Privacy-preserving Distributed Access Control for Medical Data

Christian Maulany[1], Majid Nateghizad[1], Bart Mennink[2] and Zekeriya Erkin[1]
[1]*Cyber Security Group, Department of Intelligent Systems, Delft University of Technology, The Netherlands*
[2]*Digital Security Group, Radboud University, Nijmegen, The Netherlands*

Keywords: Medical Data, Privacy, Secret Sharing, Polymorphic Encryption and Pseudonomysation.

Abstract: The availability of wearable devices such as smartwatches and fitness trackers are a recent development. Among other things, these devices can measure the activity and vital signs of their wearers. As the types of data these devices are able to gather increases the potential for them to be used as a source of data grows. This calls for a secure method of controlling the digital exchange of medical data between wearables and healthcare providers, and healthcare providers in general. By enforcing the exchange of data to go through a central authority, a patient can be given more control over who is able to access his medical data. This central authority is then given the task of monitoring access and ensuring that all access requirements are met. Though effective, this solution relies on a highly trusted central authority. In this work, we propose a scheme using Polymorphic Encryption and Pseudonomysation and Secret Sharing to provide anonymous data storage and data exchange. Our proposal removes the need for a central authority, and instead uses a group of authorities, of which a quorum is needed to facilitate the exchange of data.

1 INTRODUCTION

In recent years there has been a rise in the number of wearable devices, such as fitness trackers and smartwatches (Wearables, 2017). These devices which often work in conjunction with a smartphone are becoming more accurate in their measurements and will increasingly use cloud storage and become part of the internet of things (Page, 2015). The increased adaptation of these wearables goes hand in hand with Electronic Health Care (e-healthcare): the shift of health care to the digital domain. In an e-healthcare setting, patients are able to measure their medical condition at home, using sensors placed in wearable devices. The result is that medical measurements can be taken over a longer period in an automated fashion. In this manner costs on the number of hospital visits and work performed by a doctor can be reduced. Furthermore, the additional information gathered by these devices can improve the quality of diagnosis.

Collecting and sharing medical data in e-healthcare would benefit greatly from a cloud-based solution, improving the availability of a patient's medical data and the ease of sharing. However, handling medical data in such a manner causes privacy concern. Medical data is highly sensitive; therefore, a cloud Storage Provider (SP) should not be trusted with it. A natural solution would be to store this data in encrypted form, however, providing proper access control and anonymity become two important challenges. What is needed is a system where self-monitoring devices and healthcare providers (HCPs) are able to share medical data, while providing proper access management and security.

Several countries allow a patient to decide who is allowed to (digitally) access their medical data. This is usually done through a Central Authority (CA). The patient tells the CA which HCPs are allowed to exchange his/her medical data, fully trusting the CA to not misuse this information, and furthermore monitor and assist in this exchange accordingly. Usually only certified HCPs are allowed to connect to this system. This means third parties, such as wearable vendors acting as SPs are unable to connect. The result is that if provided at all, every vendor provides a different method of sharing collected data with HCPs, making it complicated to use data collected by wearables in the medical domain.

1.1 Polymorphic Encryption and Pseudonymisation

Polymorphic Encryption and Pseudonymisation for Personalised Healthcare (PEP) by Verheul et al. (Ver-

Maulany, C., Nateghizad, M., Mennink, B. and Erkin, Z.
Privacy-preserving Distributed Access Control for Medical Data.
DOI: 10.5220/0006841403220331
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, pages 322-331
ISBN: 978-989-758-319-3

heul et al., 2016) is a protocol that facilitates such service. PEP allows patients and HCPs to store data at SPs in ElGamal (El Gamal, 1984) encrypted form, without the SP learning anything about the contents and owners of the data. Anonymity is achieved through pseudonymisation, the act of using different identifiers for the same data at different parties. All parties own a secret key that is derived from a master secret key x, known *only by* a Key Server (KS). The parties' secret keys are related in such a way that re-keying of encryptions can be done in an efficient way, without knowing x. PEP has already been used for medical purposes (PEP, 2017) and it seamlessly fits the envisioned Dutch eID scheme (Verheul et al., 2016).

Pivotal to PEP is the role of a Transcryptor (TR), a trusted central authority through which all communication goes and who is able to re-key encryptions. For example, if a user wants to get access to its storage at an SP, it sends to TR an encryption of the pseudonym under which it is known at TR. Consecutively, TR "re-shuffles" this encryption to obtain an encryption of the pseudonym under which the user is known at SP, "re-keys" the encryption to match the key pair of SP, and sends this re-shuffled encryption to SP. The SP can then decrypt, obtain the pseudonym of the user, and grant access. TR learns nothing about the content of encryption, but only who is sharing data with whom.

Although PEP allows efficient communication and transition of data, the KS and TR own secret data that is crucial to the system. Indeed, KS's knowledge of the master secret key and TR's ability to perform re-key encryptions allows collusion of the two to derive all individual secret keys. Even stronger, collusion of TR with a user yields the master secret key. Beyond the issue of collusion, TR forms a real liability to the system, for various reasons:

- it learns all communication, can compute a mapping between pseudonyms and can perform access monitoring;
- scalability becomes a concern with more and more devices automatically storing medical data;
- if TR goes offline, the entire system stops working.

The liability of TR is acknowledged by Verheul et al. (Verheul et al., 2016) and has also been pointed out by Camenisch and Lehmann (Camenisch and Lehmann, 2017).

1.2 Distributed Transcryptor

In this work, we isolate TR from the PEP protocol and propose a solution to resolve liability issues. The solution consists of distributing the role of TR over a group of n TRs, of which a quorum of t TRs has to collaborate in order to provide the mapping of pseudonyms. Any group of less than t TRs is unable to learn decryption keys, perform re-shuffling of pseudonyms, or whatsoever, even with the help of additional HCPs or SPs.

