document

- **[GPSW06]** Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for ne-grained access control of encrypted data. In Proceedings of the 13th ACM conference on Computer and communications security, CCS '06, pages 8998, New York, NY, USA, 2006

- **[BSW07]** John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based encryption. In Proceedings of the IEEE Symposium on Security and Privacy, SP '07, pages 321334, Washington, DC, USA, 2007

- **[PKI]** Thales, What is Public Key Infrastructure (PKI)? Website: https://www.thalesesecurity.com/faq/public-key-infrastructure-pki/what-public-key-infrastructure-pki

- **[Uahhabi2014]** Z. El Uahhabi and H. El Bakkali, "A comparative study of PKI trust models," 2014 International Conference on Next Generation Networks and Services (NGNS), 2014, pp. 255-261, doi: 10.1109/NGNS.2014.6990261.

- **[Ghulam2018]** Ghulam Mustafa, Rehan Ashraf, Muhammad Ayzed Mirza, Abid Jamil, Muhammad: A review of data security and cryptographic techniques in IoT based devices. ICFNDS 2018: 47:1-47:9

- **[owasp2020]** OWASP, OWASP Mobile Top 10: A Comprehensive Guide For Mobile Developers To Counter Risks, 2020,

- **[nist800-111]** NIST 800-111, Guide to Storage Encryption Technologies for End User Devices, Recommendations of the National Institute of Standards and Technology, 2007, https://www.hhs.gov/sites/default/files/nist800111.pdf

- **[Madnani2013]** Madnani, B., & Sreedevi, N. (2013). Attribute Based Encryption for Scalable and Secure Sharing of Medical Records in Cloud Computing Design and Implementation. International Journal of Innovative Research in Computer and Communication Engineering, 1(3).

- **[IAP09]** L. Ibraimi, M. Asim, and M. Petkovic. Secure management of personal health records by applying attribute-based encryption. In 6th International Workshop on Wearable Micro and Nano Technologies for Personalized Health, pHealth'09, pages 7174, Oslo, Norway, June 2009.

- **[LYRL10]** Ming Li, Shucheng Yu, Kui Ren, and Wenjing Lou. Securing personal health records in cloud computing : Patient-Centric and Fine-Grained data access control in multi-owner settings. In Security and Privacy in Communication Networks, volume 50, pages 89106. Springer Berlin Heidelberg, 2010.

- **[ENISA2021]** ENISA. “Pseudonymisation for Personal Data Protection”. 2021.

- **[D3.1]** InteropEHRate Consortium, *D3.1 - Specification of S-EHR mobile privacy and security conformance levels - V1*, 2020. www.interopehrate.eu/resources

- **[D2.6]** InteropEHRate Consortium, *D2.6 - InteropEHRate Architecture - V3*, 2021. www.interopehrate.eu/resources

- **[D3.9]** InteropEHRate Consortium, *D3.9 - Design of libraries for HR security and privacy services - V1*, 2019. www.interopehrate.eu/resources
- **[D3.4]** InteropEHRate Consortium, *D3.4 - Specification of remote and D2D IDM mechanisms for HRs Interoperability - V2*, 2021. www.interopehrate.eu/resources
- **[D3.8]** InteropEHRate Consortium, *D3.8 - Specification of consent management and decentralized authorization mechanisms for HR Exchange - V2*, 2021. www.interopehrate.eu/resources
- **[D4.3]** InteropEHRate Consortium, *D3.4 - Specification of remote and D2D protocol and APIs for HR exchange V3*, 2021. www.interopehrate.eu/resources
- **[D4.9]** InteropEHRate Consortium, *D4.9 - Specification of protocol and APIs for research health data sharing - V2*, 2021. www.interopehrate.eu/resources
- **[D6.8]** InteropEHRate Consortium, *InteropEHRate D6.8-Design of a mobile service for data anonymization and aggregation*, 2021. www.interopehrate.eu/resources
- **[CCC2020]** Confidential Computing Consortium, Confidential Computing Consortium, 2020. Web site: https://confidentialcomputing.io
- **[IEEE2020]** IEEE, The rise of confidential computing: Big tech companies are adopting a new security model to protect data while it's in use. 2020. Web site: https://ieeexplore.ieee.org/abstract/document/9099920
- **[Göttel2019]** C. Göttel et al., "Security, Performance and Energy Trade-Offs of Hardware-Assisted Memory Protection Mechanisms," 2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS), Salvador, Brazil, 2018, pp. 133-142, doi: 10.1109/SRDS.2018.00024.
- **[Pires2019]** Rafael Pereira Pires, "Distributed systems and trusted execution environments: Trade-offs and challenges", Thèse présentée à la Faculté des sciences pour l'obtention du grade de Docteur ès sciences, 2019
- **[Google2020]** Introducing Google Cloud Confidential Computing with Confidential VMs, 2020. Web site: https://cloud.google.com/blog/products/identity-security/introducing-google-cloud-confidential-computing-with-confidential-vms
- **[EJBCA 2021]** PrimeKey, EJBCA WS Support, 2021 https://download.primekey.se/docs/EJBCA-Enterprise/latest/ws/index.html
- **[eIDAS 2017]** European Commission — DIGIT Unit D3, eIDAS-Node Installation, Configuration and Integration Manual, Version 1.3, 2017
- **[1609.2-2016]** "IEEE Standard for Wireless Access in Vehicular Environments--Security Services for Applications and Management Messages," in IEEE Std 1609.2-2016 (Revision of IEEE Std 1609.2-2013), vol., no., pp.1-240, 1 March 2016, doi: 10.1109 / IEEESTD 2016 7426684.
- **[THESSLSTORE2019]** The Difference Between Root Certificates and Intermediate Certificates, 2019 Website:https://www.thesslstore.com/blog/root-certificates-intermediate/

