

This is the final peer-reviewed accepted manuscript of:

*DLT-Based Personal Data Access Control with Key-Redistribution*

**Conference Proceedings:** 2023 Fifth International Conference on Blockchain Computing and Applications (BCCA), October 24-26, 2023, Kuwait, Kuwait.

**Author:** Fadi Barbàra; Mirko Zichichi; Stefano Ferretti; Claudio Schifanella

**Publisher:** IEEE

The final published version is available online at:

<https://doi.org/10.1109/BCCA58897.2023.10338895>

#### Rights / License:

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website:

[https://www.ieee.org/content/dam/ieee-org/ieee/web/org/pubs/author\\_version\\_faq.pdf](https://www.ieee.org/content/dam/ieee-org/ieee/web/org/pubs/author_version_faq.pdf)

This item was downloaded from the author personal website (<https://mirkozichichi.me>)

When citing, please refer to the published version.

# DLT-based Personal Data Access Control with Key-Redistribution

Fadi Barbàra<sup>\*</sup>, Mirko Zichichi<sup>†</sup>, Stefano Ferretti<sup>‡</sup>, Claudio Schifanella<sup>\*</sup>

<sup>\*</sup>*Department of Computer Science, University of Turin, Italy*

<sup>†</sup>*Ontology Engineering Group, Universidad Politécnica de Madrid, Spain*

<sup>‡</sup>*Department of Pure and Applied Sciences, University of Urbino Carlo Bo, Italy*

*fadi.barbara@unito.it, mirko.zichichi@upm.es, stefano.ferretti@uniurb.it, claudio.schifanella@unito.it*

**Abstract**—Data management services present a challenge in terms of trust, as service managers can access the data on their servers easily. Decentralized data services and smart contracts can solve problems related to the presence of centralized trusted authorities, but in turn they can introduce other issues related to compliance with data protection and regulations (e.g., GDPR). Historically, encryption has been used to address some of these concerns, but it restricts data sharing. To facilitate encrypted decentralized file storage while enabling data sharing, we propose a Key-Redistribution Proxy Re-Encryption (KeRePRE) system. KeRePRE is a decentralized and encrypted data-service, where authorization servers are part of a threshold proxy re-encryption scheme. A key-redistribution mechanism (that extends the Umbral scheme) allows for the addition and removal of managers in a decentralized and trustless manner. Additionally, we offer a proof of concept implementation, where data access control is based on an access control list, implemented as a smart contract in a DLT, and can be read-only accessed by the authorization servers.

**Index Terms**—Proxy re-encryption, Threshold scheme, GDPR, Data Sharing, Decentralized File System

## I. INTRODUCTION

Nowadays, data are of high value to individuals, businesses, and governments. The increasing amount of data generated by various sources like mobile devices, sensors, and social media has given rise to big data and data-driven decision-making. However, the volume and complexity of data have created significant challenges in managing, processing, analyzing, and protecting them, especially regarding the new directives related to data protection such as GDPR [1]. To overcome these challenges, data intermediation has become crucial. A data intermediary acts as an intermediary between data holders and recipients, facilitating the flow of data while ensuring its quality and security.

Decentralized systems such as blockchains or Distributed Ledger Technologies (DLTs) have gained considerable attention in data intermediation. These systems allow data recipients and processors to be limited to the data holder’s instructions through smart contracts, enabling intermediation with a higher level of control over data, see e.g. [2]. Decentralization, however, necessitates cryptography to secure decentralized systems and protect data while sharing it. In a trustless or semi-trusted decentralized system, comprehensive security is essential to maintain user trust. To solve these problems, approaches such as  $(t, n)$ -threshold cryptosystems

involve multiple parties performing a cryptographic operation together, using a share of a secret in a secret-sharing scheme, in order to transform one central party into a committee. In particular, a Threshold Proxy Re-Encryption (TPRE) scheme delegates some data intermediaries (proxies) to re-encrypt the encryption key in favor of a data receiver [3]–[5].

However, most threshold cryptosystems assume that data intermediaries will protect their secret shares, which works as a theoretical assumption but does not reflect the real-world conditions. For example, Verichain recently exposed a critical key-extraction attack [6], which invalidates implementations of commonly used threshold cryptosystems. Similarly, the security firm Trail of Bits found a critical bug in common threshold signature libraries [7]. In these cases, the presence of a key redistribution would have been put to good use, in order to remove the vulnerability from the present committee. In this paper, we propose a Key-Redistribution Proxy Re-Encryption (KeRePRE) for TPREs to manage situations where parties lose their share, are corrupted or faulty, in a semi-trusted decentralized environment. The mechanism enables key rotation, addition, or deletion [8] and helps manage compromised or leaked keys and handle situations when new parties are added or old parties are dismissed.

Furthermore, we focus on a specific strand of literature concerned with data access control managed using cryptographic techniques in a decentralized manner. The literature usually deals with personal data, which requires strong data protection and security mechanisms because they identify or render identifiable a data subject [2], [3], [9], [10]. We propose a DLT based Personal Data Store (PDS) to enable data subjects to decide how (through smart contracts) and where to store their data and handle data encryption and key distribution using TPRE with the key redistribution mechanism. The PDS comprises two main components, Decentralized File Storage (DFS), which stores the data to encrypt or decrypt, and a DLT, which avoids the typical *DR*awbacks of server-based approaches, such as censorship or single-point-of-failure, and offers features like data traceability, verifiability, and smart contract execution.

*Our Contributions:* In summary our contributions are:

- Our proposal, KeRePRE, builds upon Nunez’s previous work [5] and introduces a novel threshold-based proxy re-encryption scheme that supports key-redistribution.

- To showcase the feasibility of our proposal, we have developed an independent implementation specifically for the Umbral system. The code is publicly available on GitHub<sup>1</sup>.
- Our proposed scheme, KeRePRE, has the potential to create a decentralized Personal Data Store (PDS) linked to a Distributed Ledger Technology (DLT) that is fully compliant with the GDPR regulations. We demonstrate this capability in our research.