Our solution relies on the threshold secret sharing scheme (Shamir, 1979) used to share all secret data. The re-randomisation, re-keying, and re-shuffling of ElGamal encryptions in the PEP protocol are performed using a neat junction of multiparty protocols based on secret sharing (Ben-Or et al., 1988). A difficulty arises in that re-keying turns out to need exponentiation of the inverse of the shared value, adding up to the complexity of properly composing the individual protocols. Also, we introduce the re-key-shuffling operation often used in PEP and demonstrate that its multiparty computation protocol compares favorably with the re-keying and re-shuffling protocols individually. The protocol is described in detail in Section 4.

From a security perspective, the individual ingredients of our solution are all unconditionally secure in the universal composability framework (Canetti, 2001) except with negligible probability. This readily allows it to be used as black-box in PEP without sacrificing security.

The improved security against collusion is therefore immediate. Beyond collusion, the mapping between pseudonyms cannot be performed by a single TR, as at least t of them are needed to perform such a mapping. The novel setup additionally increases the scalability and availability of such a system: as long as t TRs are online, the system will remain fully functional. A formal security treatment is given in Section 5.

We perform a complexity and performance analysis of our solution in Section 6. The number of multiplications and exponentiations in the multiparty phase (the most expensive steps) are $O(t)$, where t is the number of parties needed to evaluate successfully. In case of $t = 1$, i.e., the classical TR, re-key-shuffling takes 671ms, whereas for $t = 10$ it takes around 733ms and for $t = 40$ it takes around 931ms.[1]

2 PRELIMINARIES

We will introduce the ElGamal cryptosystem in Section 2.1, secret sharing in Section 2.2, and various

[1]Optimization of our implementation may reduce the running time, but it is expected this reduction is more evident for larger values of t.

protocols for secret sharing in Section 2.3.

2.1 ElGamal Cryptosystem

ElGamal is an asymmetric cryptosystem relying on the difficulty of computing discrete logarithms over finite fields (El Gamal, 1984). Let p and q be large primes such that $p = 2q+1$. Let $g \in \mathbb{F}_p^*$ be a generator of multiplicative subgroup $\mathbb{G}_q$ of order q. The secret key is a random value $x < q$, from which the corresponding public key $y = g^x \bmod p$ is derived. Encryption under public key y is performed as $E_y(M) = \langle b,c\rangle = \langle g^r, y^r \cdot M\rangle \bmod p$, where $r < q$ is a random value. Decryption using secret key x is performed as $D_x(\langle b,c\rangle) = \frac{c}{b^x} = \frac{y^r \cdot M}{g^{rx}} = M \bmod p$. The ElGamal cryptosystem is semantically secure if the Decisional Diffie-Hellman (DDH) problem, determine whether $a \cdot b = c$ for given $g, g^a, g^b, g^c \in \mathbb{G}_q$, is hard on $\mathbb{G}_q$.

2.2 Secret Sharing

Let q be any positive integer. A (t,n) threshold scheme allows a secret $S < q$ to be shared among n parties, in such a way that t colluding parties can recover S but any set of $t-1$ parties learns no information about S. We denote by $[S]_q$ the event that $S < q$ is (t,n) threshold shared.

A (t,n) threshold scheme can be constructed using secret sharing (Shamir, 1979). Assume that q is prime, and let f be a polynomial of degree $t-1$ such that $f(0) = S$. Value S is shared among n parties by giving each party i a unique point (u_i, v_i) on f, where u_i is public while v_i is kept secret. Given t unique points on f, any point on the polynomial f, including $S = f(0)$, can be computed using Lagrange interpolation. Let π be a sequence of t unique numbers in $\{1,\ldots,n\}$. In order to obtain point x we calculate

$$f(x) = \sum_{i \in \pi} \ell_i(x) \bmod p\,, \tag{1}$$

where

$$\ell_i(x) = v_i \prod_{j \in \pi, j \neq i} \frac{x - u_j}{u_i - u_j} \bmod p\,. \tag{2}$$

2.3 Secret Sharing Protocols

Multiparty computation based on the secret sharing was first explored by Ben-Or et al. (Ben-Or et al., 1988). Addition of secrets can be done by the n parties trivially at no communication cost, but various more advanced protocols have appeared over time (see, e.g., (Bar-Ilan and Beaver, 1989; Beaver, 2000; Cramer et al., 2000; Algesheimer et al., 2002; Damgård et al., 2006; Toft, 2009)). In this work, we will require protocols for the addition of secrets, the multiplication of secrets, and the generation of random (invertible) secrets, all of which we derive from Bar-Ilan and Beaver (Bar-Ilan and Beaver, 1989), Damgård et al. (Damgård et al., 2006), and Toft (Toft, 2007). In addition, we will require a protocol for exponentiation with a secret, which differs from the former protocols in the setting, interface, and ultimately the efficiency. We will elaborate on the security of below protocols in Section 5.

2.3.1 Addition of Secrets

Consider two sharings $[S]_q$ and $[T]_q$. The parties can generate a sharing $[S+T]_q$ non-interactively by adding their individual shares of S and T. In more detail, if S and T are shared using polynomials f and g of degree $t-1$, every party i holds shares $f(u_i)$ and $g(u_i)$ and can compute the share $h(u_i) = f(u_i) + g(u_i) \bmod p$ corresponding to $[S+T]_q$. Polynomial h is likewise of degree $t-1$. We denote the protocol by

$$\text{add}([S]_q, [T]_q) \coloneqq [S+T]_q\,.$$

Note that the protocol immediately yields a protocol for multiplication of a share with known value, as $[S + \cdots + S]_q = [S]_q + \cdots + [S]_q$.

