D3.5

Specification of data encryption mechanisms for mobile and web applications - V1

ABSTRACT

This deliverable provides the first version of the specification of protocols for encryption mechanics for both health data storage and health data exchange. This document also provides a detailed technical background, which is a necessary step to move forward.

Delivery Date	28th August 2020
Work Package	WP3
Task	T3.2
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, Dimitris Papamartzivanos	UBIT
Contributors	Stella Dimopoulou, Crysostomos Symvoulidis	BYTE
Reviewers	Paolo Marcheschi	FTGM
Reviewers	Christina Kotsiopoulou	HYGEIA

LOGTABLE

Version	Date	Change	Author	Partner
0.1	17-02-20	First draft of ToC	Sofianna Menesidou, Entso Veliou	UBIT
0.2	20-02-20	Introduction	Sofianna Menesidou	UBIT
0.3	25-02-20	Encryption for Data in Transit, Encryption for Data in Storage	Stella Dimopoulou, Crysostomos Symvoulidis	BYTE
0.4	04-03-20	Technical Background	Sofianna Menesidou	UBIT
0.5	10-03-20	Cloud Data Storage and Break-glass Encryption	Sofianna Menesidou	UBIT
0.6	28-04-20	InteropEHRate Encrypted Storage, InteropEHRate Encrypted Communication	Sofianna Menesidou	UBIT
0.7	12-05-20	R2D Backup and R2D Emergency Encrypted Communication Conceptual API	Sofianna Menesidou	UBIT
0.8	25-05-20	Conclusions, Internal review	Sofianna Menesidou, Dimitris Papamartzivanos	UBIT
0.9	21-06-20	Internal review updates	Sofianna Menesidou	UBIT
1.0	23-06-20	Quality check	Argyro Mavrogiorgou	UPRC
1.1	26-06-20	Review and Quality check	Laura Pucci	ENG
1.2	03-07-20	InteropEHRate Encrypted Storage	Sofianna Menesidou	UBIT
1.3	06-07-20	InteropEHRate Encrypted Storage	Sofianna Menesidou	UBIT
1.4	21-07-20	InteropEHRate Encrypted Storage	Sofianna Menesidou	UBIT
vFinal	28-08-20	Final review and version for submission	Laura Pucci	ENG

ACRONYMS

Acronym	Term and definition
ABE	Attribute-based encryption
AES	Advanced Encryption Standard
BIOS	Basic Input/Output System
BLE	Bluetooth Low Energy
CA	Certificate Authority
CP-ABE	Ciphertext policy Attribute-based encryption
CRUD	create, read, update and delete
DES	Data Encryption Standard
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
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
RBAC	Role Based Access Control
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 [45] [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 a 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 a 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 mean 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 transferred and stored in the cloud. In fact, the outsourcing to cloud brings several security risks.

Due to the high value of the sensitive health data, the 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 [13] [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 [3] [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 among them. 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 from the 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-based specification 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.
- 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.
- Section 4 concludes the deliverable, including the updates and the future development plans.

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

Not applicable. This deliverable contains the first version of the Specification of encryption mechanisms.

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 exchanged and a detailed literature review on the challenging cloud data storage.

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 [44] [Ghulam2018]. Cryptography is used to store and transfer the data in such a form that only the sender and the receiver can understand it 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 [14] [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 [14] [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 Private Key Generator (PKG) [35] [BF03].

Attribute-based encryption: Attribute-based encryption (ABE) is a recent promising cryptographic method proposed by Sahai and Waters in 2005 [34] [SW05]. The ABE technique extends the 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 for avoiding unauthorized data access since the access policy is embedded in the ciphertext itself [41] [Lounis2014]. The two main variants of ABE are the Key-Policy Attribute-Based Encryption (KP-ABE) [42] [GPSW06] and the Ciphertext Policy Attribute-Based Encryption (CP-ABE) [43] [BSW07].

2.2. Encryption for Data in Transit

Security is one of the main challenges when it comes to eHealth services and is crucial requirement for the transmission of required health data over the network. Data in transit are vulnerable to interception and potentially redirection attacks. InteropEHRate deals with three protocols namely the D2D, R2D and RDS. The D2D is over Bluetooth without Internet usage, while the R2D 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 [25] [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 [26] [Loveless2018] [29] [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 that the establishment of the secure channel, they can securely exchange their data [28] [bon2016] [29] [Ravikiran].

Prior to Bluetooth version 2.1, pairing was not secure at all [26] [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 by 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 [27] [Corella2015]. In the context of InteropEHRate, the latest AES encryption of Bluetooth will be used, apart from the application level encryption.

