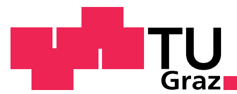


Nieddu Franco, BSc

Key Management and Key-Loss Recovery for Cloud Data Storage Systems using Multi-Use Proxy Re-Encryption

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

O.Univ.-Prof. Dipl-Ing. Dr.techn. Posh Reinhard

Institute of Applied Information Processing and Communications
Head: O.Univ.-Prof. Dipl-Ing. Dr.techn. Posh Reinhard

Graz, August 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Abstract

Nowadays, cloud-based data storages are heavily used in a vast number of fields. This includes, among others, storing business-related data, medical records, or pictures of friends and family. Of course, the confidentiality, integrity, and availability of the data have to be ensured. Services use dedicated safeguards to provide these security aspects. One possible solution is to use cryptography and store the key material on the client's devices. Nevertheless, there is an obvious downside with this approach, namely the complete loss of access to the documents if all of the user's devices are lost.

This thesis proposes a novel key management system that solves said downside utilizing Multi-Use Proxy Re-Encryption (MUPRE) while still keeping the key material unextractable if it was created within the secure hardware of the client's device. The key management approach is embedded in a data sharing system that enables users to register multiple devices. Every device creates an own MUPRE key pair, stores potentially sensitive data, synchronizes these data on all devices, and shares the data with distinct users. The underlying cryptosystem is a hybrid approach, where the asymmetric part is a MUPRE scheme. For the symmetric part, we use AES. Furthermore, we evaluate multiple MUPRE schemes concerning specific properties that are relevant for our use case. Additionally, we discuss how to translate a scheme that relies on type 1 bilinear pairings into a type 3 pairing scheme.

As a result, we solved a common problem that cloud-based systems face, which rely on client-based keys, namely recovery from device-loss. We showcased how Multi-Use Proxy Re-Encryption can be used to share data between users in a secure and user-friendly manner.

Keywords— cloud-based, data storage, key management, Multi-Use Proxy Re-Encryption

Kurzfassung

Heutzutage werden cloud-basierte Datenspeicher in vielen verschiedenen Gebieten verwendet. Diese Gebiete beinhalten unter anderem das Speichern von geschäftsbezogenen Daten, medizinischen Daten oder Fotos von Freunden und Familie. Selbstverständlich müssen diese Datenspeicher die Vertraulichkeit, Integrität und Erreichbarkeit der Daten gewährleisten. Dienste verwenden eine Vielzahl an Mechanismen, um diese Sicherheitsaspekte anzubieten. Kryptografie ist eine mögliche Lösung für dieses Problem, wobei die verwendeten Schlüssel am Endgerät der BenutzerInnen erstellt und verwaltet werden. Bei diesem Ansatz führt der Verlust aller Endgeräte von BenutzerInnen zu einem offensichtlichen Nachteil, nämlich zu einem kompletten Zugangsverlust von allen Dokumenten.

Diese Masterarbeit stellt eine neuartige Schlüsselverwaltung vor, welche die besagten Nachteile von cloud-basierten Datenspeichern löst. Dies wird durch die Verwendung von Multi-Use Proxy Re-Encryption (MUPRE) ermöglicht, wobei Schlüssel, die in der sicheren Hardware eines Endgeräts erstellt wurden, weiterhin nicht extrahierbar bleiben. Die Schlüsselverwaltung ist in einem System zur Datenspeicherung eingebettet, welches BenutzerInnen ermöglicht, mehrere Endgeräte zu registrieren, wobei jedes Endgerät ein eigenes MUPRE Schlüsselpaar erzeugt. Des Weiteren ist es möglich, potenziell sensible Daten zu speichern, besagte Daten über alle Endgeräte zu synchronisieren und die Daten mit Dritten zu teilen. Das verwendete hybride Kryptosystem nutzt ein MUPRE Schema und AES. Des Weiteren evaluieren wir mehrere MUPRE Schemen mit speziellem Augenmerk auf die Anforderungen für unsere Schlüsselverwaltung. Zusätzlich diskutieren wir, wie es möglich ist, ein Schema, welches bilineare Pairings des Typs 1 verwendet, in ein Schema zu übersetzen, welches Typ 3 Pairings verwendet.

Zusammenfassend wird mit unserer Arbeit ein Problem von cloud-basierten Systemen gelöst, die kryptografische Schlüssel auf den Endgeräten ihrer NutzerInnen speichern. Mit unserer Lösung ist es möglich, nach dem Verlust aller Endgeräte den Zugriff auf verschlüsselte Daten wieder zu erlangen. Außerdem zeigen wir, wie es möglich ist, mit Multi-Use Proxy Re-Encryption ein System zu erstellen, welches NutzerInnen ermöglicht, Daten mit Dritten in sicherer und nutzerInnenfreundlicher Art zu teilen.