2.3.2 Multiplication of Secrets

The parties can generate a secret sharing over polynomial $h = f \cdot g$ with threshold $t_h = t_f + t_g - 1$ as follows. Let π be a group of unique parties, with $|\pi| = t_f + t_g - 1$. Every party i computes $h'(x_i) = f(x_i) \cdot g(x_i)$. The degree of h' equals $t_f + t_g - 2$. In order to obtain a secret sharing with threshold t_h, every party i creates a secret sharing for $H'_i = h'(x_i) \prod_{j \in \pi, j \neq i} \frac{x_j}{x_i} x_j$, giving every other party j share $H'_{i,j}$. After exchanging shares every i computes their secret share of h as $\sum_{j \in \pi} H'_{j,i}$. We denote the protocol by

$$\text{mult}([S]_q, [T]_q) \coloneqq [ST]_q\,.$$

2.3.3 Generation of Random Secret

In order to generate a sharing $[R]_q$ of a secret random value R, every party i picks a random polynomial r_i of degree $t-1$ and gives every party j a share $r_{i,j}$. Every party j in turn computes $\sum_{i=1}^{n} r_{i,j}$ as their secret share of R. We denote the protocol by $\text{rand}() \coloneqq [R]_q$.

2.3.4 Generation of Invertible Random Secret

The parties can generate a sharing of (R, R^{-1}), where R is a random invertible value, as follows: first, evaluate $[R]_q = \text{rand}()$ and $[S]_q = \text{rand}()$, and compute

$[RS]_q = \text{rand}([R]_q, [S]_q)$ and reveal $T := RS$. If T is invertible, compute $[TS]_q = [(RS)^{-1}S]_q = [R^{-1}]_q$ non-interactively. The protocol fails if T is not invertible, which happens with probability at most $2/q$. We denote the protocol by $\text{rand}^*() := ([R]_q, [R^{-1}]_q)$.

2.3.5 Exponentiation with Secret

Consider a value M and a sharing $[S]_q$. The parties can compute M^S as follows. Given π, a sequence of t unique numbers in $\{1,\ldots,n\}$, every member i in π calculates $S_i = \ell_i(0)$ using Eq. (2), and computes partial result $R_i = M^{S_i}$. These partial results are combined using

$$\prod_{i\in\pi} R_i = \prod_{i\in\pi} M^{S_i} = M^{\sum_{i\in\pi} S_i} = M^S . \tag{3}$$

We denote the protocol by

$$\exp(M, [S]_q) := M^S .$$

Damgård et al. (Damgård et al., 2006) gave a protocol for obtaining $[T^S]_q$ from $[T]_q$ and $[S]_q$. In our setting, it suffices to output a revealed value rather than a shared value, and this allows us to simplify the protocol significantly. The exponentiation protocol as outlined above appeared before, among others, in (Desmedt and Frankel, 1989, Sect. 3.1) and (Jakobsson, 1999, Sect. 4).

3 POLYMORPHIC ENCRYPTION AND PSEUDONYMISATION

The Polymorphic Encryption and Pseudonymisation (PEP) protocol Verheul et al. (Verheul et al., 2016) allows multiple parties to provide and/or use external storage without parties learning each others' identities. All parties are known by certain pseudonyms, and a trusted transcryptor (TR) takes the task of transforming encryptions and relaying them from one party to another one. In the context of e-health, we are mainly concerned with patients P and HCPs A.

At the core of PEP is a Key Server (KS), that generates a master secret key x and publishes the corresponding public key $y = g^x$. Every party A in the protocol holds a related key pair (x_A, y_A), where $x_A = x \cdot K_A$ and $y_A = y^{K_A}$. The value K_A is the key-factor for A and is only known by TR. Patient P is given a unique identifier $\text{pid}_P \in \mathbb{G}_q$. At every party A providing or accessing data about P, P will be known by a pseudonym of pid_P, namely the value

$$\text{pid}_P@A = \text{pid}_P^{S_A} ,$$

Table 1: Symbols and their meaning.

Symbol	Description
x, y	master key pair
x_A, y_A	key pair of party A
K_A	key-factor for party A
S_A	pseudonym-factor for party A
pid_P	personal identifier of patient P
ppid_P	polymorphic pseudonym of P
$\text{pid}_P@A$	pseudonym of patient P at party A
$\text{epid}_P@A$	encrypted $\text{pid}_P@A$

where S_A is the S_A is the pseudonym-factor of A, a fixed random value known only by the TR. We refer to Table 1 for an overview of the notation.

However, PEP does not operate on plain identities pid_P, but rather on encryptions. Let $\text{ppid}_P = E_y(\text{pid}_P) = \langle b, c\rangle$, an encryption of the identifier of patient P under public key y. Taking advantage of the homomorphic properties of ElGamal, the TR is able to transformations between encrypted pseudonyms.

3.1 PEP Operations

PEP builds upon the following operations:

- *re-randomisation* takes an encryption $\langle b, c\rangle$ and a random value $r < q$, and produces a randomized encryption of the same message,

$$RR_r(\langle b, c\rangle) := \langle g^r \cdot b, y^r \cdot c\rangle \bmod p . \tag{4}$$

- *re-keying* takes an encryption $\langle b, c\rangle$ encrypted under public key y and a value $K < q$ and produces an encryption under public key y^K (hence under private key $x' = x \cdot K$),

$$RK_K(\langle b, c\rangle) := \langle b^{K^{-1}}, c\rangle \bmod p , \tag{5}$$

  where K^{-1} is the inverse of K mod p.