2.2.2. Data exchange over Internet

Traditionally, secure socket layer (SSL) is used for establishing 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 letter "S" is referred to Security. HTTPS is used to establish a secure communication over a computer network [32] [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 [33] [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 [31] [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. The negotiation for the agreement on the utilised algorithms is handled internally by the protocol. The most frequent algorithm for symmetric encryption is Advanced Encryption Standard (AES) and for asymmetric encryption is Diffie-Hellman [30] [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 [46] [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 [46] [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, the 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, the full-disk encryption (FDE) and the 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 [16] [androidd2020]. In 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 [17] [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) [17] [androidf2020].

- **iOS Data Storage** - Apple automates by default the FBE encryption process of an iPhone from version 8 and above [18] [kaspersky] with a 256-bit AES encryption [19] [applesec]. The data which is stored on the phone is automatically encrypted through a unique identifier which is 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 [20] [nield2020] [21] [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 [22] [microsoftder2019]. The encryption process is using either AES or Triple DES algorithm [23] [microsofttde2019]. The process of encryption and decryption are real time and they are completely transparent to the applications that have access to these databases [22] [microsoftder2019]. NoSQL databases, and specifically MongoDB, support data-in-motion encryption and the data-at-rest encryption [24] [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 encrypted. 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 [47] [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 [10] [Scafuro2019]. Several works in the literature deal with the concept of break-glass encryption for cloud storage [10] [Scafuro2019] [12] [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 [8] [Abbas2014]. However, this approach does not protect the medical data against internal attacks [8] [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 [36] [KRS+03]

[37] [GSMB03] [38] [BCHL09] [39] [dVFJ+07] [40] [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 [5] [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 [4] [Zhang2010] [7] [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 [12] [Oliveira2020]. Moreover, several works attempt to address access control of encrypted data by using secret sharing schemes combined with identity-based encryption [1] [Benaloh1988] [2] [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 [9] [Zhang2016]. The work in [10] [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 [12] [Oliveira2020]. The authors in [15] [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 [48] [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 [49] [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 [41] [Lounis2014]. The work in [11] [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 [12] [Oliveira2020] proposes the usage of the ciphertext-policy ABE (CP-ABE) associated with policies defined for emergency situations, based on the research [3] [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 the medical data and CP-ABE for the 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 major IT players provide some form of "infrastructure-as-a-service" (IaaS) commercial offerings, including Microsoft, Google and Amazon [55] [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 [55] [Göttel2019]. However, the 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 [55] [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 [53] [CCC2020]. Different TEE implementations vary in terms of features. The two most know TEE technologies are Intel SGX and AMD SEV.

- **Intel Intel Software Guard Extension (SGX)** [56] [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)** [56] [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 to 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 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 [57] [Google2020]. Table 1 below summarizes these two know TEE technologies used for confidential computing [56] [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 [56] [Pires2019]

3. INTEROPEHRATE SPECIFICATION FOR ENCRYPTION MECHANISMS

The purpose of this Chapter is to show how the InteropEHRate project will handle encryption/decryption mechanisms, for data storage and data exchange in the context of InteropEHRate use cases, namely medical visit, research protocol and emergency scenarios. In this section is provided an overview of how the different actors and organizations involved in the InteropEHRate architecture in D2.4 [52] interact with each other in the context of the encryption mechanisms. 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 the 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).

3.1.InteropEHRate Encrypted Storage and Communication

InteropEHRate architecture involves three communication protocols: the device-to-device (D2D), the remote-to-device (R2D) and the research data sharing (RDS). In the context of R2D and RDS, 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 the TLS 1.2 and Bluetooth encryption are missing or not enabled. This deliverable will focus on the application level encryption specification. As can be seen in Figure 1 below, apart from the storage encryption, encryption is needed in all the communication channels, namely R2D Access, R2D Backup, R2D Emergency, RDS Research and D2D since sensitive medical data are transferred.