***Keywords*—** cloud-basiert, Datenspeicher, Schlüsselverwaltung, Multi-Use Proxy Re-Encryption

Acknowledgements

I want to take the opportunity to thank Felix Hörandner, who supported me during the last two years. Without his support and assistance, it would not have been possible for me to complete this thesis.

Furthermore, I want to thank Harald Bratko, who sparked my interest in cyber-security during my internship at the IAIK five years ago and is now a colleague within my working group.

Last but not least, a special thanks to all my friends and family that supported me during the previous years, primarily to Fabian, Tina, and Daniel. I would not be here without you.

Contents

Abstract iii

Acknowledgements v

1 Introduction 3
1.1 Contribution 4
1.2 Outline 5

2 Related Work 6
2.1 Secure Data Sharing Cryptography 6
2.2 Recovery from Key-Loss Techniques 8
2.3 Complete Deployed Systems 10
2.3.1 Tresorit 11
2.3.2 pCloud 13
2.3.3 SpiderOak 14

3 Background 17
3.1 Pairing-Based Cryptography 17
3.2 Proxy Re-Encryption 19
3.3 Multi-Use Proxy Re-Encryption 19
3.4 Hardware-Backed Keys 21

4 Multi-User Data Sharing System and Recovery from Device-Loss Strategies 24
4.1 Challenges 24
4.2 Data Sharing 25
4.3 Recovery from Device-Loss 29
4.4 Trusted Recovery Users 35
4.5 Key-Authenticity 38
4.6 Trust Assumptions 43

5 Evaluation of PRE Schemes 44
5.1 Properties of Proxy Re-Encryption Schemes 44
5.2 Evaluation Process 48
5.2.1 Initial MUPRE Schemes 48
5.2.2 PRE based on Bilinear Pairings 49
5.2.3 PRE based on Lattices 51
5.3 Chosen scheme 52

6 Translation Process 55
6.1 Trivial Approach 55
6.2 Advanced Approach 58
6.3 Translated Scheme 60

7 Implementation of Our System 64
7.1 Actors and Components 64
7.1.1 MUPRE Library 64
7.1.2 Android Application 66
7.1.3 Cloud Service 67
7.1.4 MUPRE Certificate Authority 69
7.2 Process 70
7.2.1 Registration of a Primary Device (PD) 70
7.2.2 Registration of an Secondary Device (SD) 71
7.2.3 Handling of Documents 72
7.2.4 Sharing of Documents 74
7.2.5 Recovery after Device-Loss 76
7.3 Discussion 79
7.3.1 Performance Evaluation 79
7.3.2 Discussion of Challenges 83

8 Conclusion 85

References 86

List of Figures

1 Data flow of a Proxy Re-Encryption (PRE) scheme 20
2 The separation of the Trusted Execution Environment (TEE) and Rich Execution Environment (REE) 22
3 Complete operation flow of the system 27
4 Messages sent to recovery from SD loss 31
5 Messages sent to recover from PD loss when there is no SD registered 34
6 Messages sent to recover with a threshold of trusted users 37
7 Flat hierarchy of certified public keys 39
8 The hierarchical structure of certified public keys 41
9 Certification of Alice's recovery device with her trusted recovery user Bob 42
10 Basic dependency graph for the translation process 58
11 Labeled dependency graph for the translation process 59
12 Cyclic dependency graph for the translation process 60
13 Components of our system 65
14 Database definition of the cloud service 68
15 Messages sent for certification of the MUPRE public key 69
16 Registration Process 71
a Home screen 71
b Specification for username and password 71
c Additional information for the certificate 71
17 The welcome screen of the application after registration 72
18 User Interface for the registration of an SD 73
a Push notification on the PD 73
b Input for password to finish registration 73
19 Document handling 75
a All documents for which the device has access 75
b Prompt to upload a text document 75
c A downloaded, decrypted document 75
20 User Interface for sharing of documents 77
a All pairings from this device 77
b Prompt to send a pairing request to another user 77
c The push notification on the device of the other user 77
d Input for the distinct user to enter password 77
21 User Interface for recovery 78
a Input for username, password, new alias, and trusted user . 78

b The push notification on the trusted user's phone 78
c The input where the trusted user has to input their password 78