- *re-shuffling* takes an encryption $\langle b, c\rangle$ of message M and a value $S < q$ and produces an encryption of M^S,

$$RS_S(\langle b, c\rangle) := \langle b^S, c^S\rangle \bmod p . \tag{6}$$

3.2 Storing Data

An encrypted identifier ppid_P is stored at all participants supplying data for P. When a data supplier DS wants to store data D at storage facility SF, it first re-randomizes the ppid_P to form a polymorphic pseudonym, $PP = RR_r(\text{ppid}_P)$ using random r. Next, it encrypts the data D to obtain $C = E_y(D)$. The values PP and C are sent to the TR who re-keys and re-shuffles PP using K_{SF} and S_{SF}, respectively, which results in the encrypted polymorphic pseudonym:

$$\text{epid}_P@SF = RS_{S_{SF}}(RK_{K_{SF}}(PP)) . \tag{7}$$

TR sends $\text{epid}_P@SF$ and C to storage facility SF. SF then decrypts $\text{epid}_P@SF$ using his private key: $D_{y_{SF}}(\text{epid}_P@SF) = \text{pid}_P@SF = \text{pid}_P^{S_{SF}}$. SF stores C using $\text{pid}_P@SF$ as a reference.

3.3 Retrieving Data

Let A be a participant authorized to retrieve data D stored encrypted as $C = E_y(D)$ at storage facility SF. Since A is eligible to retrieve D he knows $\text{ppid}_P = E_y(\text{pid}_P)$. A performs $PP = RR_r(\text{ppid}_P)$ using random r and sends PP to the TR. TR computes $\text{epid}_P@SF$ as in Eq. (7), and sends the result to SF. SF decrypts $\text{epid}_P@SF$ to obtain $\text{pid}_P@SF$ as before, and uses that to look up the requested encrypted data C. SF re-randomizes C, $C' = RR_s(C)$ using random s, and sends C' to TR. TR re-keys and re-shuffles C' using the key-factor and pseudonym-factor of A, $R = RK_{K_A}(C')$, and forwards it to A. A can subsequently decrypt and obtain $D' = D_{x_A}(C') = D$.

3.4 Limitations of PEP

Through the pseudonymisation process, participants have to communicate through the TR. This creates a central party, who can perform access management, logging and monitoring. However, this centralization and other design decisions create several security threats.

- The key server holds the master private key x. If it collaborates with the storage facility, all data and all polymorphic pseudonyms can be decrypted.
- TR holds secrets K_A and S_A for all parties A, whereas each participant A holds $x_A = x \cdot K_A$. If TR collaborates with any party A, they can learn x, creating the same security threat as compromising the key server. Furthermore, they can compute any other private key, as $x_B = x \cdot K_B$. Furthermore, they are able to learn personal identifier $\text{pid}_{A'}$ from any pseudonym of A'.

To prevent malicious behavior of the key server and TR, PEP proposes the use of Hardware Security Modules (HSMs). These HSMs manage cryptographic keys, key-factors and pseudonym-factors and all PEP operations can only be performed inside these HSMs. Though this prevents direct leakage of these secret values, TR is still able to transform any data or pseudonym, such that it can be decrypted by any participant.

Besides security threats, the construction of PEP can only be scaled vertically. Keeping the context of wearables in mind, we can only expect the amount of data which the TR needs to process to increase. Furthermore, whenever TR goes offline, no exchange of data can take place.

4 PRIVACY-PRESERVING DISTRIBUTED ACCESS CONTROL

In the original PEP a single TR is used to evaluate the pseudonym and re-keying mappings of Section 3.1. To obtain our solution where these operations are instead performed by a quorum of t out of n TRs, we first show how these operations can be performed in a (t,n) manner (Section 4.1). Next, we demonstrate how new parties and TRs can be added to the scheme after setup (Section 4.2 and 4.3). Finally, we suggest a way through which t' TRs can jointly perform key generation and key management, which would remove the need for a trusted key server (Section 4.4).

4.1 Distributed PEP Operations

Let $\text{ppid}_P = E_y(\text{pid}_P) = \langle b,c \rangle$, an encryption of the identifier of patient P. We describe how to compute the PEP operations described in Section 3.1 in a distributed fashion. As generating $[S_P]_q$ and $([K_P]_q, [K_P^{-1}]_q)$ are necessary for performing distributed PEP operations, TRs can pre-compute the tuple of $< [S_P]_q, [K_P]_q, [K_P^{-1}]_q >$ for each patient P. Moreover, TRs can pre-compute $[Q_P]_q := [K_P^{-1} \cdot S_P]_q = \text{mult}([K_P^{-1}]_q, [S_P]_q)$, which is used later in operations.

- *Distributed re-keying* of $\langle b,c \rangle$ for a shared value $[K]_q$ with $K < q$ is performed using the exponentiation protocol of Section 2.3:

$$RK_K(\langle b,c \rangle) = \langle \exp(b, [K^{-1}]_q), c \rangle \bmod p. \quad (8)$$

- *Distributed re-shuffling* of $\langle b,c \rangle$ of message M for a shared value $[S]_q$ with $S < q$ is performed using the exponentiation protocol of Section 2.3:

$$RS_S(\langle b,c \rangle) = \langle \exp(b, [S]_q), \exp(b, [S]_q) \rangle \bmod p. \quad (9)$$

PEP often performs re-keying and re-shuffling on the same encryption consecutively, i.e., it often evaluates $RS_S \circ RK_K$, where S and K correspond to the same party. Instead of applying distributed re-keying and re-shuffling independently, we can speed-up computation significantly by using the pre-computation of $[Q]_q = [K^{-1} \cdot S]_q$, as the TRs do for every party. In more detail, we define *distributed re-key-shuffling*: on input of an encryption $\langle b,c \rangle$, and shared values $[Q]_q$

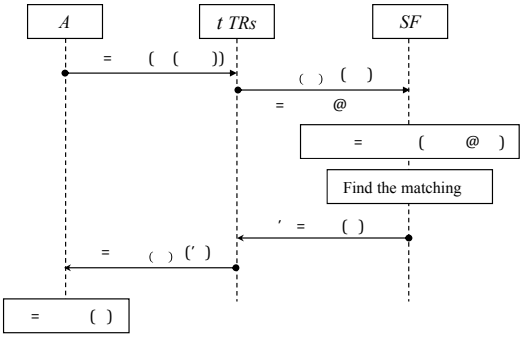

Figure 1: Data retrieving in the distributed PEP.

and $[S]_q$ with $Q, S < q$, re-key-shuffling is performed using the exponentiation protocol of Section 2.3:

$$RKS_{Q,S}(\langle b,c\rangle) = \langle \exp(b,[Q]_q), \exp(b,[S]_q)\rangle \bmod p. \tag{10}$$

Re-key-shuffling by design satisfies $RKS_{K^{-1},S,S} = RS_S \circ RK_{K^{-1}}$.