*Outline:* The structure of this paper is as follows. In Section II, we introduce the building blocks that comprise the KeRePRE system. In Section III, we describe how KeRePRE operates within the context of the Umbral system. We then demonstrate how KeRePRE operates and analyze its security and data protection enhancements in Section IV and Section V, respectively. In Section VI, we present an overview of other works in the fields of threshold proxy re-encryption schemes and personal data sharing mechanisms that involve a Distributed Ledger Technology (DLT). Finally, we conclude in Section VII.

## II. BACKGROUND

In this section, we give an overview of the system’s architectural components needed to understand Section III. Throughout this section, we assume two parties: a data-holder (DH) which has some data stored in a decentralized personal data-space (PDS) and a data-receiver (DR) which wants to access it.

### A. Threshold cryptosystems

A  $(t, n)$ -threshold cryptosystem involves multiple parties in a set  $\mathcal{P} = P_{i \in \mathcal{I}}$  performing a cryptographic operation together. The key feature of this system is that a minimum number of parties, referred to as the “threshold”  $t$ , must participate for the cryptosystem to succeed. In other words, the cryptosystem will only work if at least  $t$  out of the  $n$  parties are honest and follow the protocol. Since generally a cryptosystem is used to give access to some information, we say that  $\Sigma = (t, n)$  is an *access structure*.

---

#### Algorithm 1 LssPrep for a $(t, n)$ secret sharing

---

**Require:**  $\mathcal{I}$ : set of identities of parties,  $t$ : threshold,  $s$ : secret

- 1:  $n = \text{len } \mathcal{I}$
- 2: **for**  $i = 1 \dots t - 1$  **do**  $\triangleright$  Initialize  $a_i$  for  $i = 0, \dots, t - 1$
- 3:      $a_i \leftarrow_{\mathcal{S}} \mathbb{F}$
- 4: **end for**
- 5:  $a_0 = s$   $\triangleright q(0) = s$
- 6: Initialize  $q(\cdot) = a_0 + \sum_{i=1}^{t-1} a_i \cdot i$   $\triangleright$  Polynomial initialization
- 7: **for**  $i$  in  $\mathcal{I}$  **do**  $\triangleright$  The values of  $\mathcal{I}$  must be numbers in the field  $\mathbb{F}$
- 8:     Send  $(i, q(i))$  to party  $i$
- 9: **end for**

---

Two research strands involve threshold cryptosystems: threshold signing and secret sharing. On the one hand,  $(t, n)$ -threshold signing, considered to have been introduced by Desmedt in 1987 [11], is a process where  $t$ -of- $n$  parties are involved into signing a message on behalf of all  $n$  participants. On the other hand, in  $(t, n)$  secret sharing a secret  $s$  is split into  $n$  different parts called *shares* (or *fragments*) such that  $t$  of them are necessary to reconstruct the original  $s$ . Among many secret sharing schemes, we focus on the Shamir Secret Sharing (SSS) one since it is the one used in both Umbral and the redistribution mechanism our work is based on.

The scheme is based on polynomial interpolation over a field. In a field  $\mathbb{F}$ , it is well known that given  $t$  points in the 2-dimensional plane  $\{(x_i, y_i)\}_{i=1}^t$  there is one and only one polynomial  $q(x)$  of degree  $t - 1$  such that  $q(x_i) = y_i$  for each  $i = 1, \dots, t$ . Assume a *secret* number  $s$ . As mentioned, this secret can be split and shared to  $n$  parties in such a way that  $t$  of those shares are needed to reconstruct  $s$ : see LssPrep in Algorithm 1.

Since the shares are distinct points on a plane for polynomial  $q(\cdot)$ , the SSS scheme uses Lagrange Polynomials applied to the shares, as presented in Algorithm 2, to reconstruct the secret.

---

#### Algorithm 2 LssRec for a $(t, n)$ secret sharing

---

**Require:**  $\mathcal{I}$ : set of identities of parties,  $t$ : threshold

- 1:  $n = \text{len } \mathcal{I}$
- 2: Wait for  $t$  shares  $i_{j_1}, q(i_{j_1})$  from parties  $i_{j_1}, \dots, i_{j_t} \in \mathcal{I}$
- 3:  $L = 0$
- 4: **for**  $k = 1 \dots t$  **do**
- 5:      $\lambda_k = \prod_{w=1, w \neq k}^t \frac{i_{j_w}}{i_{j_w} - i_{j_k}}$   $\triangleright$  Create a Lagrange basis
- 6:      $L = L + \lambda_k q(i_{j_k})$   $\triangleright$  Use Lagrange polynomials to create  $q(0) = s$
- 7: **end for**
- 8: **return**  $L$   $\triangleright L = q(0) = s$

---

### B. Proxy re-encryption schemes

A well-known problem faced by data holders (DHs) in a decentralized PDS is the lack of direct control over the outsourced data in  $\mathcal{E}$  [12], [13], which raises security concerns, especially in (pseudo) anonymous settings.

One of the most effective ways to deal with this issue is for the *DH* to encrypt the data before uploading it to the PDS [14]. This naive solution, though, inhibits the delegation of access (i.e. “sharing”) to a data receiver *DR* of a piece of data  $pd_i$ , since the process requires *DH* to download, re-encrypt  $pd_i$  for *DR* and re-upload  $pd_i$ . To solve this issue Blaze *et al.* introduced the Proxy Re-Encryption (PRE) scheme in [15]. A PRE is a semi-trusted proxy that transforms a cyphertext encrypted for *DH* to a cyphertext encrypted for *DR*, without decrypting the cyphertext or leaking the related plaintext. Specifically, with a PRE, *DH* can encrypt  $pd_i$  under its own public key before uploading it to the PDS. Then, after receiving the request of data sharing from *DR*, *DH* can generate a *proxy re-encryption key* and send it to the PRE. The

<sup>1</sup><https://github.com/disnocen/umbral-rs>

PRE is then able to re-encrypt  $pd_i$  into a cyphertext under the public key of  $DR$ .