Abbrevations

ECC	Elliptic Curve Cryptography
PRE	Proxy Re-Encryption
MUPRE	Multi-Use Proxy Re-Encryption
DHP	Diffie-Hellman Problem
DLP	Discrete Logarithm Problem
BDHP	Bilinear Diffie-Hellman Problem
CPA	Chosen-Plaintext Attack
CCA	Chosen-Ciphertext Attack
CCA1	Non-Adaptive Chosen-Ciphertext Attack
CCA2	Adaptive Chosen-Ciphertext Attack
LWE	Learning-With-Errors
CDH	Computational Diffie-Hellman
IDBC	ID-based cryptography
PD	Primary Device
SD	Secondary Device
TEE	Trusted Execution Environment
SGX	Software Guard Extensions
REE	Rich Execution Environment
PKI	Public Key Infrastructure
CA	Certificate Authority
ECDSA	Elliptic Curve Digital Signature Algorithm
AES	Advanced Encryption Standard
GCM	Galois/Counter Mode
CFB	Cipher Feedback Mode

IV	Initialization Vector
SHA-256	Secure Hash Algorithm 256
SHA-512	Secure Hash Algorithm 512
RSA	Rivest–Shamir–Adleman
TLS	Transport Layer Security
ABE	Attribute-Based Encryption
IBE	Identity-Based Encryption
FE	Functional Encryption
MPC	Multi-Party Computation
HMAC	Hash-Based Message Authentication Code
TGDH	Tree-Based Group Diffie-Hellman
PBKDF2	Password-Based Key Derivation Function 2

1 Introduction

Nowadays, a vast amount of services utilize cloud storages due to their usefulness. Such storages enable to persistently store data in a way such that users are not burdened with, among others, the inconvenience of availability and backing-up their data. Due to these systems' inherent design, availability is no longer that much of an issue as users can register multiple devices and synchronize their data on each device. Furthermore, it is also possible to share documents conveniently with distinct users. In contrast to conventional ways of storing data on a physical hard drive, users also outsource backing-up their data, as the cloud storage stores data redundantly. However, storing sensitive data in cleartext places high trust requirements on the cloud storage. By incorporating cryptographic primitives, it is also possible to protect data integrity and confidentiality, enabling to store sensitive data.

When using cryptography, data availability and sharing of data fundamentally rely on managing and distributing keys on the system's users' devices. Conventionally, if users lose their devices and, therefore, their key material, this could lead to specific problems and, in the worst-case, even complete data loss. For that reason, cloud services rely on dedicated strategies to recover from device-loss. If users register multiple devices, access to their data is still intact after losing a single device. However, there may be users who registered only a single device. cloud services have to offer data-access recovery strategies also for these users.

cloud services use various approaches to recover from device-loss with different advantages and disadvantages. The simplest way to tackle the problem is to redundantly store the key material, e.g., on a flash drive or physically print the key on paper. An obvious disadvantage is that users have to have a secure location where they put the key and remember this location after a possibly long time when recovery becomes necessary. More sophisticated approaches may use password-protection or biometric access control, as proposed by Kaliski, 2000 and Dodis et al., 2004, respectively, and store the key in the cloud. Unfortunately, these kinds of key-protection suffer from fundamental problems, such as low entropy, which is vulnerable to brute-force attacks. It is possible to use, e.g., secret sharing mechanisms, originally proposed by Shamir, 1979, which enables us to split a secret and distribute the created parts to distinct semi-trusted parties to counteract the problem of limited entropy. On-demand the secret can then be reconstructed if a well-defined amount of trusted entities are available.

All those approaches have in common, that the key has to be extractable from the device that created it. This leads to an apparent vulnerability if a user's device got stolen or is lost. Nowadays, there are ways to create key material securely with the help of dedicated hardware incorporated in devices, like Intel's Software Guard Extensions or the ARM's Trusted Execution Environment. With these technologies, keys are bound to the device and are not extractable.

This thesis proposes a cloud-based data sharing solution that supports multiple devices per user, where the key material of every device is backed by secure-hardware. Alongside the sharing and storage capabilities, we propose a key management system that enables device-loss recovery strategies, even for users with a single device. We utilize a cryptographic primitive called Multi-Use Proxy Re-Encryption (MUPRE), originally proposed by Blaze et al., 1998, to implement the sharing aspect of our system and the key management.

1.1 Contribution

This thesis offers four main contributions, which serve as a basis for our paper (Hörandner and Nieddu, 2019):

Cloud-Based Data Sharing System using Multi-Use Proxy Re-Encryption We propose a cloud-based data storage system that relies on MUPRE to share data with distinct users. Furthermore, it is possible for users to register multiple devices at the service and to synchronize all their documents in a user-friendly manner. Also, for the synchronization between the user's devices MUPRE is used.