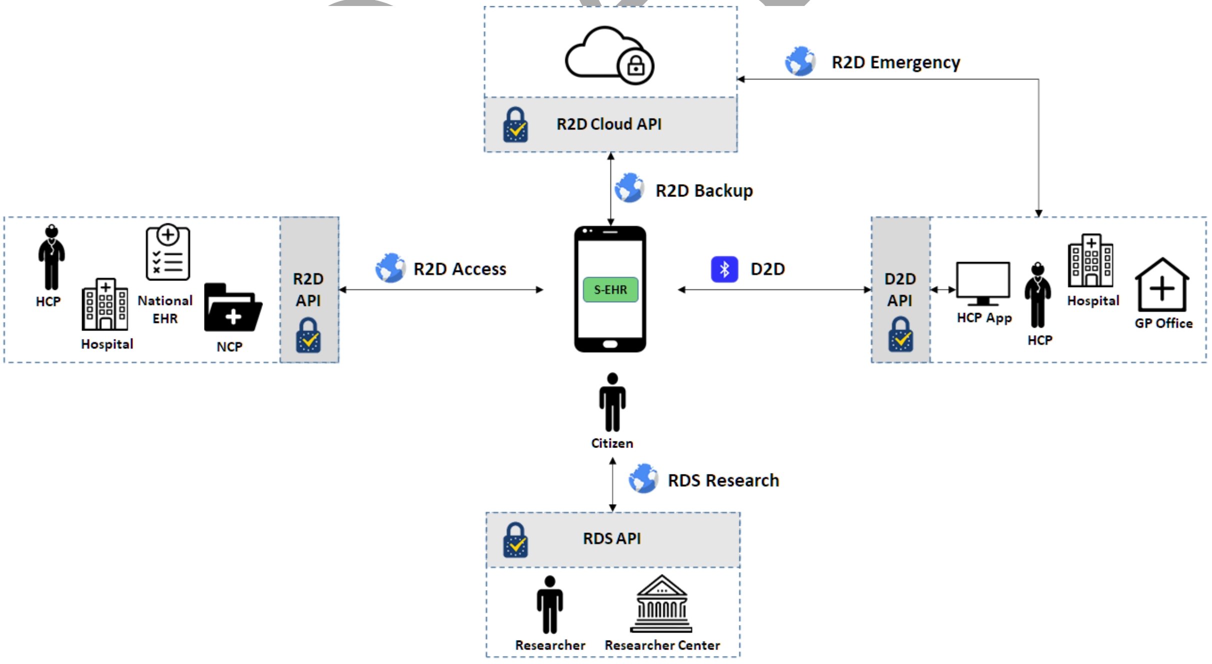

Figure 1 – Usage of the D2D, R2D and RDS protocols

3.2.1. InteropEHRate Security Protocol Phases

InteropEHRate security protocol for D2D, R2D Access and RDS, towards establishing an encrypted storage and communication channel consists of the following five phases:

- **Phase 1**: *Bootstrap* - In this phase the prerequisites regarding certificate acquisition on both entities are performed.
- **Phase 2**: *IDM Phase* - In this phase each entity verifies the identity of the other entity by certificate exchange and signature verification.
- **Phase 3**: *Consent Management Phase* - In this phase the patient gives his consent for process upon his data.
- **Phase 4**: *Key Establishment Phase* - In this phase a symmetric key establishment is performed for secure communication.
- **Phase 5**: *Encrypted Storage and Communication Phase* - In this phase both parties use the established symmetric key to transfer and store data in encrypted form.

InteropEHRate security protocol for R2D Backup and R2D Emergency, towards establishing an encrypted storage and communication channel consists of the following three phases:

- **Phase 1**: *Consent Management Phase* - In this phase the patient gives his consent for process upon his data.
- **Phase 2**: *Encrypted Storage and Communication Phase* - In this phase both parties use the established symmetric key to transfer and store data in encrypted form.
- **Phase 3**: *Emergency Phase* - In this phase the HCP request and retrieves the encrypted data.

This deliverable will focus on Phases 4 - *Key Establishment Phase* and 5 - *Encrypted Storage and Communication Phase* in the context of D2D, R2D Access and RDS and on Phases 2 - *Encrypted Storage and Communication Phase* and 3 - *Emergency Phase* in the context of R2D Backup and R2D Emergency, since previous Phases were already defined in the previous InteropEHRate deliverables D3.3 [50] , D3.7 [51] and are out of scope of this deliverable. However, all previous Phases should be completed successfully before the key establishment and the encrypted communication phases take place.

3.2.2. D2D, R2D Access and R2R Encrypted Communication Conceptual API

The following *Figure 2* displays the UML diagram representing the five phases including the encryption/decryption functionality based on the D2D. Phases 4 and 5 should be the same in all the D2D, R2D and RDS. D3.10 will provide further details regarding the design of the libraries.