While these cryptographic primitives solve the privacy problem (the proxy can not read the data stored/sent), it does not solve the *ensorship* problem: the proxy can block any request and decline any sharing of the data. In that regard, the proxy acts as a trusted and custodial (of data) party.

Similarly to other projects, it is possible to mitigate this risk by employing multiple nodes together and create a threshold cryptosystem for proxy re-encryption nodes, in other words a threshold PRE. One way to achieve that is explained in the work by Nunez [5], which is called Umbral. Umbral is a threshold PRE which uses a Key Encapsulation Mechanism (KEM) to obtain a Data Encryption Method (DEM). More explicitly, in Umbral, each file  $pd \in \mathcal{D}$  from a  $DH$  is encrypted with a symmetric key  $K \in \mathcal{K}$ . The encrypted file is a couple  $(\text{Enc}(pd, K), \text{Enc}_{pk_{DH}}(K))$ . The Umbral threshold PRE leverages the ReKeyGen procedure to output multiple shares of the re-encryption key via a SSS scheme. These shares are called fragments or more concisely  $kFrag$ , in [5], and are distributed to the node operators as part of the ReKeyGen routine. More formally:

**Definition 1** (TPRE). A  $(t, n)$ -threshold proxy re-encryption scheme (TPRE) is a tuple of algorithms (KeyGen, ReKeyGen, Encapsulate, ReEncapsulate, DecapsulateFragments):

- $(sk_A, pk_A) \leftarrow \text{KeyGen}(1^\lambda)$  On input security parameter  $\lambda$ , the key generation algorithm KeyGen outputs a pair of secret and public keys  $(sk_A, pk_A)$  for user A.
- $kFrag_1, \dots, kFrag_n \leftarrow \text{ReKeyGen}(sk_A, pk_B, n, t)$ : On input the secret key  $sk_A$  of user A (generally the  $DH$ ), the public key  $pk_B$  of user B (generally the  $DR$ ), a number of shares  $n$  and a threshold  $t$ , the re-encryption key generation algorithm ReKeyGen computes the re-encryption key  $rk_{A \rightarrow B}$  and then uses SSS scheme to share it in  $n$  different  $kFrag$ s, where  $kFrag_i = (id_i, rk_i, opt)$ , with  $id_i$  the identity of node  $i$ ,  $rk_i$  its share of the re-encryption key and  $opt$  optional arguments depending on the implementation.
- $(K, \gamma_K) \leftarrow \text{Encapsulate}(pk_A)$ : On input the public key  $pk_A$  of user A, the algorithm Encapsulate outputs a symmetric key  $K \in \mathcal{K}$  used to encrypt the data and a capsule  $\gamma_K = \text{Enc}(K)$ .
- $cFrag_i \leftarrow \text{ReEncapsulate}(kFrag_i, \gamma_K)$ : On input a key share  $kFrag_i$  and a capsule  $\gamma_K$ , algorithm ReEncapsulate outputs a share (or fragment) of the capsule  $cFrag_i$  of the capsule  $\gamma_K$ .
- $K \leftarrow \text{DecapsulateFragments}(sk_B, pk_A, \{cFrag\}_{i=1}^n)$ : On input the secret key  $sk_B$  of user B, the public key  $pk_A$  of user A and at least  $t$   $cFrag$ s, algorithm DecapsulateFragments outputs  $K$  (note that it is the same  $K$  of algorithm Encapsulate).

Figure 1 highlights the flow of the procedures which we explain in details in Section IV

From Definition 1, it is easy to see that a PRE is an extension of a public key encryption scheme (PKE). Therefore a PRE must follow the security models of PKEs which present an interesting challenge. On the one hand, PREs have to guarantee confidentiality and validity of the cyphertexts as any PKE. On the other, PREs have to allow re-encryption of cyphertexts. A thorough overview of how different schemes deal with the challenge is presented by Nunez *et al.* in [5].

### C. Key redistribution mechanisms

Most threshold cryptosystems, and particularly secret sharing schemes, assume parties will take good care of their share. In fact, if some party  $P_i$  loses a share, it is generally said that  $P_i$  is corrupted or faulty. No further analysis on  $P_i$  is done, since from the point of view of the threshold cryptosystem, no single party is important as long as the majority or minority of them is still honest, depending on the access structure of the cryptosystem.

On the other hand, real world deployments of these systems have to deal with such problems. For example in Proactive Secret Sharing schemes [16], [17] the participants refresh (or rotate) their key shares periodically, in order to avoid these kinds of problems or at least mitigating them. The process is known as key-refresh or key-rotation.

However, in proactive secret sharing schemes, the access structure is not changed: the set of parties required for threshold secret sharing are the same before and after the key-refresh. Therefore, the only way to extend or shrink the access structure once it is in place is by performing a new distribution of the shares. This is costly, since it requires  $DH$  to recompute all the shares. Consequently, new approaches have been proposed in the literature to deal with this issue.

Among those proposals, one that is beneficial for the goal of this paper is the process of *redistribution* of shares. Unlike key refreshing schemes, a redistribution of shares is performed by the AS, supports the change of the access structure and requires no input by the  $DH$  (beside some authorization if needed by the general system).

In this proposal and proof of concept<sup>2</sup>, we use a redistribution method as presented by Desmedt *et al.* in [11]. The method leverages a SSS scheme once more and treats each share as a secret on its own. More formally, given a  $(t, n)$ -SSS scheme with shares  $s_1, \dots, s_n$ :

- $s'_1, \dots, s'_m \leftarrow \text{DesRedistr}(s_1, \dots, s_n)$ : On input  $t \leq k \leq n$  shares from a  $(t, n)$ -SSS for secret  $s$ , algorithm DesRedistr outputs  $m$  secret shares  $s'_1, \dots, s'_m$  such that  $k$  of them are needed to reconstruct  $s$ .