- **[KENNEDY2016]** Kennedy, E. and Millard, C., "Data security and multi-factor authentication: Analysis of requirements under EU law and in selected EU Member States", Computer Law & Security Review, 32/1, 91-110, 2016.
- **[ESENS2017]** Masi , M., Bittins, S. Cunha, J. and Atzeni, A., "e-SENS 5.2 eHealth eIDAS eID Pilot: Technical Feasibility Report", 2017.

5 APPENDIX A

This section summarises all the notions used in the design of the cryptographic libraries.

Symbol	Description
G	Multiplicative group
g	Generator
Z_p^*	Group
r	Random value
q, p	Large primes
σ	Cryptographic signature
Pr	Private key
C	Certificate
Ver	Verify
N	Nonce
Z	Symmetric key
Enc	Encryption
Dec	Decryption
$Auth$	Authentication/Authorization
m	Health data
QR	QR code
$tstamp$	Timestamp
T_{ID}	Transient identifier - anonymous assertion

L_{ID}	Long-term identifier - real id
pid	Pseudo-id
$PGen$	Pseudonym generation based on group signatures
$pseu$	Pseudonym
An	Anonymized the data with anonymous signing
CSR	Certificate signing request
$eidtkn$	eIDAS token

Table 3 - Notation used

JSON-schema for the D2D Security Message

The JSON-schema for the D2D requests is specified below:

```
{
  "$id": "http://example.com/example.json",
  "$schema": "http://json-schema.org/draft-07/schema",
  "description": "The root schema of a D2DSecurityMessage",
  "required": [
    "header",
    "operation",
    "body"
  ],
  "type": "object",
  "properties": {
    "header": {
      "$id": "#/properties/header",
      "type": "object",
      "title": "The header schema",
      "description": "An explanation about the purpose of this instance.",
      "default": {},
      "examples": [
        {
          "timeStamp": "2021-07-26T14:13:13.553Z",
          "agent": "JRE 1.8.0_261 - Windows 10 10.0",
          "protocol": "D2D",
          "version": "1"
        }
      ],
      "required": [
        "timeStamp",
        "agent",
```

```
        "protocol",
        "version"
      ],
      "properties": {
        "timeStamp": {
          "$id": "#/properties/header/properties/timeStamp",
          "examples": [
            "2021-07-26T14:13:13.553Z"
          ],
          "type": "string"
        },
        "agent": {
          "$id": "#/properties/header/properties/agent",
          "description": "The agent that created the message",
          "examples": [
            "JRE 1.8.0_261 - Windows 10 10.0"
          ],
          "type": "string"
        },
        "protocol": {
          "$id": "#/properties/header/properties/protocol",
          "default": "D2D",
          "description": "The name of the used protocol.",
          "enum": [
            "D2D"
          ],
          "type": "string"
        },
        "version": {
          "$id": "#/properties/header/properties/version",
          "default": "1",
          "description": "version of the protocol used",
          "type": "string"
        }
      },
      "additionalProperties": true
    },
    "operation": {
      "$id": "#/properties/operation",
      "description": "The name of the operation under execution of the D2D security protocol",
      "examples": [
        "HELLO_SEHR"
      ],
      "enum": [
        "HELLO_SEHR",
        "HELLO_HCP",
        "SEHR_PUBLIC_KEY",
        "HCP_PUBLIC_KEY",
        "UNSIGNED_CONSENT",
        "SIGNED_CONSENT"
```

```
        ],
        "type": "string"
      },
      "body": {
        "$id": "#/properties/body",
        "description": "The body of the message contains the exchanged data",
        "type": "string"
      }
    },
    "additionalProperties": true
}
```


JSON sample for HCPPublicKey message


```
{
        "header": {
                "timeStamp": "2021-07-26T14:13:13.553Z",
                "agent": "JRE 1.8.0_261 - Windows 10 10.0",
                "protocol": "D2D",
                "version": "1"
        },
        "operation": "HCP_PUBLIC_KEY",
        "body": "XTYRE8768LO8fwrqwm4l523k5203434279824jkhg2GTUYEbjgfg3232ljo9\u003d..."
}
```


JSON sample for citizenPublicKey message


```
{
        "header": {
                "timeStamp": "2021-07-26T14:13:14.553Z",
                "agent": "JRE 1.8.0_261 - Windows 10 10.0",
                "protocol": "D2D",
                "version": "1"
        },
        "operation": "SEHR_PUBLIC_KEY",
        "body": "MIHfMIGXBgkqhkiG9w0BAwEwgYkCQQD8poLOjhLKuibvzPcRDlJtsHiwXt3d..."
}
```