Key Management System for Recovery from Device-Loss This thesis defines a key management system that allows users to securely create and store their key material inside their devices' secure hardware. Furthermore, it enables users to recover access to their data if they lose all devices registered at the service where they can make their own trust decisions.

Evaluation of Proxy Re-Encryption Schemes Additionally, this thesis evaluates multiple PRE schemes. We give an overview of specific properties of PRE schemes, define which requirements have to be met for our data sharing solution

and key management system, and finally decide on a scheme which then can be used for our proof of concept implementation.

Implementation to Showcase Feasibility and Evaluate Performance We state how we created the proof of concept implementation of our system. We integrated all proposed parts of the system, including the ability to share data with distinct users, the key management system, and the MUPRE scheme. We further evaluate and discuss the performance of the system and especially the MUPRE scheme.

1.2 Outline

The first Chapter of this thesis aims to give an introduction to the thesis. Chapter 2 discusses work related to our proposed system. Chapter 3 presents the background knowledge necessary to understand the remaining thesis. Next, Chapter 4 offers the cloud-based data sharing system, alongside the recovery strategies from device-loss. Then, Chapter 5 evaluates multiple Multi-Use Proxy Re-Encryption schemes against requirements identified for our system. Chapter 6 explains how we transform the chosen scheme, so it is possible to implement it practically. Afterward, Chapter 7 elaborates on our proof of concept implementation and states the challenges we faced during implementation. Finally, Chapter 8 concludes the thesis.

2 Related Work

In this section, we discuss other technologies that are related to our proposed key management system. In the first part, we will focus on cryptographic primitives that may be used for data sharing in the cloud setting and how these building blocks differ from our used cryptosystem. The second part will evaluate different solutions for key-loss recovery. The section then concludes with a discussion of three real-world cloud-based data storages, how they function, and their contrast to our proposed system.

2.1 Secure Data Sharing Cryptography

Over the next paragraphs, we will state different cryptographic building blocks that also allow secure data sharing in the context of cloud-based data storages. We will further state the difference to our system.

Hybrid Encryption and Lockbox Constructions One of the critical aspects of every cloud-based data storage is the end-to-end confidentiality that has to be ensured. There are multiple ways to achieve this, one being hybrid encryption. Hybrid encryption combines the advantages of symmetric and asymmetric encryption by encrypting the plaintext with a symmetric cipher. It then encrypts the key of the symmetric cipher asymmetrically.

In their work, Fu, 1999 expand on the idea of hybrid encryption by proposing "lockbox" constructions - the term lockbox refers to a key that is encrypted with another key. They give a real-world example to understand their concept better. Imagine putting a key to a house inside a box. Additionally, a lock is attached to the box that contains the key. Now, everyone who knows the lock's combination can obtain the key for the house and enter it. In summary, a lockbox allows users to store a key in public, and everyone with correct access rights can obtain the key.

On the one hand, data sharing is easily possible with lockbox constructions and also with hybrid encryption. Our system also relies on hybrid encryption, where a MUPRE scheme is used for the asymmetric part. On the other hand, the recovery strategies from device-loss defined in 4.3 cannot be implemented with the same security guarantees and trust assumptions using lockboxes or conventional hybrid encryption.

Attribute-Based Encryption (ABE) Proposed by Goyal et al., 2006, ABE also enables users to share data. In contrast to conventional asymmetric encryption, where it is necessary to share the private keys to decrypt existing ciphertexts, ABE allows for ciphertexts to be decrypted by multiple decryption keys as long as the keys are associated with the correct access rights. This is done by labeling the ciphertexts with certain attributes and decryption keys with the corresponding structures that control whether the key can decrypt the ciphertext. The decryption keys are derived from a master secret key and are generated by a trusted third party.

Unfortunately, ABE assumes high trust assumptions, as the trusted third party that creates the decryption keys can decrypt any ciphertext. Regarding the trust assumptions for our proposed system, read Section 4.6.

IBE Identity-Based Encryption (IBE) was proposed initially by Shamir, 1984, but remained an open problem for several years, as in his work, Shamir only gave an identity-based signature scheme. Some years later, Boneh and Franklin, 2001 gave an IBE scheme based on the Weil pairing.

The main advantage for IBE is that keys can be generated from publicly known values bound to an identity, such as an email address. The corresponding private key is then generated from a trusted third party, similar to ABE, with the help of a master key. This procedure inherently allows the key-generating party to create private keys for every known identity and, in turn, enables it to decrypt all ciphertexts, which is a disadvantage when comparing it with our proposed system.

Functional Encryption (FE) Initially formalized in Boneh et al., 2011, FE is a cryptographic primitive that allows a key holder to learn a function of an encrypted value, but nothing about the value itself.