In practice DesRedistr transforms a  $(t, n)$ -SSS into a  $(k, m)$ -SSS.

It is easy to see that if  $k = t$  and  $m > n$  then DesRedistr adds a new party to the access structure, while if  $t < m < n$  then DesRedistr removes party from the access structure. We

<sup>2</sup>A working implementation of the key redistribution for KeRePRE can be found at <https://github.com/dinocen/umbral-rs/blob/master/src/internal/keyredistrib.rs>

TABLE I  
COMPARISON OF THE NAMES BETWEEN THE UMBRAL PROJECT AND OUR  
EXTENSION KeRePRE

KeRePRE	Umbral
Data Subject	N/A
Data Holder	Alice
DFS Provider	N/A
Authorization Server	Proxy Re-encryption Node
Data Recipient	Bob

will see in Section V-A the constraints on  $k$  and  $m$  related on  $t$  and  $n$ . A working example tailored to our purposes is presented in Algorithm 3.

### III. KEY REDISTRIBUTION

Umbral, which lacks a key redistribution mechanism, cannot be directly used for key rotation or redistribution. However, in this section, we demonstrate how we can expand Umbral's capabilities to create a threshold PRE that is suitable for real-world use. We will explain in Section IV how the system can be applied in a DLT-based PDS to enhance its security and perform tasks such as key addition, key deletion, or simple key-refresh. The terminology used is clarified in Section III-A, where we also introduce the actors and the architecture model. The key-redistribution mechanism in KeRePRE is divided into two algorithms: one for managing the  $kFrag$ s (Section III-B) and the other for managing the  $cFrag$ s (Section III-C).

#### A. Actors and architectural components

1) *Actors*: We define different actors that have one or more roles in the system. In detail, we identify the following actors:

- **Data subject** ( $DS$ ) - The natural person that uses a personal device that in turn generates personal data.
- **Data holder** ( $DH$ ) - The legal or natural person who has the right or obligation or the ability to make available specific data (both personal and non).
- **Data intermediary** ( $DI$ ) - The legal or natural person who mediates between those holders who wish to make their data available and data recipients. We have two specializations of data intermediary:
  - **DFS provider** ( $SP$ ) - The one that provides the access to the DFS. This actor provides functionalities attributed of storing and serving (encrypted) personal data.
  - **Authorization Server** ( $AS$ ) - The one that provides the access to the DLT to the authorization service, i.e., takes part to the cryptosystem.
- **Data recipient** ( $DR$ ) - The legal or natural person to whom the data holder makes data available.

Since the names of the actors involved in KeRePRE are different from the usual names due to the specific use case involved, we provide a comparison of the names for easy access in Table I

2) *Architecture Model*: In the following, we use a model to refer to the elements managed in the system.

- The data holder actor controls a set of personal data that have not been encrypted, i.e.,  $\mathcal{D} = \{pd_l \mid 1 \leq l \leq o\}$  where  $o$  is the amount of pieces of data  $DH$  has.
- Furthermore,  $\mathcal{K} = \{k_{pd_l} \mid \text{Enc}_{k_{pd_l}}(pd_l), 1 \leq l \leq o\}$  is the data holder's set of keys used to encrypt personal data and  $\mathcal{E} = \{epd_l \mid epd_l = \text{Enc}_{k_{pd_l}}(pd_l), 1 \leq l \leq o\}$  is the set of encrypted personal data.
- Data holders and authorization servers control a set of capsules  $\mathcal{C} = \{\gamma_{k_{pd_l}} \mid \gamma_{k_{pd_l}} = \text{Enc}_{pk_{DH}}(k_{pd_l}), 1 \leq l \leq o\}$ , where  $pk_{DH}$  is the public key of the data holder (see Definition 1), that contain a key used to encrypt a piece of personal data.
- We consider that all DFS providers  $SP$  store the data holders' set of encrypted personal data  $edp \in \mathcal{E}$  and the associated set of decentralized identifiers used to identify the  $epd$ . In this case, the decentralized identifier is equal to a hash pointer obtained by hashing the  $epd$ , i.e.,  $HP = \{hp_{epd_l} \mid hp_{epd_l} = \text{Hash}(epd_l), 1 \leq l \leq o\}$  where Hash is a predetermined hash function (e.g., in the IPFS DFS these hash pointers are CIDs). Thus,  $hp_{epd_x}$  is both the identifier of the  $epd_x$  datum in the DFS and the on-chain hash pointer, i.e., that will be stored in the DLT.

#### B. $kFrag$ redistribution

We assume that either  $DH$  or  $DR$  triggers a key redistribution. Note that this triggering may be part of a notification system in the application that asks  $DH$  or  $DR$  if they want to act to mitigate a potential threat (such as a share corruption in one of the operator nodes). We show a representation of the dynamics of the extended system in Figure 1.

A  $kFrag$  redistribution is a procedure that transforms a  $kFrag$  into a  $kFrag'$ . More formally:

- $(id, rk', opt) \leftarrow \text{kFragRedistr}(id, rk, opt)$ : On input a  $kFrag$ , the  $\text{kFragRedistr}$  algorithm outputs a new  $kFrag$  with the updated re-encryption key share.

In particular,  $\text{kFragRedistr}$  focuses on the update of the  $rk$  component into a  $rk'$  component. The complete  $\text{kFragRedistr}$  algorithm for a  $(t, n)$  threshold cryptosystem performed by each party  $P_i, i = 1 \dots n$  is presented in Algorithm 3.

To perform a key re-distribution, we use the Desmedt routine  $\text{DesRedistr}$  introduced in Section II-C (lines 7-14 of Algorithm 3). Instead of passing the  $id$  to the  $\text{LsssPrep}$  routine, as done in Algorithm 1, we pass the hashed id  $hid$  instead: this change is done to maintain compatibility with the original Umbral protocol and does not affect the security of the Desmedt key-redistribution protocol. On the other hand, note that we pass the actual  $id$  as parameter in the  $\text{LsssRec}$  at line 14.