Figure 1 shows the process of retrieving data from SF in the distributed PEP, where an authorized party A requests to obtain data D belong to a patient pid_P. The main differences between the original PEP and the distributed version in retrieving data are the operations $\text{epid}_P@SF$ and R that are performed in a distributed fashion by using t TRs.

4.2 Generating New Key and Pseudonym-factors

Let A be a new party given permission to access some data owned by patient P. At least $2t-1$ TRs (refer to Section 2.3.2) create secret sharings $[S_A]_q$, $([K_A]_q, [K_A^{-1}]_q)$, and $[Q_A]_q = [K_A^{-1} \cdot S_A]_q$. For any TR not involved in the generation process, shares can be generated using Lagrange interpolation.

4.3 Adding New Transcryptor

After setup, it is possible to add or replace a TR. Let TR_k be a new TR joining the setup. Given a (t,n) secret sharing of secret $Z \in \mathbb{G}_q$, new shares for Z can be generated using t TRs. Let π be a quorum of t TRs. First, TR_k is assigned a unique point u_k not used by any other TR. Every TR $i \in \pi$ computes a partial result for the secret share of Z for k as

$$Z_{k,i} = Z_i \prod_{j \in \pi, j \neq i} \frac{u_k - u_j}{u_i - u_j} \bmod p, \tag{11}$$

and sends $Z_{k,i}$ to k. k can compute his secret share of Z using

$$Z_k = \sum_{i \in \pi} Z_{k,i} \bmod p. \tag{12}$$

This way TR_k can be given secret shares of all pseudonym-factors, key-factors, and secret shares of any pre-computations.

4.4 Master Key Generation

The role of a trusted key server can instead be distributed among t' TRs. Secret key x becomes a (t',n) threshold sharing of a secret random value. The exponentiation protocol of Section 2.3 allows the parties to jointly compute and reveal $y = g^x$.

For any participant P with (t,n) secretly shared key-factor K_P, $t'+t-1$ TRs can collaborate to compute a share of the private key x_P of P. First they compute $[x_P]_q = [x \cdot K_P]_q = \text{mult}([x]_q, [K_P]_q)$. The TRs can send their partial results to P, who can combine those to obtain his secret key x_P.

Note that throughout this process no TR ever learns nor holds the value of x. Furthermore, the secret sharing of x can be done using t' greater than t, improving the security of x. Also instead of giving all TRs secret shares for x, shares can be given only to TRs with increased trust.

5 SECURITY OF DISTRIBUTED PEP

In this section, we provide proofs to show that our distributed PEP is simulation secure in the semi-honest security model. Informally, we mean that the probability that an adversary can learn private information from truly generated data by the parties in our protocols is at most negligibly more than the probability that an adversary can learn from given randomly generated data. We use the simulatability paradigm (Lindell, 2017) in our proofs, where the adversary takes the control of the network and tries to obtain the final result of the protocol by itself as the only party in the protocol. In this paradigm, security is defined as a comparison of computation work-flow in the "real world" and the "ideal world".

In real world, a protocol can be broken into subprotocols or computations that are carried out by each party throughout the protocol. Let us denote π as the distributed PEP protocol; we can split π into two parts: $\pi = \pi_{TR}$ and π_{SF}, which are performed in a quorum of t TRs and SF, respectively. In retrieving data, as an instance, π_{TR} takes $RR_r(\text{ppid}_P)$ as an input from A and outputs $\text{epid}_P@SF$ to π_{SF}. Then, π_{SF} decrypts the given encryptions from $\pi = \pi_{TR}$ to obtain $\text{pid}_P@SF$ and starts searching for corresponding ciphertext C. Afterwards, π_{SF} re-randomizes C and

sends it to $\pi = \pi_{TR}$. Thus, to retrieve specific encrypted data from the storage facility the encrypted messages flow from one party to another party and together they send back the result to the party A. Assuming SF is corrupted by an adversary $\mathcal{A}$, then $\mathcal{A}$ has access to $\text{pid}_P@SF$, and the encrypted search result C. Similarly, when one of the TRs is corrupted, the adversary has access to the intermediate computation results.

In an ideal world, it is assumed that one of the parties is corrupted by an adversary. Then, he uses a simulator to generate the outputs of the other parties. This would be similar to performing the protocol with just one corrupted party. In the ideal world, an adversary $\acute{\mathcal{A}}$, who has control over SF, has only access to its pair of $(\text{pid}_P@SF, C)$, and the garbage inputs given from a simulator instead of the correct result of π_{TR}. The goal is to show that $\mathcal{A}$ can learn equal or negligibly more than $\acute{\mathcal{A}}$, meaning that they are computationally indistinguishable. Then we can conclude that the protocol is a simulation secure protocol.

5.1 Security of Key Generation

Since, in our setting, the TRs are assumed to be semi-trusted (not fully trusted like original PEP) we need to show that any set of corrupted $t-1$ TRs cannot deduce any information about the secret.