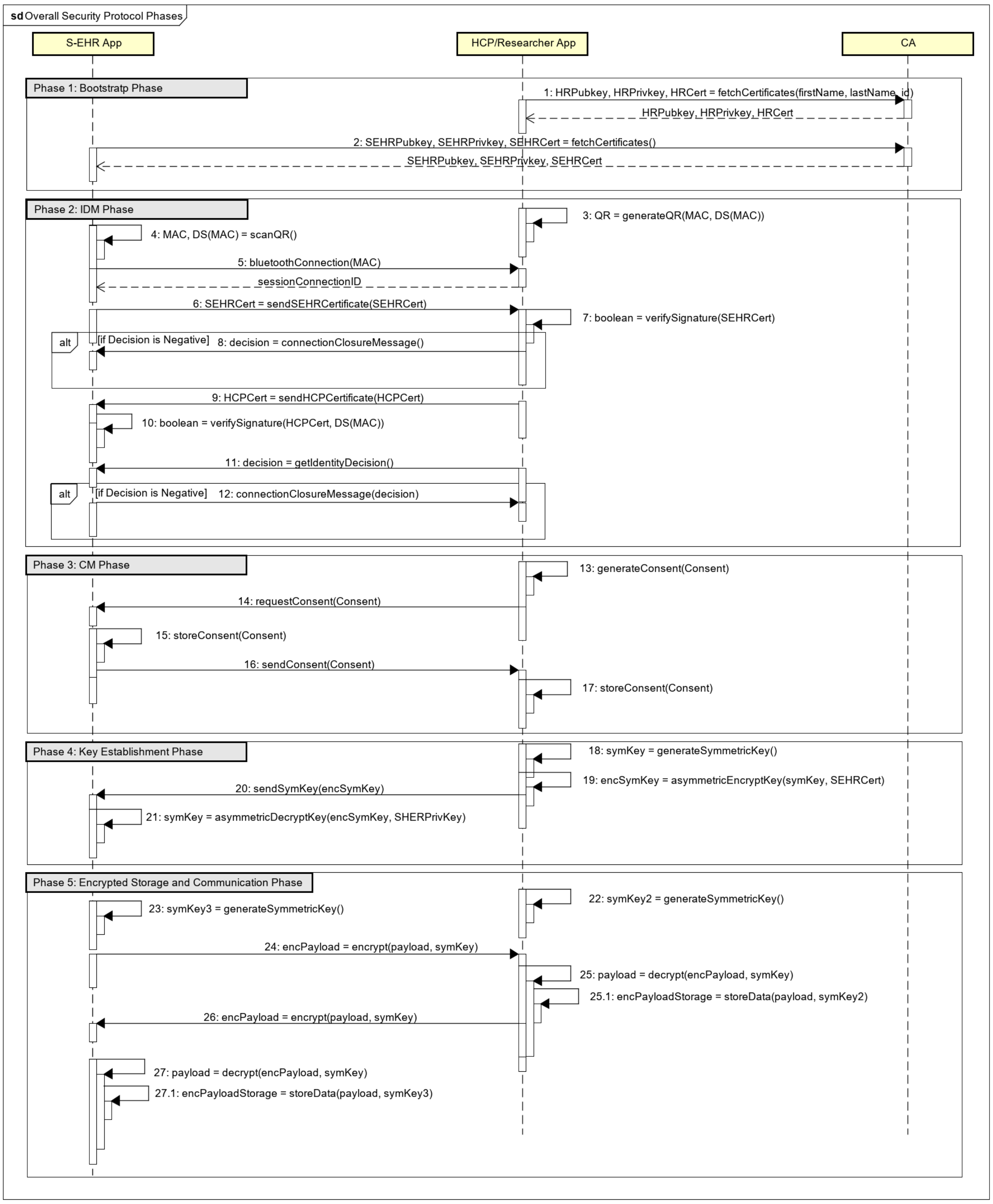
sd Overall Security Protocol Phases
S-EHR App
HCP/Researcher App
CA
Phase 1: Bootstratp Phase
1: HRPubkey, HRPrivkey, HRCert = fetchCertificates(firstName, lastName, id)
HRPubkey, HRPrivkey, HRCert
2: SEHRPubkey, SEHRPrivkey, SEHRCert = fetchCertificates()
SEHRPubkey, SEHRPrivkey, SEHRCert
Phase 2: IDM Phase
3: QR = generateQR(MAC, DS(MAC))
4: MAC, DS(MAC) = scanQR()
5: bluetoothConnection(MAC)
sessionConnectionID
6: SEHRCert = sendSEHRCertificate(SEHRCert)
7: boolean = verifySignature(SEHRCert)
alt
[if Decision is Negative]
8: decision = connectionClosureMessage()
9: HCPCert = sendHCPCertificate(HCPCert)
10: boolean = verifySignature(HCPCert, DS(MAC))
11: decision = getIdentityDecision()
alt
[if Decision is Negative]
12: connectionClosureMessage(decision)
Phase 3: CM Phase
13: generateConsent(Consent)
14: requestConsent(Consent)
15: storeConsent(Consent)
16: sendConsent(Consent)
17: storeConsent(Consent)
Phase 4: Key Establishment Phase
18: symKey = generateSymmetricKey()
19: encSymKey = asymmetricEncryptKey(symKey, SEHRCert)
20: sendSymKey(encSymKey)
21: symKey = asymmetricDecryptKey(encSymKey, SHERPrivKey)
Phase 5: Encrypted Storage and Communication Phase
22: symKey2 = generateSymmetricKey()
23: symKey3 = generateSymmetricKey()
24: encPayload = encrypt(payload, symKey)
25: payload = decrypt(encPayload, symKey)
25.1: encPayloadStorage = storeData(payload, symKey2)
26: encPayload = encrypt(payload, symKey)
27: payload = decrypt(encPayload, symKey)
27.1: encPayloadStorage = storeData(payload, symKey3)