#### C. $cFrag$ redistribution

As mentioned in Section II-B, for each  $kFrag$  there is a  $cFrag$ . The latter is used by  $DR$  to recover the data after a Re – Encryption has been performed by the node operators.

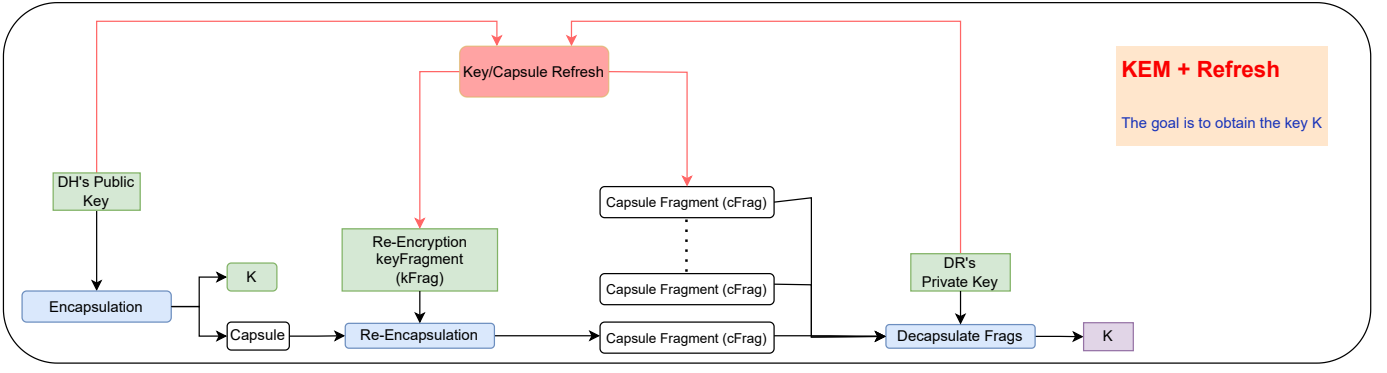


Fig. 1. The image represents the Umbral work flow with our key redistribution extension (in red). Either  $DH$  or  $DR$  can trigger a key redistribution procedure. The nodes in the threshold proxy re-encryption operate the  $kFrag$  and  $cFrag$  redistribution.

---

### Algorithm 3 $kFragRedistr$ for a $(t, n)$ threshold scheme

---

**Require:**  $kFrag_i = (id_i, rk_i, U^{rk_i} \dots), D, sid$

- 1: Get  $id_j$  from each party  $P_j$
  - 2: **for**  $j = 1 \dots n$  **do**
  - 3:   **if**  $j \neq i$  **then**
  - 4:     Compute  $hid_j = H(D, id_j)$
  - 5:   **end if**
  - 6: **end for**
  - 7:  $\{s_{i,j}\}_{j=1}^n \leftarrow \text{LssPrep}(\{hid_j\}_{j=1}^n, rk_i)$
  - 8: **for**  $j = 1 \dots n$  **do**
  - 9:   **if**  $j \neq i$  **then**
  - 10:      $P_i$  sends share  $(id_i, s_{i,j})$  to party  $P_j$
  - 11:   **end if**
  - 12: **end for**
  - 13: Wait for all shares  $\{(id_j, s_{j,i})\}_{j=1}^n$  from parties  $P_j$
  - 14:  $rk' \leftarrow \text{LssRec}(\{(id_j, s_{j,i})\}_{j=1}^n)$
  - 15: Erase  $rk_i$  and  $s_{i,j} \forall j$
  - 16: Output  $kFrag'_i = (id_i, rk'_i, U^{rk'_i} \dots)$
- 

---

### Algorithm 4 $cFragRefresh$ for a $(t, n)$ threshold scheme

---

**Require:**  $cFrag_i, rk_i, rk'_i, sid$

- 1:  $(E_i, V_i, id_i, X_A) \leftarrow cFrag_i$
  - 2:  $E'_i = E_i^{rk'_i/rk_i}$
  - 3:  $V'_i = V_i^{rk'_i/rk_i}$
  - 4: Output  $cFrag'_i = (E'_i, V'_i, id_i, X_A)$
- 

Algorithm 4 shows how a  $cFrag$  is updated. Note that  $cFragRefresh$  must be performed after  $kFragRedistr$ , since knowledge of the new  $rk'$  is necessary to operate the update of the  $cFrag$ . To see why the change works note that:

$$E'_i = E_i^{\frac{rk'_i}{rk_i}} = (E^{rk_i})^{\frac{rk'_i}{rk_i}} = E^{rk'_i} \quad (1)$$

so  $E'_i$  is constructed as if it came directly from the ReEncryption function. Moreover, the change does *not* require the nodes operators to know  $E$  (which would be unfeasible

because of the discrete logarithm problem). All these considerations works similarly for  $V'_i$ .

## IV. A DLT BASED PDS

This section outlines our proposed scheme: a DLT-based Personal Data Storage (PDS). The design of the PDS addresses two key issues: the lack of transparency in managing personal information and the inability to access and make personal data interoperable. Our PDS offers a user-centered model for managing personal data, where storage is separated from the application logic. Providers of personal data apps and data intermediaries can leverage PDS to demonstrate their compliance with regulations such as GDPR [18]. The resulting PDS system is GDPR compliant, providing protection to users' data, and promotes transparent personal data sharing.

Our system architecture relies on the use of DLTs and a decentralized file storage (DFS). DLTs offer technological guarantees for trusted data management and sharing, as they provide a fully auditable decentralized access control policy management and evaluation via smart contracts. This feature enables the actors involved in data processing and control to demonstrate their compliance transparently. The DFS is combined with the DLT to overcome the scalability and privacy issues associated with DLTs while preserving the benefits of decentralization. A DFS is used for storing data outside the DLT through "off-chain" storage, and tracing all data references in the DLT through "on-chain" storage.