Definition 5.1. Let the shared secret be a value S, and algorithm π be a (t,n) threshold secret sharing scheme, where $\pi(S) = [s_1, \cdots, s_n]$. Then, any adversary $\mathcal{A}$ that has access to $t-1$ shares from $[s_1, \cdots, s_n]$ cannot infer any information about the values of S.

5.2 Security of Storing Data

In the process of storing data, SF does not have any input and is denoted by ϕ. $output_i^{StoringData}(PP, C; S_{SF_j}, K_{SF_j}; \phi; n)$ represents the output of each party in storing data. The joint output of two parties TRs and SF can be represented as:

$$output^{StoringData}(PP, C; S_{SF_j}, K_{SF_j}; \phi; n)) = (output_1^{StoringData}(PP, C; S_{SF_j}, K_{SF_j}; \phi; n), \quad output_2^{StoringData}(PP, C; S_{SF_j}, K_{SF_j}; \phi; n)). \tag{13}$$

Definition 5.2. the process of storing data securely computes $f = (DS_f, TR_f, SF_f)$ in the semi-honest security setting if there exist PPT algorithms Sim_{TR} and Sim_{SF} such that:

$$\{(Sim_{TR}(1^n, S_{SF}, K_{SF}, TR_f, f))\} \stackrel{c}{\equiv} \{(view_{TR}^f(PP, C; S_{SF_j}, K_{SF_j}; \phi; n), output^f(PP, C; S_{SF_j}, K_{SF_j}; \phi; n))\}. \tag{14}$$

and

$$\{(Sim_{SF}(1^n, \phi, SF_f, f))\} \stackrel{c}{\equiv} \{(view_{SF}^f(PP, C; S_{SF_j}, K_{SF_j}; \phi; n), output^f(PP, C; S_{SF_j}, K_{SF_j}; \phi; n))\}. \tag{15}$$

Denote the computation of $PP = RR_r(\text{ppid}_P)$ in the data supplier as DS_{f_1}, $C = E_y(D)$ as DS_{f_2}, $\text{epid}_P@SF = RS_{S_{SF}}(RK_{K_{SF}}(PP))$ as TR_{f_1}, and $\text{pid}_P@SF = D_{y_{SF}}(\text{epid}_P@SF)$ as SF_{f_1}. Let $f = (DS_f, TR_f, SF_f)$, where $DS_f = (DS_{f_1}, DS_{f_2})$, $TR_f = (TR_{f_1})$, $SF_f = (SF_{f_1})$ the f to be the PPT functionality for storing data. The view of the i^{th} party $i \in \{DS, TR, SF\}$ for storing data on $(PP, C; S_{SF_j}, K_{SF_j}; \phi)$ and security parameter n is denoted by $view_i^{StoringData}(PP, C; S_{SF_j}, K_{SF_j}; \phi; n) = (w, r_i; m_1^i, \cdots, m_v^i)$, where $j \in \{1, \cdots, t\}$, $w \in \{PP, C, S_{SF}, K_{SF}, \phi\}$ based on i, r_i is the random number for i^{th} party, and m_z^i is the z^{th} messages received by i^{th} party.

Theorem 1. *The protocol for storing data using distributed PEP securely computes the functionality f, when at most $t-1$ TRs are corrupted by adversary $\mathcal{A}$ in the presence of semi-honest adversaries.*

Proof. We need to show that any set of $t-1$ colluding TRs cannot computationally distinguish between generated messages and outputs from $\mathcal{S}_1$ that is the simulation of DS, and randomly generated data. TRs receive two outputs from $\mathcal{S}_1$, PP and C. Given S_{SF}, K_{SF}, and 1^n (security parameter), storing data works as follow:

1. $\mathcal{S}_1$ chooses two random numbers r_1 and r_2 as $\acute{\text{pid}}_P$ and $\acute{D}$, respectively. Then, it performs $DS_{f_1}(r_1)$ and $DS_{f_2}(r_2)$ to obtain $\acute{PP}$ and $\acute{C}$ and sends them to a quorum of t TRs.
2. t TRs perform TR_{f_1} to obtain $\text{epid}_P@SF$, and send it to DS together with C.
3. $\mathcal{S}_3$ performs SF_{f_1} to obtain pid_P^{SF}, then stores the pair $(\text{pid}_P^{SF}, \acute{C})$ in its database.

The output of the simulation can be written $Sim_{TR}(1^n, S_{SF}, K_{SF}, TR_f, f) = (S_{SF}, K_{SF}; \acute{PP}, \acute{C})$. The real view of any TR involved in the protocol can be represented as $view_{TR}^f(S_{SF}, K_{SF}) = (S_{SF}, K_{SF}; PP, C)$. It can be observed that $\mathcal{A}$ cannot distinguish between C and $\acute{C}$ since the ElGamal cryptosystem is semantically secure. Moreover, since any set of $t-1$

corrupted TRs cannot decrypt any ciphertext; therefore, they are unable to distinguish between PP and $\acute{PP}$. □

Theorem 2. *The protocol for storing data using distributed PEP securely computes the functionality f, when SF is corrupted by adversary $\mathcal{A}$ in the presence of semi-honest adversaries.*

Proof. Assuming that SF is corrupted, we need to show that $\mathcal{A}$ cannot distinguish between random generated results from other parties and the correct versions. SF receives two outputs from $\mathcal{S}_2$, as a simulator for TRs, that are $\text{epid}_P@SF$ and C. The process of storing data with the simulator $\mathcal{S}_2$, and given 1^n as the security parameter is as follows:

1. $\mathcal{S}_2$ chooses three random numbers r_1, r_2, and r_3 as $\acute{\text{pid}}_P$, $\acute{D}$, and S_{SF}, respectively, then encrypts them under public key y. Afterwards, $\mathcal{S}_2$ performs TR_{f_1} to obtain $\acute{\text{epid}}_P@SF$ and sends $(\acute{\text{epid}}_P@SF, \acute{C})$ to SF.
2. SF performs SF_{f_1} to get $\acute{\text{pid}}_P^{SF}$, then stores the pair $(\acute{\text{pid}}_P^{SF}, \acute{C})$ in its database.