Definition 1. *A FE scheme for a specific functionality f, defined over (K, X) consists of the following operations, which satisfy the correctness conditions for all $k \in K$ and $x \in X$:*

Setup(1^κ) $\rightarrow$ (pp, mk): *This operation generates a public and a master secret key pair* (pp, mk)*, depending on the input parameter κ.*

KeyGen(mk, k) $\rightarrow$ sk: *On input of a master secret key* mk *and some* k *, the KeyGen operation creates a secret key* sk *for* k.

Enc(pp, x) $\rightarrow$ c: *The input to this operation is a public key* pk *and some message* x. *The operations outputs a ciphertext* c *of the message* x

Dec(sk, c) $\rightarrow$ y: *The decrypt operations uses the secret key* sk *to compute* $f(\mathrm{k}, \mathrm{x})$ *from* c

FE received a lot of attention during the last years, with remarkable progress and has many theoretical fields of application. One of the interesting use cases is that it is able to generalize existing cryptographic primitives like ABE as well as IBE. For ABE, imagine defining the functionality f like:

$$f(k,x) = \begin{cases} x & \text{if } k \text{ has attributes attached that allows to decrypt } x \\ \perp & \text{otherwise} \end{cases}$$

Furthermore, for the IBE case, define f as follows:

$$f(k,x) = \begin{cases} x & \text{if } k \text{ is an identity that allows to decrypt } x \\ \perp & \text{otherwise} \end{cases}$$

Because of the possibility to formalize ABE and IBE through FE, FE enables data sharing in the cloud setting as well. Unfortunately, also for FE, there is a third party involved that may get access to all existing ciphertexts and has therefore much higher trust assumptions than our proposed system.

All three described cryptographic building blocks suffer from the same inherent problem. There are ways to limit the required trust, for example, by using Multi-Party Computation (MPC) to create the corresponding private keys. MPC allows us to split the trust, such that the possibility to read any ciphertext is not centralized. In this setting, a certain threshold of parties has to be corrupted to break the system. Nevertheless, these approaches also have to be paired with a strategy to recover from key-loss.

2.2 Recovery from Key-Loss Techniques

This section discusses different ways to recover from key-loss and how these techniques differ from our approach. The focus lies on cloud-based systems that rely on storing cryptographic keys on the client's device, as also our system is built with this approach.

Password-Based Encryption After creating a key on the user's device, the key is further encrypted and uploaded to a cloud storage. The encrypting key is derived from a password given by the user with a key derivation function. Multiple schemes are specifically designed for this exact purpose. If a user then loses their device, the encrypted key can be downloaded to another device. If the user can reenter the password, the key is decrypted, and in turn, the access to the documents stored in the cloud is restored.

There are multiple downsides with this approach. First of all, relying on passwords provided by users can lead to problems if the entropy of the password is not high enough and is therefore guessable by attackers. Furthermore, enforcing high-entropy enforces users to either remember long passwords or persistently store the password somewhere, e.g., as a paper-key or on the hard drive of a device. Loss of the high-entropy password renders the recovery impossible.

Biometric Encryption Like password-based encryption, biometric encryption uses key derivation to encrypt the key created on the user's device, which can then be uploaded to a cloud storage. In contrast to password-based encryption, where some information the user knows, e.g., a password, is used to derive the key, biometric encryption uses the user's biometric templates, like in the most common case fingerprints. There are multiple ways to derive cryptographic keys from such templates, which can be seen, for example, in the work from Jin et al., 2004 or Dodis et al., 2004. In their work, Jin et al., 2004 discuss the advantages of biohashing, whereas Dodis et al., 2004 propose to use fuzzy extractors to derive keys from noisy data.

Unfortunately, as seen in Nagar et al., 2010, biometric encryption has certain disadvantages. The first apparent downside is that the biometric template used to create the key must be kept confidential. Users can easily create a fresh template when using password-based encryption schemes if the template (the password) leaked. In contrast, users cannot switch the template when using biometric encryption as it is their biometric identifier. Systems may enforce that additional data is used to generate the keys to counteract this problem. This additional data has to be kept confidential and available, so biometric encryption faces the same problems as password-based encryption.

Secret Sharing The concept of secret sharing was proposed by Shamir, 1979. It solves multiple problems regarding key management in a cloud setting where

potential corrupted parties are involved. To share a secret, the secret is divided into n shares. This secret can then be reconstructed if one obtains k of these shares, where k is some previously defined threshold. In contrast, no information about the secret is leaked even if someone is in possession of $k - 1$ shares.

In the context of cloud-based data storages, users can generate a recovery key, then use secret sharing to split the key, and finally transmit the n shares to partially trusted third parties, as seen in Huang et al., 2011. These partially trusted third parties may be the user, the cloud service, or someone of the user's social network. The user has to convince k parties to transmit their corresponding shares to regenerate the recovery key.