The proposed system has four operation: Data Storage, Data Sharing, New AS Addition and AS removal. In the following we describe the operations

### A. Data Storage

We assume each actor has a unique pair of asymmetric keys obtained via KeyGen (see Definition 1) In particular,  $DH$  has key-pair  $(sk_{DH}, pk_{DH})$ ,  $DR$  has key-pair  $(sk_{DR}, pk_{DR})$ , and  $SP_i$  has key-pair  $(sk_{SP_i}, pk_{SP_i})$ .

The data holder  $DH$  encrypts its personal data  $pd$ , obtaining  $epd \in \mathcal{E}$ , using a symmetric key  $K \in \mathcal{K}$  obtained through the function  $\text{Encapsulate}(pk_{DH})$ . As part of the Encapsulate function,  $K$  is placed in a capsule  $\gamma_K$  (See Section II-B). The capsule is sent to the authorization servers AS.

The (personal) data  $epd$  is then stored in a DFS associated with a DS and it is accessible via a P2P network with data replication mechanisms, making it widely available.  $epd$  can be referenced with its hash pointer  $hp_{epd}$ . The pointer is based on the content hash digest, such as IPFS’s content identifiers, or CIDs: this ensures data verifiability, since data may have been altered since it was stored in the PDS and must be audited.

### B. Data Sharing

If a user  $U$  wants to access  $pd$ , it has to ask  $DH$  for authorization. If  $DH$  grants it, then the user becomes a data receiver  $DR$ . In practice that means  $U$  sends its public key  $pk_U$  to  $DH$  as part of the request. By accepting the request,  $DH$  performs  $\text{ReKeyGen}(sk_{DH}, pk_U, n, t)$  obtaining the new key fragments  $kFrag$ .  $DH$  sends the  $kFrag$  to the ASs: each of them performs  $\text{ReEncapsulate}$  on the received  $kFrag$ . The state of  $U$  changes after being granted access, so it becomes a data receiver  $DR$  (its key pair is denoted  $(sk_{DR}, pk_{DR})$  from now on).

The pointer  $hp_{epd}$  is stored in a smart contract implementing personal data access control. Note that different DLTs and/or services can use the same data storage system, facilitating the creation of a PDS for data portability. Therefore, the primary use of the DLT is the execution of smart contracts implementing personal data access control.

To obtain all the  $cFrag$ ,  $DR$  has to prove ownership of  $pk_{DR}$ . To do that,  $DR$  can sign the message  $hp_{epd}$  with  $sk_{DR}$ . The signature can be either posted on a bulletin board monitored by the ASs, or sent to each AS directly, depending on the implementation.

Upon receiving the signature, each AS checks in the ACL in the smart contract if  $DR$  can access  $hp_{epd}$ . If so, each  $AS_i$  sends  $cFrag_i$  to  $DR$ .

Finally, after collecting all the  $cFrag_i$ ,  $DR$  can obtain the symmetric key  $K$  and decrypt  $epd$ .

### C. New AS Addition

Assuming KeRePRE grows in its user base, it is important for the system to scale accordingly. On the one hand the ACL management can not scale: the ACL is managed by a smart contract, therefore scaling that part means dealing with the topic of blockchain scaling, which we discussed in a previous work [18]. On the other hand, it is possible to add new authorization servers to the PDS management. Using terminology from Section III-A, this means that new ASs have to be added to the access structure.

To see how it is possible, assume the current access structure for a PDS is  $(t, n)$ , i.e.  $t$  ASs among  $n$  are needed to perform the  $\text{ReEncapsulate}$  algorithm for  $DR$  so that  $DR$  can obtain  $K$  via the  $\text{DecapsulateFrag}$ s algorithm on the  $\{cFrag\}_{i=1}^t$  (see Section III-A). Then, a new node  $SP_{n+1}$  can be added by performing  $\text{kFragRedistr}$  to create a  $(t, n+1)$  access structure. Specifically, with reference to Algorithm 3, it is possible to derive  $hid_j$  for  $j = 1, \dots, n+1$  and  $\text{LsssPrep}$  can be called for  $\{hid\}_{j=1}^{n+1}$ . Furthermore, it is easy to see how the access

structure can be incremented not just by 1, but for arbitrary  $\nu > 0$  using the same method and in just one iteration create a new access structure of  $(t, n + \nu)$ .

### D. AS Removal

It is easy to imagine that in the course of operations, at least a subset of nodes becomes faulty or are compromised. While ascertain when a  $AS$  has become malicious is outside the scope of the paper, we focus on how to deal with such cases.

As in Section IV-C, assume a current access structure of  $(t, n)$ . We split the explanation into two parts: we first deal with the case where there are still  $m$  honest nodes with  $t \leq m < n$  and then we deal with the case where  $m < t$ .

If there are  $m$  honest nodes,  $t \leq m < n$ , then it is possible to perform  $\text{kFragRedistr}$  involving only those  $m$  honest nodes and excluding the  $n - m$  malicious ones. Since the  $m$  nodes are honest by hypothesis, then we can trust them to perform data deletion, and therefore exclude the  $n - m$  malicious nodes forever, as we explain in Section V-A. This procedure creates a new access structure of  $(t, m)$ .

On the other hand, if  $m < t$ , then the system is highly compromised and it is impossible to have a secure redistribution mechanism at this point, see proof of Theorem 1.

## V. ANALYSIS

### A. Security

One of the innovations of the proposed PDS is the ability to extend a TPRES to facilitate decentralized and encrypted data management with a dynamic access structure. The security analysis is focused on this aspect. Specifically, the security of adding members (Section IV-C) and deleting members (Section IV-D) needs to be demonstrated. To achieve this, a definition for security within the context of a share-redistribution scheme is introduced. It should be noted that an access structure  $(t, n)$  requires at least  $t$  out of  $n$  parties to reconstruct a secret  $s$ .