Figure 2 – Overall Security Protocol Phases (D2D, R2D Access and RDS)

The following tables provide a complete description of all the methods, which are similar for the D2D, R2D Access and RDS communication channels (see Figure 1).

Operation generateSymmetricKey / Phase 4 / Key Establishment

Name	generateSymmetricKey
Description	The symmetric key generation method takes no input. It outputs the symmetric session key encryption in transit. More specifically, the Java KeyGenerator class (javax.crypto.KeyGenerator) will be used to generate symmetric encryption keys of 256 bit length keys for secure communication.
Arguments	• No arguments
Return Value	• Symmetric key symKey
Exceptions	• GeneralSecurityException • IOException
Preconditions	• All the Phases from 1 to 3 to finish successfully.

Table 2 - generateSymmetricKey

Operation asymmetricEncryptKey / Phase 4 / Key Establishment

Name	asymmetricEncryptKey
Description	The asymmetric encrypt key takes as input the generated symmetric key symKey and the SEHRCert for secure key exchange. It outputs the encrypted symmetric key encSymKey. Asymmetric encryption algorithm will be used for encryption (e.g. Diffie-Hellman key exchange or RSA-based encryption).
Arguments	• Symmetric key symKey • Certificate SEHRCert
Return Value	• Encrypted symmetric key encSymKey
Exceptions	• GeneralSecurityException • IOException
Preconditions	• All the Phases from 1 to 3 to finish successfully.

Table 3 – asymmetricEncryptKey

Operation sendSymKey / Phase 4 / Key Establishment

Name	sendSymKey
Description	The sendSymKey method takes as input the encrypted symmetric key encSymKey. It outputs the encrypted symmetric key encSymKey, in order both parties have the same key.
Arguments	• Encrypted symmetric key encSymKey
Return Value	• Encrypted symmetric key encSymKey
Exceptions	• GeneralSecurityException • IOException
Preconditions	• All the Phases from 1 to 3 to finish successfully.

Table 4 - sendSymKey

Operation asymmetricDecryptKey / Phase 4 / Key Establishment

Name	asymmetricDecryptKey
Description	The asymmetricDecryptKey takes as input the encrypted symmetric key encSymKey and the Private key SHERPrivKey. It outputs the symmetric key symKey. The same algorithm with encryption will be used for decryption.
Arguments	• Encrypted symmetric key encSymKey • Private key SHERPrivKey
Return Value	• Symmetric key symKey
Exceptions	• GeneralSecurityException • IOException
Preconditions	• All the Phases from 1 to 3 to finish successfully.

Table 5 – asymmetricDecryptKey

Operation encrypt / Phase 5 / Encrypted Communication

Name	encrypt
Description	The encryption method takes as input the payload that needs to transfer securely and the symmetric key symKey. It outputs the encrypted payload encPayload. More specifically, AES 256 block cipher will be used as an encryption/decryption algorithm. This method applies to both the entities for secure communication.
Arguments	• Payload payload • Symmetric key symKey
Return Value	• Encrypted payload encPayload
Exceptions	• GeneralSecurityException • IOException
Preconditions	• All the Phases from 1 to 4 to finish successfully.

Table 6 – encrypt

Operation decrypt / Phase 5 / Encrypted Communication

Name	decrypt
Description	The decryption algorithm takes as input encrypted payload encPayload and the symmetric key symKey. It outputs the payload. More specifically, AES 256 block cipher will be used as an encryption/decryption algorithm. This method applies to both the entities for secure communication.
Arguments	• Encrypted payload encPayload • Symmetric key symKey
Return Value	• Payload payload
Exceptions	• GeneralSecurityException
Preconditions	• All the Phases from 1 to 4 to finish successfully.