However, neither the original proposal by Shamir, 1979 nor the work from Huang et al., 2011 addresses the problem of authenticating requests such that only authorized parties receive shares.

Password-Protected Secret Sharing There are multiple ways to incorporate authentication mechanisms into a secret sharing protocol. For example introducing password-based authentication, such that entities holding shares, can verify that a request is coming from an authorized source, for example, proposed by Bagherzandi et al., 2011. In their work, the authors present a secret sharing protocol, where a user on recovery has to authenticate themself with a password to the parties that hold the shares. The authentication is done with a zero-knowledge protocol, implying that the shareholders do not learn the user's password.

This approach solves the inherent problem of other secret sharing protocols, namely the user's lack of authentication methods. Nevertheless, secret sharing protocols, like proposed by Shamir, 1979, Huang et al., 2011, or Bagherzandi et al., 2011, require that the secret key is extractable from the device. Our proposed key management system is extensible with secret sharing, as seen in Section 4.4. The main advantage is that our system leaves the unextractability of the secret key intact, meaning it still possible to utilize hardware protection for keys.

2.3 Complete Deployed Systems

Of course, various systems offer similar services as our proposed data storage, which are deployed in the real-world. This section will discuss three cloud-based data storages that offer high security and compares their solution with ours.

2.3.1 Tresorit

By enabling secure storage of documents, synchronizing files, and sharing documents, **Tresorit** offers the same functionality as our proposed system. Their core security guarantee is the end-to-end encryption that is warranted during every action a user performs. In contrast to other cloud storages, they incorporate client-side encryption, implying that no document in plaintext is uploaded to their servers.

There is official client software for multiple operating systems, including macOS, Windows, and a Linux beta version. The end-to-end encryption used in the ecosystem of Tresorit is described in TRESORIT, 2014. The client-side encryption uses Advanced Encryption Standard (AES) in Cipher Feedback Mode (CFB) with a fresh 256-bit key that the client software chooses. To further ensure the integrity of documents, Tresorit computes a Hash-Based Message Authentication Code (HMAC), usually with Secure Hash Algorithm 512 (SHA-512).

If a user wants to upload a document, the encrypted document is added to the cloud's existing directory structure. These structures represent the client-side directories and consist of other directories and files called "Tresor". The directories hold the symmetric keys of the files and other directories they contain, resulting in a hierarchy of symmetric keys. It is only possible to add documents to the Tresor if one knows the root key. The so-called Agreement Module provides the root key. There are two different types of Agreement Module, namely one based on Rivest–Shamir–Adleman (RSA) and one on Tree-Based Group Diffie-Hellman (TGDH).

The RSA-based Agreement Module contains multiple pre-master secrets. If a user wants to share their Tresor with another distinct user, a new pre-master secret is added to the Agreement Module, which is encrypted with the distinct user's RSA public key. If a user wants to access the shared Tresor, they have to provide their private key to the Agreement Module. The Agreement Module decrypts the corresponding pre-master secret and then computes the root key of the Tresor. In contrast to the RSA-based module, the TGDH-based Agreement Module does not store the pre-master secrets, but the users' certificates.

To share the access of a Tresor with other users, Tresorit relies on a 3-step authentication process done via email. In the first step, users authenticate themself with an X.509 certificate and the corresponding RSA-2048 private key. After that authentication, an invitation is sent to the user with a 256-bit random secret. Finally, the two users perform a handshake to establish the keys during a challenge-response

protocol where the invitee presents the random secret received via email during the protocol.

Tresorit also accounts for the case that a user lost all devices or forgot their login password. If either a user has access to a previously paired client device but forgot the password, or did loose all registered devices but still remembers the password, the access to the user's Tresors can be restored. If the user lost all devices and also forgot their password, the access cannot be restored.

On registration, a random 256-bit master salt value is generated for the user. This master salt and the user's chosen password is used to derive a 256-bit master key with Password-Based Key Derivation Function 2 (PBKDF2). The master key is then used to encrypt the so-called "Roaming Profile", which contains the user's Transport Layer Security (TLS) certificate and the associated private key, and the certificates and keys used to get access to the user's Tresors, including the keys relevant for the Agreement Module. The encrypted Roaming Profile is uploaded to the cloud and also stored on the user's device. Furthermore, an HMAC using SHA-512 is computed over the Roaming Profile for integrity protection. Additionally to the master key, a 160-bit authentication salt value is computed, which is then used, together with the user's password, to derive an authentication key, again, using PBKDF2. This authentication key is also sent to the server.