**Definition 2** (Secure Redistribution). *Let  $\text{Redist}$  be a share redistribution scheme form an access structure  $\Sigma = (t, n)$  to a access structure  $\Sigma' = (k, m)$  for a secret  $s$ , with  $m \geq t$ . Let  $\mathcal{P}$  be the set of parties for  $\Sigma$ ,  $\mathcal{P}'$  the set of parties in  $\Sigma'$  such that  $|\mathcal{P} \cap \mathcal{P}'| \geq t$ . Then  $\text{Redist}$  is secure if after its run parties from the set  $\mathcal{P} \setminus \mathcal{P}'$  are not able to reconstruct the secret  $s$  anymore.*

We are now ready to state:

**Theorem 1.** *In the hypothesis of Definition 2, if  $2t > n$ , then  $\text{kFragRedistr}$  as presented in Algorithm 3 is a secure share redistribution scheme from  $\Sigma = (t, n)$  to  $\Sigma' = (k, m)$  with  $m \geq t$  and  $k \leq m$ .*

*Proof.* The proof strongly follows the work of Desmedt *et al.* [11], since the routine  $\text{kFragRedistr}$  is inspired by it.

First of all, note that a change from an access structure  $\Sigma = (t, n)$  to a access structure  $\Sigma' = (k, m)$  is feasible if and only if there are still at least  $t$  honest parties, otherwise it is impossible to reconstruct the secret  $s$  in the first place.

This is equivalent to ask for a honest majority since  $t > \lfloor \frac{n}{2} \rfloor$ . Furthermore, note that if  $m < t$  then it is not possible to reconstruct the secret, since  $t$  is the least amount of number of parties in  $\mathcal{P}$  needed to reconstruct  $s$  according to  $\Sigma$ . If all the constraints are satisfied, then kFragRedistr is equivalent to the system of Theorem 1 in [11]. Consequently, it is possible to apply Corollary 3 of [11] and conclude that it is sufficient that all the honest parties in  $\mathcal{P}$  erase  $rk_i$  and  $s_{i,j} \forall j$  to guarantee that parties in  $\mathcal{P} \setminus \mathcal{P}'$  can not reconstruct secret  $s$ . Parties are required to do this operation in Line 15 of Algorithm 3.  $\square$

## B. Data Protection

As per Article 32 of the General Data Protection Regulation (GDPR) [1], data controllers, or *authorization servers* (Section III-A, must adopt appropriate security measures to guarantee the confidentiality and integrity of personal data. Encryption is one among the recommended security measures to protect the data from unauthorized access, albeit not a perfect one. In fact, the disclosure of an encryption key is a considerable risk to the confidentiality and integrity of personal data. In such a scenario, data controllers must promptly take actions in compliance with the GDPR (as per Articles 33 and 34) [1].

In this paper, we assert that KeRePRE enhances the efficacy of certain steps that the data controller must perform in such an eventuality. The primary step is implementing corrective measure to prevent future breaches. This may include enhancing its encryption protocols and improving access controls. KeRePRE enables authorization servers to perform key deletion and addition procedures easily, thus allowing them to comply with the implementation of corrective measures.

## VI. RELATED WORKS

### A. Threshold Proxy Re-Encryption in DLTs

One of the first applications of TPRE into a DLT environment can be seen in the proposal by Chen *et al.* in [4]. The authors apply a TPRE to the access permission mechanism of a consortium DLT. On the other hand, Egorov *et al.* in [19] propose a management service in a decentralized network provide in encryption and cryptographic access control. Similar to our proposal, their TPRE reference software is Umbral [5]. Differently from them, we add a key re-distribution mechanism.

More recently, Chen *et al.* in [3] propose an architecture that converges TPRE and DLT consensus algorithm for the creation of a decentralized key management system tailored for the Internet of Things. This system too lacks a key redistribution mechanism. Bai *et al.* [20] propose a GDPR-compliant data storage and sharing framework using blockchain for smart healthcare systems where a PRE network is used to share the encrypted data. Also in their case, there is no use of key redistribution mechanisms. Furthermore, their PRE network solution does not involve a threshold scheme and can lead to single-point-of-failures.

### B. DLT and personal data sharing

Several research works have proposed the use of DLTs for data management in order to create innovative smart services and promote social good applications [21], [22]. Typically, these approaches involve storing data off-chain while utilizing the DLT to provide data access transparency and granular control at the user level. Access control mechanisms that leverage DLTs and smart contracts have been proposed to solve centralization and privacy issues (see for example the proposal by Jemel *et al.* [23]) and to enable secure storage, sharing, and transmission of data. Many researchers have focused on designing data management systems that preserve user control over their data and meet GDPR requirements. For instance, Merlec *et al.* [9] present a GDPR-compliant system where users have control over their personal data collection and transaction history is recorded on the blockchain for data provenance. Similarly, Hawig *et al.* [24] propose a distributed architecture to exchange health data, while Koshina *et al.* [10] leverages smart contracts for consent to enable healthcare data exchange. In both cases, users can keep their medical data in a personal data account hosted on any cloud-based data management service and customize consent preferences using smart contracts. Chang *et al.* present DeepLinQ [25], a multi-blockchain architecture similar to our proposal. It facilitates privacy-preserving data sharing in healthcare by providing granular access control and smart contracts. Finally, Yan *et al.* [26] introduce a Personal Data Store (PDS) that enables users to collect, store, and share their data with third parties using a Secret Sharing scheme. However, this approach is not GDPR compliant since personal data is stored on-chain.