SF does not have access to the private key of DS, thus it cannot decrypt $\acute{C}$ and distinguish between $\acute{C}$ and C since the ElGamal cryptosystem is semantically secure. Moreover, TRs re-shuffle the encrypted id before sending it to SF; therefore, SF cannot link $\acute{pid}_P^{SF}$ back to its owner. Therefore, we can conclude that

$$\{(Sim_{SF}(1^n, \phi, SF_f, f))\} \stackrel{c}{\equiv} \{(view_{SF}^f(PP, C; S_{SF_j}, K_{SF_j}; \phi; n), output^f(PP, C; S_{SF_j}, K_{SF_j}; \phi; n))\}. \quad (16)$$

□

5.3 Security of Retrieving Data

Assume that party A is authorized to retrieve the data for pid_P, denote the computation of $\text{ppid}_P = E_y(\text{pid}_P)$ as A_{f_1}, $PP = RR_r(\text{ppid}_P)$ as A_{f_2}, $\text{epid}_P@SF = RS_{S_{SF}}(RK_{K_{SF}}(PP))$ as TR_{f_1}, $\text{pid}_P^{S_{SF}} = D_{y_{SF}}(\text{epid}_P@SF)$ as SF_{f_1}, $C' = RR_s(C)$ as SF_{f_2}, and $R = RK_{K_A}(C')$ as TR_{f_2}. Then, $f = (A_f, TR_f, SF_f)$, where $A_f = (A_{f_1}, A_{f_2})$, $TR_f = (TR_{f_1}, TR_{f_2})$, and $SF_f = (SF_{f_1}, SF_{f_2})$.

Theorem 3. *The protocol for retrieving data using distributed PEP securely computes the functionality f, when at most $t-1$ TRs are corrupted by adversary $\mathcal{A}$ in the presence of semi-honest adversaries.*

Proof. We need to show that any of the TRs or a quorum of $t-1$ TRs cannot infer any information about the private data after performing the protocol for retrieving data. Let $\mathcal{S}_1$ be the simulator of party A and $\mathcal{S}_3$ the simulator of SF. Given S_{SF} and K_{SF} as the inputs of TRs and 1^n as a security parameter, retrieving data from SF for party A works as follows:

1. $\mathcal{S}_1$ chooses a random number r_1 and performs $A_{f_1}(r_1)$ and A_{f_2} to obtain $\acute{PP}$ and sends it to a set of t TRs.
2. TRs receive $\acute{PP}$ and perform TR_{f_1} to get $\text{epid}_P@SF$, then send it to $\mathcal{S}_3$.
3. $\mathcal{S}_3$ performs SF_{f_1}, encrypts a random number to get C, executes $\hat{C} = SF_{f_2}(C)$, and sends $\hat{C}$ to TRs.
4. TRs perform TR_{f_2} and send R to $\mathcal{S}_1$.

The output of the simulation can be written $Sim_{TR}(1^n, S_{SF}, K_{SF}, TR_f, f) = (S_{SF}, K_{SF}; \acute{PP}, \hat{C})$. The real view of any TR involved in the protocol can be represented as $view_{TR}^f(S_{SF}, K_{SF}) = (S_{SF}, K_{SF}; PP, \acute{C})$. Since having at most $t-1$ corrupted TRs does not enable the decryption of the given messages, the TRs cannot distinguish between PP and $\acute{PP}$, and $\acute{C}$ and $\hat{C}$. □

Theorem 4. *The protocol for retrieving data using distributed PEP securely computes the functionality f, when SF is corrupted by adversary $\mathcal{A}$ in the presence of semi-honest adversaries.*

Proof. In the process of retrieving data securely, SF plays an important role in finding the data in the remote database. If SF becomes corrupted by an adversary $\mathcal{A}$, we need to guarantee that $\mathcal{A}$ can infer any sensitive information from the given messages. SF receives $\text{epid}_P@SF$ from the TRs, decrypts it, and finds the matching data. Let $\mathcal{S}_2$ be the simulator for TRs, then the process of retrieving data is as follows:

1. $\mathcal{S}_2$ chooses two random numbers r_1 and r_2 as $\acute{PP}$ and S_{SF}, respectively. Afterwards, $\mathcal{S}_2$ performs $TR_{f_1}(\acute{PP}, r_2)$ to obtain $\acute{\text{epid}}_P@SF$ and sends it to SF.
2. SF performs $SF_{f_1}(\acute{\text{epid}}_P@SF)$ to get $\acute{\text{pid}}_P^{SF}$ and checks all pairs of $(\text{pid}_P^{SF}, \acute{C})$ in its database to find the matching data. Then, it executes SF_{f_2} and sends the result back to the TRs.

The output of the simulation for retrieving data when SF is corrupted is $Sim_{TR}(1^n, \text{pid}_P^{SF}, \acute{C}, SF_f, f) = (\text{pid}_P^{SF}, \acute{C}; \acute{\text{epid}}_P@SF)$. The real view of any TR involved in the protocol can be represented as $view_{SF}^f(\text{pid}_P^{SF}, \acute{C}; \text{epid}_P@SF)$. Therefore. if $\mathcal{A}$ cannot distinguish between $\text{epid}_P@SF$ and $\acute{\text{epid}}_P@SF$,

then we can conclude the protocol is simulation secure. Note that both ciphertexts are encrypted with K_{SF}, thus SF can decrypt both to obtain pid_P^{SF} and $\acute{\text{pid}}_P^{SF}$. However, $\mathcal{A}$ is still unable to distinguish between pid_P^{SF} and $\acute{\text{pid}}_P^{SF}$, since they are re-shuffled with different S_{SF} values and $\mathcal{A}$ cannot track back to check who is the owner of the data. □

6 EFFICIENCY

6.1 Complexity

A complexity analysis of our distributed *RR*, *RK*, *RS*, and *RKS* is given in Table 2. Here, the total amount of multiplications and exponentiations is computed in case t TRs participate. The merging of the precomputations (phase (ii) in *RR*, and the combination step for exponentiation, Eq. (3)) is assumed to be evaluated by one participant. All operations show a complexity of $O(t)$. For comparison, in the original PEP, operations have the complexity of $O(1)$.