If a user wants to log in to the service with a registered device, the 256-bit master key is regenerated on the client's side with the stored salt value and the user's password. The master key can be used to decrypt the locally saved Roaming Profile such that all necessary keys are available to the client software. In case a new device tries to log in, e.g., when a user reinstalled their operating system or lost all other devices, the default login process is not possible because the encrypted Roaming Profile is not stored on the new device. The user has to perform a proprietary challenge-response protocol based on the authentication key generated during the registration process to authenticate at the Tresorit servers. On success, the server sends the encrypted Roaming Profile to the new client, where the default login process can now be invoked.

To sum things up, Tresorit uses a hybrid encryption approach to store the user's documents securely. It uses AES in CFB mode and RSA to encrypt the corresponding AES secret keys. This approach is similar to our solution regarding the persistent storage of data. We also embed hybrid encryption, but we use MUPRE to encrypt the AES keys. There is one advantage regarding the storage of the cryptographic keys and their usage in our key management design, namely we can store the private key of the asymmetric primitive in the secure hardware of

the client's device, which is not possible for Tresorit, as their design enforces the extractability of the RSA key. For users to authenticate themselves to another user, both systems rely on some external channel and Public Key Infrastructure (PKI).

As already mentioned, Tresorit's design does not allow for the usage of hardware-backed keys. Another difference between Tresorit and our system is how we handle the recovery from device-loss. Whereas Tresorit solely relies on password-based authentication, our system has another form of authentication. We recommend using the authentication mechanism described in Section 4.4, where the user in recovery has to convince a threshold of semi-trusted users to reconstruct at least one MUPRE re-key which then can be used to restore the user's access. Not only are all involved asymmetric keys inside the secure hardware of the corresponding devices, but the protection from misusing the recovery mechanism is more robust with our approach.

2.3.2 pCloud

pCloud is a service provider that also offers, among others, a secure cloud storage. In the default case without any additional security packages, pCloud uses server-side encryption with 256-bit AES. Furthermore, during transfer, they rely on a channel secured by TLS. They enable device synchronization and offer a web application, a desktop application, and multiple mobile applications.

As an additional security feature, they offer an encryption package that is extra charged. This package is called pCloud Crypto and is advertised as an additional layer of security. In contrast to the general package, it enables client-side encryption with 256-bit AES. Every file and folder is encrypted with a fresh key, whereas these keys are encrypted with 4096-bit RSA. In contrast to their competition, pCloud Crypto offers the possibility to decide if data should be encrypted on the client-side or not. They argue that it is beneficial if some data may be stored with server-side encryption. For example, they give server-side rendered thumbnails for pictures or transcoding of media files such that they are playable in the cloud, as this would be impossible with client-side encryption.

This duality of server-side and client-side encryption is enabled by providing the opportunity to "lock" individual folders. If a user locks a file, it is encrypted with an AES key, which is derived from the so-called Crypto Pass. The Crypto Pass is a high-entropy password. If a user forgets their Crypto Pass while using pCloud Crypto, the encrypted data is lost. Furthermore, file sharing with users is not

possible with pCloud Crypto. Nevertheless, pCloud offers another package called pCloud Business. It may be possible to retrieve access to encrypted data if the Crypto Pass is lost and enables the feature of sharing data with distinct users.

pCloud Business is a separate package of pCloud. The package targets businesses that want to store their data in a common place and collaboratively work on it. As the owner of a pCloud Business account, it is also possible to add pCloud Crypto to the shared folders. The owner creates an encrypted folder and can share subfolders with distinct users. This process implies that the owner of the account has complete access to all the stored data. For the sub-folders, it is necessary to create a temporary Crypto Pass that the other users need to access the encrypted folders. If the business package owner forgets the Crypto Pass, it is implied that it may be possible to recover the access to the encrypted folder with the help of the support of pCloud. However, due to the project's closed-source nature, it is not stated how this process works. Furthermore, if the support can restore access to encrypted documents, end-to-end encryption is not guaranteed.

In contrast to our system, the sharing capabilities of pCloud are only possible if a user relies on server-side encryption or for the pCloud Business users. Although pCloud Business users can share data with distinct users, the package targets explicitly businesses and is designed so that the owner of the account has complete access to all the stored data. Our system enables data sharing with distinct users, even with client-side encryption.

Data recovery for pCloud users without using any additional packages is easily made, as this service relies on server-side encryption. The authentication mechanism relies on simple password authentication, although it is possible to enable two-factor authentication. If using client-side encryption with pCloud Crypto, the loss of the Crypto Pass implies complete data loss. As already mentioned, it is hinted that for pCloud Business users, there may be a way for the support of pCloud to recover access to documents, but it is not stated how this is possible. Our system offers a user-friendly way of data recovery with minimal trust decisions of the user, even with client-side encryption.