Table 7 – decrypt

Operation generateSymmtericKey / Phase 5 / Encrypted Storage

Name	generateSymmetricKey
Description	The symmetric key generation method takes no input. It outputs the symmetric key for encryption in storage. More specifically, the Java KeyGenerator class (javax.crypto.KeyGenerator) will be used to generate symmetric encryption keys of 256 bit length keys for secure storage. This method applies to both the entities for secure storage.
Arguments	• No arguments
Return Value	• Symmetric key symKey
Exceptions	• GeneralSecurityException • IOException
Preconditions	N/A

Table 8 – generateSymmetricKey

Operation storeData / Phase 5 / Encrypted Storage

Name	storeData
Description	The safe storage of the data. It encrypts the data before stored with a different symmetric known only to the application that stores the data. This method applies to both the entities for secure communication.
Arguments	• Payload payload • Symmetric key symKey2/symKey3
Return Value	• Encrypted payload encPayloadStorage
Exceptions	• GeneralSecurityException • IOException
Preconditions	• This symmetric key should be different from the one used for the encrypted communication for security purposes.

Table 9 – storeData

3.2.3. R2D Backup and R2D Emergency Encrypted Communication Conceptual API

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 and uploads them to the cloud for storage. The symmetric key is added in a QR code, in order the HCP could access it for emergency cased. The HCP will decrypt the ciphertext if after scanning the QR code and retrieving the symmetric key. The following *Figure 3* displays the UML diagram representing the three phases including the encryption/decryption functionality. This deliverable will focus on Phases 2 and 3 since, Phase 1 is out of scope of this deliverable.

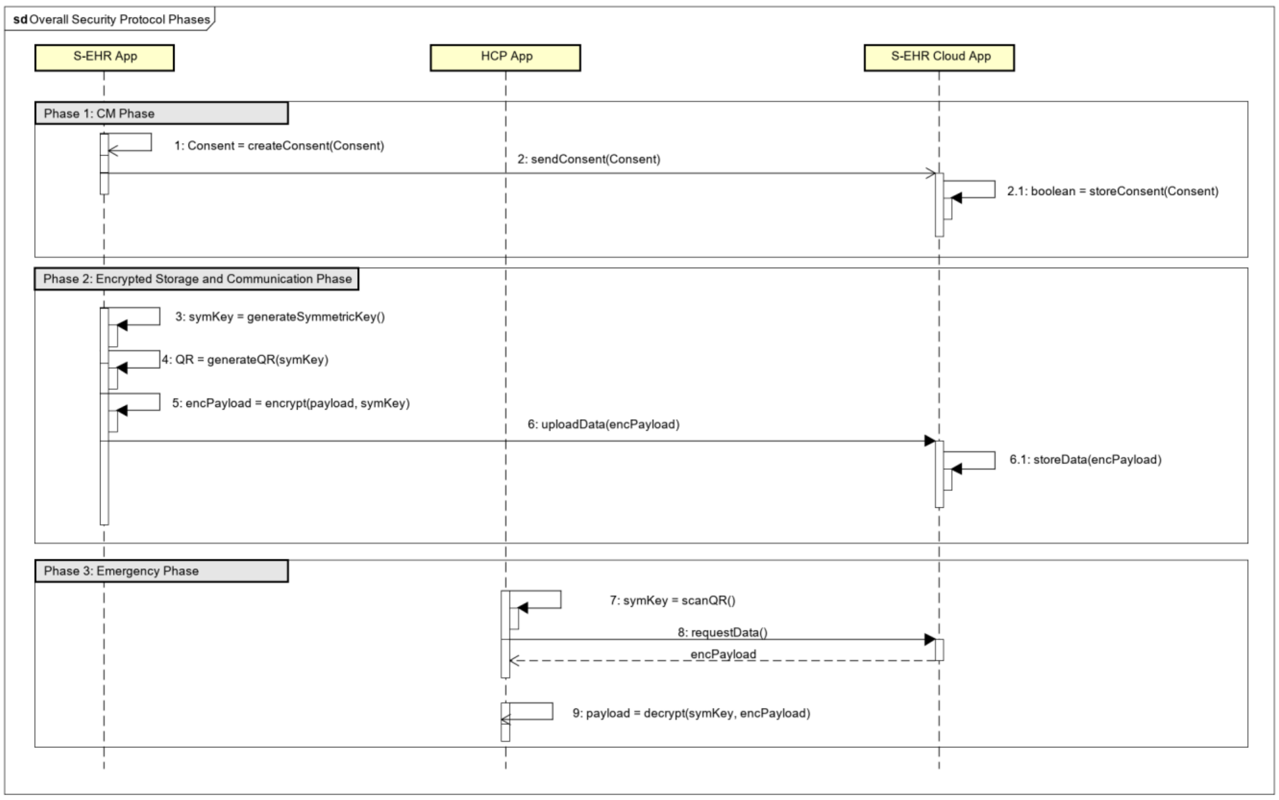
sd Overall Security Protocol Phases
S-EHR App
HCP App
S-EHR Cloud App
Phase 1: CM Phase
1: Consent = createConsent(Consent)
2: sendConsent(Consent)
2.1: boolean = storeConsent(Consent)
Phase 2: Encrypted Storage and Communication Phase
3: symKey = generateSymmetricKey()
4: QR = generateQR(symKey)
5: encPayload = encrypt(payload, symKey)
6: uploadData(encPayload)
6.1: storeData(encPayload)
Phase 3: Emergency Phase
7: symKey = scanQR()
8: requestData()
encPayload
9: payload = decrypt(symKey, encPayload)