Table 2: Computational complexity of distributed PEP operations.

Operation	Mult.	Exponen.	Overall
RR	$2t+2$	0	$O(t)$
RK	t	t	$O(t)$
RS	$2t$	$2t$	$O(t)$
RKS	$2t$	$2t$	$O(t)$

For any distributed PEP operation the analysis of the message complexity is as follows: a participant needs to send a polymorphic pseudonym and encrypted data C to t TRs, who subsequently perform their operations and return their partial results to the single receiver. This gives us a tight bound of $\theta(2t)$ messages.

6.2 Performance

Although we have provided a distributed *RR* operation, this operation is not used by the TRs in our particular application, and we focus on distributed *RKS*. We have implemented the protocol using C++ and the GMP library[2], and tested their performance on a machine running Windows 10.0, with an Intel Core i5-7200 running at 2.50GHz.

To have a clear view on the cost of the individual steps in the protocol, we have separated *RKS* as follows. Let π be a set of unique TRs in n with $|\pi| = t$. Let Q_i, S_i be the partial results of the Lagrange interpolation of secret shares $[Q]_q$ and $[S]_q$ for TR i (cf., Eq. (1)).

- Every TR $i \in \pi$ computes a partial result as

$$PRKS_{Q_i,S_i}(\langle b,c\rangle) := \langle b_i,c_i\rangle = \langle b^{Q_i}, c^{S_i}\rangle . \quad (17)$$

- One party combines the partial results as

$$CPR(\{\langle b_i,c_i\rangle\}_{i\in\pi}) := \langle \prod_{i\in\pi} b_i, \prod_{i\in\pi} c_i\rangle . \quad (18)$$

Table 3 gives the computation time for performing t *PRKS* operations and combining the partial results using *CPR*, for different values of t. The *PRKS* computation time is the average time spend by a single TR, averaged over 1000 evaluations.

Table 3: Average computation time of 1000 *PRKS* and *CPR* operations.

t	*PRKS* (ms)	*CPR* (ms)
10	707	26
19	754	68
31	776	95
40	803	128

7 CONCLUSION

In this work, we proposed a solution for controlling access in a fully distributed manner. The solution relies on multiple applications of Secret Sharing, along with multiparty computation protocols on shared values. Our solution performs well complexity and performance wise, and is unconditionally secure as long as no "too large group" collides. If combined with the original PEP protocol by Verheul et al. (Verheul et al., 2016), our distributed transcryptor thus allows for improved security, while at the same time facilitating scalability and robustness.

REFERENCES

Algesheimer, J., Camenisch, J., and Shoup, V. (2002). Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In Yung, M., editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432. Springer.

Bar-Ilan, J. and Beaver, D. (1989). Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In Rudnicki, P., editor, *Proceedings of*

[2]See https://gmplib.org.

the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989, pages 201–209. ACM.

Beaver, D. (2000). Minimal-latency secure function evaluation. In (Preneel, 2000), pages 335–350.

Ben-Or, M., Goldwasser, S., and Wigderson, A. (1988). Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Simon, J., editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM.

Camenisch, J. and Lehmann, A. (2017). Privacy-preserving user-auditable pseudonym systems. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 269–284. IEEE.

Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society.

Cramer, R., Damgård, I., and Maurer, U. M. (2000). General secure multi-party computation from any linear secret-sharing scheme. In (Preneel, 2000), pages 316–334.

Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J. B., and Toft, T. (2006). Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Halevi, S. and Rabin, T., editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer.

Desmedt, Y. and Frankel, Y. (1989). Threshold cryptosystems. In Brassard, G., editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer.

El Gamal, T. (1984). A public key cryptosystem and a signature scheme based on discrete logarithms. In Blakley, G. R. and Chaum, D., editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer.

Jakobsson, M. (1999). On quorum controlled asymmetric proxy re-encryption. In Imai, H. and Zheng, Y., editors, *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings*, volume 1560 of *Lecture Notes in Computer Science*, pages 112–121. Springer.

Lindell, Y. (2017). How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography.*, pages 277–346.

Page, T. (2015). A forecast of the adoption of wearable technology. *IJTD*, 6(2):12–29.

PEP (2017). Polymorphic encryption and pseudonymisation for personalised healthcare. https://pep.cs.ru.nl.

Preneel, B., editor (2000). *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*. Springer.

Shamir, A. (1979). How to share a secret. *Commun. ACM*, 22(11):612–613.

Toft, T. (2007). *Primitives and Applications for Multi-party Computation*. PhD thesis, University of Aarhus, Aarhus.

Toft, T. (2009). Constant-rounds, almost-linear bit-decomposition of secret shared values. In Fischlin, M., editor, *Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*, volume 5473 of *Lecture Notes in Computer Science*, pages 357–371. Springer.

Verheul, E., Jacobs, B., Meijer, C., Hildebrandt, M., and de Ruiter, J. (2016). Polymorphic encryption and pseudonymisation for personalised healthcare. Cryptology ePrint Archive, Report 2016/411.

Wearables (2017). Wearables market to be worth $25 billion by 2019. http://www.ccsinsight.com/press.