## VII. CONCLUSIONS AND FUTURE WORKS

This paper proposes a DLT-based personal data storage system that utilizes a threshold proxy re-encryption scheme combined with key redistribution

In particular, the use of smart contracts let users define and enforce access policies for personal data: smart contracts can provide a transparent and auditable mechanism for managing access control that is resistant to tampering and unauthorized modifications, while the immutability of DLTs ensures that access policies cannot be altered without the explicit consent of all relevant parties.

While an implementation of the key re-distribution is available online<sup>3</sup>, in the future we aim to complete the implementation of the whole system.

## REFERENCES

- [1] European Parliament. Regulation (eu) 2016/679, 2016.
- [2] Mirko Zichichi, Stefano Ferretti, Gabriele D'Angelo, and Víctor Rodríguez-Doncel. Personal data access control through distributed authorization. In *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, pages 1–4. IEEE, 2020.
- [3] Yingwen Chen, Bowen Hu, Hujie Yu, Zhimin Duan, and Junxin Huang. A threshold proxy re-encryption scheme for secure iot data sharing based on blockchain. *Electronics*, 10(19):2359, 2021.

<sup>3</sup>The implementation can be found at <https://github.com/dinocen/umbral-rs>



- [4] Xi Chen, Yun Liu, Yong Li, and Changlu Lin. Threshold proxy re-encryption and its application in blockchain. In *Cloud Computing and Security: 4th International Conference, ICCCS 2018, Haikou, China, June 8–10, 2018, Revised Selected Papers, Part IV 4*, pages 16–25. Springer, 2018.
- [5] David Nuñez. Umbral: A threshold proxy re-encryption scheme. Technical report, NuCypher Inc., 2018.
- [6] Verichains. [VSA-2022-120] Multichain: Key Extraction Vulnerability in fastMPC’s Secure Multi-Party Client (smpc). <https://drive.google.com/file/d/1zEgQcb4NDjFi8rvTb2cPD5b5Zxif1-5N/view>, 2022.
- [7] Trail of Bits. Disclosing Shamir’s Secret Sharing vulnerabilities and announcing ZKDocs. <https://blog.trailofbits.com/2021/12/21/disclosing-shamirs-secret-sharing-vulnerabilities-and-announcing-zkdocs/>, 2021.
- [8] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. A survey of ECDSA threshold signing. *IACR Cryptol. ePrint Arch.*, page 1390, 2020.
- [9] Mpyana Mwamba Merlec, Youn Kyu Lee, Seng-Phil Hong, and Hoh Peter In. A smart contract-based dynamic consent management system for personal data usage under gdpr. *Sensors*, 21(23):7994, 2021.
- [10] Mirko Koscina, David Manset, Claudia Negri, and Octavio Perez. Enabling trust in healthcare data exchange with a federated blockchain-based architecture. In *IEEE/WIC/ACM International Conference on Web Intelligence-Companion Volume*, pages 231–237, 2019.
- [11] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. Technical Report ISSE TR-97-01, George Mason University, 1997.
- [12] Kresimir Popovic and Zeljko Hocenski. Cloud computing security issues and challenges. *The 33rd International Convention MIPRO*, pages 344–349, 2010.
- [13] Kui Ren, Cong Wang, and Qian Wang. Security challenges for the public cloud. *IEEE Internet Computing*, 16(1):69–73, 2012.
- [14] Smitha Sundareswaran, Anna Squicciarini, and Dan Lin. Ensuring distributed accountability for data sharing in the cloud. *IEEE Transactions on Dependable and Secure Computing*, 9(4):556–568, 2012.
- [15] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT ’98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 1998.
- [16] Ventzislav Nikov and Svetla Nikova. On proactive secret sharing schemes. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 308–325. Springer, 2004.
- [17] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO ’95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer, 1995.
- [18] Mirko Zichichi, Stefano Ferretti, Gabriele D’Angelo, and Víctor Rodríguez-Doncel. Data governance through a multi-dlt architecture in view of the gdpr. *Cluster Computing*, 25(6):4515 – 4542, 2022.
- [19] Michael Egorov, MacLane Wilkison, and David Nuñez. Nucypher kms: Decentralized key management system. *arXiv preprint arXiv:1707.06140*, 2017.
- [20] Pinky Bai, Sushil Kumar, Kirshna Kumar, Omprakash Kaiwartya, Mufti Mahmud, and Jaime Lloret. Gdpr compliant data storage and sharing in smart healthcare system: a blockchain-based solution. *Electronics*, 11(20):3311, 2022.
- [21] Muqaddas Naz, Fahad A Al-zahrani, Rabiya Khalid, Nadeem Javaid, Ali Mustafa Qamar, Muhammad Khalil Afzal, and Muhammad Shafiq. A secure data sharing platform using blockchain and interplanetary file system. *Sustainability*, 11(24):7054, 2019.
- [22] Victor Ortega, Faiza Bouchmal, and Jose F Monserrat. Trusted 5G vehicular networks: Blockchains and content-centric networking. *IEEE Vehicular Technology Magazine*, 13(2), 2018.
- [23] Mayssa Jemel and Ahmed Serhrouchni. Decentralized access control mechanism with temporal dimension based on blockchain. In *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*, pages 177–182. IEEE, 2017.
- [24] David Hawig, Chao Zhou, Sebastian Fuhrhop, Andre S Fialho, and Navin Ramachandran. Designing a distributed ledger technology system for interoperable and general data protection regulation-compliant health data exchange: a use case in blood glucose data. *Journal of medical Internet research*, 21(6):e13665, 2019.
- [25] Edward Y Chang, Shih-Wei Liao, Chun-Ting Liu, Wei-Chen Lin, Pin-Wei Liao, Wei-Kang Fu, Chung-Huan Mei, and Emily J Chang. Deeping: distributed multi-layer ledgers for privacy-preserving data sharing. In *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 173–178. IEEE, 2018.
- [26] Zhu Yan, Guhua Gan, and Khaled Riad. Bc-pds: protecting privacy and self-sovereignty through blockchains for openpds. In *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 138–144. IEEE, 2017.