Figure 3 – Overall Security Protocol Phases (R2D Backup and R2D Emergency)

Operation generateSymmtericKey / Phase 2 / Encrypted Communication and Storage

Name	generateSymmtericKey
Description	The generateSymmtericKey generates a symmetric key for AES encryption. This will be used to encrypt user's data.
Arguments	• No arguments
Return Value	• Symmetric key symKey

Exceptions	• GeneralSecurityException • IOException
Preconditions	N/A

Table 10 – generateSymmtericKey

Operation generateQR / Phase 2 / Key Establishment

Name	generateQR
Description	The generateQR generates the QR code including the symmetric key that is necessary for decryption in emergency situations.
Arguments	• Symmetric key symKey
Return Value	• QR code
Exceptions	• GeneralSecurityException • IOException
Preconditions	N/A

Table 11 – generateQR

Operation encrypt / Phase 2 / Encrypted Communication and Storage

Name	encrypt
Description	The encryption algorithm takes as input the payload that needs to transfer securely and the symmetric key symKey. It outputs the encrypted payload encPayload. More specifically, AES 256 block cipher will be used as an encryption/decryption algorithm.
Arguments	• Payload payload • Symmetric key symKey
Return Value	• Encrypted payload encPayload

Exceptions	• GeneralSecurityException • IOException
Preconditions	N/A

Table 12 – encrypt

Operation uploadData / Phase 2 / Encrypted Communication and Storage

Name	uploadData
Description	The uploadData method uploads the encrypted data to the cloud. It outputs the encrypted payload encPayload.
Arguments	• Encrypted payload encPayload
Return Value	• Encrypted payload encPayload
Exceptions	• GeneralSecurityException • IOException
Preconditions	• Internet Connection

Table 13 – uploadData

Operation storeData / Phase 2 / Encrypted Storage

Name	storeData
Description	The safe storage of the encrypted data to the cloud. The cloud provider stores the encrypted uploaded data, without the ability to decrypt the data.
Arguments	• Encrypted payload encPayload
Return Value	• No return value
Exceptions	• GeneralSecurityException • IOException

Preconditions	• User is registered in the cloud provider

Table 14 – storeData

Operation scanQR / Phase 3 / Key Establishment

Name	scanQR
Description	The functionality to retrieve the symmetric key for decryption in emergency situations.
Arguments	• No arguments
Return Value	• Symmetric key symKey
Exceptions	• GeneralSecurityException • IOException
Preconditions	N/A

Table 15 – scanQR

Operation requestData / Phase 3 / Encrypted Communication

Name	requestData
Description	The HCP request to retrieve citizens data. HCP provides the necessary credentials (e.g. attributes) for authentication to the cloud and to retrieve the encrypted stored payload.
Arguments	• String [] attributes (e.g. username, password, role, hospital, etc.)
Return Value	• Encrypted payload encPayload
Exceptions	• GeneralSecurityException • IOException

Preconditions	• Internet Connection • Successful Authorization of the HCP

Table 16 - requestData

Operation decrypt / Phase 3 / Encrypted Communication

Name	decrypt
Description	The decryption algorithm takes as input encrypted payload encPayload and the symmetric key symKey. It outputs the payload. More specifically, AES 256 block cipher will be used as an encryption/decryption algorithm.
Arguments	• Encrypted payload encPayload • Symmetric key symKey
Return Value	• Payload payload
Exceptions	• GeneralSecurityException • IOException
Preconditions	• Internet Connection • Successful retrieval of the encrypted data from the cloud

Table 17 – decrypt

4. CONCLUSIONS AND NEXT STEPS

In this report, it's defined the first version of the specification of data encryption mechanisms for mobile and web applications. A technical background with state-of-the-art encryption mechanism and crypto primitives are also provided. This document presents a first draft of the encryption mechanisms for mobile and web applications, reflecting the current understanding by the project consortium. More specifically, this deliverable includes the encryption aspects of all the involved architecture components, protocols, and scenarios for data at rest and in-transit.

A second updated version (final version) of this report is planned for March 2021. The following version will include more details about the encryption mechanism, possible inclusion of ABE encryption in the emergency scenario and further research on the confidential computing for privacy preservation. Last but not least, the second version will include any possible updates regarding the current specification based on the new knowledge acquired from the second year of the project.

REFERENCES

[1] [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

[2] [Brickell1989] E. F. Brickell. Some ideal secret sharing schemes. Journal of Combinatorial Mathematics and Combinatorial Computing, 6:105–113, 1989

[3] [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.

[4] [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

[5] [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

[6] [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

[7] [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

[8] [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

[9] [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

[10] [Scafuro2019] Scafuro A, Break-glass encryption. In: IACR International Workshop on Public Key Cryptography. Springer, pp 34–62, 2019.

[11] [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.

[12] [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).

[13] [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.

[14] [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.

[15] [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

[16] [androidd2020] Android Open Source Project, "Full-disk Encryption", 2020. Web site: https://source.android.com/security/encryption/full-disk

[17] [androidf2020] Android Open Source Project, "File-based Encryption", 2020 Web site: https://source.android.com/security/encryption/file-based

[18] [kaspersky] Kaspersky, "iPhone Encryption: How to Encrypt Your iPhone". Web site: https://usa.kaspersky.com/resource-centerz/preemptive-safety/iphone-encryption

[19] [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

[20] [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/

[21] [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

[22] [microsoftder2019] Microsoft, "Data Encryption at Rest", 2019. Web site: https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/security/transparent-data-encryption

[23] [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

[24] [Townsend] Townsend Security, "The Definitive Guide to MongoDB Encryption & Key Management". Web site: https://info.townsendsecurity.com/mongodb-encryption-key-management-definitive-guide

[25] [Lecroy] Teledyne Lecroy, "How Encryption Works in Bluetooth". Web site: http://www.fte.com/webhelp/bpa500/Content/Documentation/WhitePapers/BPA600/Encryption/HowEncryptionWorks.htm

[26] [Loveless2018] Mark Loveless, "Understanding Bluetooth Security", 2018. Web site: https://duo.com/decipher/understanding-bluetooth-security

[27] [Corella2015] Francisco Corella, “Has Bluetooth Become Secure?”, 2015. Web site: https://pomcor.com/2015/06/03/has-bluetooth-become-secure

[28] [bon2016] Matthew Bon, “A Basic Introduction to BLE Security”, 2016. https://www.digikey.com/eewiki/display/Wireless/A+Basic+Introduction+to+BLE+Security

[29] [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

[30] [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/

[31] [Lake2019] Josh Lake, “What is TLS and how does it work?”, 2019. Web site: https://www.comparitech.com/blog/information-security/tls-encryption/

[32] [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/

[33] [Kothari2019] Kewal Kothari, “How does SSL/TLS make HTTPS secure?”, 2019. Web site: https://hackernoon.com/how-does-ssl-tls-make-https-secure-d247bd4e4cae

[34] [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.

[35] [BF03] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. SIAM J. Comput., 32(3) :586615, March 2003.

[36] [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.

[37] [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.

[38] [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

[39] [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.

[40] [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.

[41] [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

[42] [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

[43] [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

[44] [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

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

[46] [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

[47] [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).

[48] [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.

[49] [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.

[50] [D3.3] Specification of remote and D2D IDM mechanisms for HRs Interoperability - V1, 2019. https://www.interopehrate.eu/resources/#dels

[51] [D3.7] Specification of consent management and decentralized authorization mechanisms for HR Exchange - V1, 2019. https://www.interopehrate.eu/resources/#dels

[52] [D2.4] InteropEHRate Architecture - V1, 2019. https://www.interopehrate.eu/resources/#dels

[53] [CCC2020] Confidential Computing Consortium, Confidential Computing Consortium, 2020. Web site: https://confidentialcomputing.io

[54] [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

[55] [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.

[56] [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

[57] [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