2.3.3 SpiderOak

Originally released in December 2007, **SpiderOak** offers a cloud-based backup system, along with file-sharing capabilities called "SpiderOak One". SpiderOak One uses a two-layered encryption approach. Every file is encrypted with a fresh

AES key. These encryption keys are the "outer layer" of their approach, and this outer layer is encrypted with AES in CFB mode of operation. Furthermore, a HMAC with Secure Hash Algorithm 256 (SHA-256) is computed to protect the integrity of the keys. The key used for encrypting the outer layer is derived from a key derivation function, namely PBKDF2 with SHA-256, 16384 rounds, and a 32 bytes random salt value. They guarantee that, if using their client, no keys, files, or even metadata of files are stored in plaintext on the server. They also use TLS secured channels for communication with their servers.

SpiderOak One provides separate clients for Windows, macOS, and Android. They also offer the possibility to log in via the web, although all client-side security guarantees only hold if the user is logged in with the client software. When using the web platform, the password is transmitted to the server. Therefore SpiderOak recommends using one of the clients. The synchronization between different devices is enabled by password-based encryption. If a user provides the password used for encrypting their outer layer, it is possible to decrypt the data on any device.

For multi-user sharing, SpiderOak One offers the possibility to create so-called "ShareRooms". A user can specify folders or files which should be added to a ShareRoom. Suppose another person (not necessarily another user of SpiderOak One) has access to the link. In that case, the person can read the room's data, as long as the room persists, although the remaining data of the user remains confidential. This can be done with the two-layered approach of SpiderOak, as every file is encrypted with a separate key.

Additionally, there is a more sophisticated solution for multi-user sharing, called "SpiderOak CrossClave". The service targets users who want to work on their data collectively and persistently store it, supporting the platforms macOS, iOS, Windows, and Android. SpiderOak CrossClave guarantees client-side encryption and utilizes hardware-backed certificate chains. For authenticated file encryption, the service uses AES-Galois/Counter Mode (GCM). A decentralized authority implemented as blockchain is used for access rights to documents. Every time a folder is shared, a blockchain is generated with it. On this blockchain, all privileged user information or group membership is stored. Tampering with the blockchain is immediately noticed by all other parties of the system. Additionally, the blockchain is used for version control. All modifications of the documents are stored on the blockchain.

To summarize, as SpiderOak One uses password-based encryption, users do not face problems if they lose their devices. The client software can be reinstalled, and if the user still remembers the password, the access to the documents is restored. The

downside of this approach is that integrating hardware-backed keys is not possible, as the key used to decrypt the outer layer is computed with a key derivation function. Furthermore, the sharing mechanism of SpiderOak implies that everyone with access to the sharing link can read the shared data, including SpiderOak. With our approach, end-to-end encryption is still possible.

SpiderOak CrossClave allows users to share their data while keeping client-based end-to-end encryption intact. It also utilizes hardware-backed keys for multiple platforms. During the registration process of CrossClave, the client software presents a recovery key, consisting of multiple English words. If users lose access to all their devices or add a new device to their account, this recovery key must be provided. If the user also loses the recovery key, it is impossible to recover access to the account. With our system, the recovery process is not bound to a piece of information like a password. Depending on the concrete implementation of the recovery process described in Section 4.4, the availability of the needed parties may vary, but usually, the process is always possible to invoke.

3 Background

In this section, we recall basic background knowledge and definitions to understand the thesis. The first part of the section fundamentally discusses pairing-based cryptography. The second part states how Proxy Re-Encryption schemes work and enumerate all operations of such a scheme. The section concludes with a discussion on hardware-backed keys.

3.1 Pairing-Based Cryptography

To formulate the fundament of Pairing-Based Cryptography, it is necessary to define some mathematical operations. The following explanation is inspired by Menezes, 2009.

Let G_1, G_2 be two groups of prime order p which are generated by $Q \in G_1, R \in G_2$, respectively. Furthermore, let G_T be a group of order n. The groups G_1 and G_2 are additively written, and G_T is written multiplicatively. A bilinear pairing is a map e

$$e : G_1 \times G_2 \rightarrow G_T,$$

which satisfies the following properties:

Bilinearity: $\forall a, b \in \mathbb{F}_p^* : e(aQ, bR) = e(Q, R)^{ab}$.
Non-Degeneracy: $e(Q, R) \neq 1$
Computability: e can be computed efficiently

In the current definition, we described the bilinear map e on the domain $G_1 \times G_2$, but there are multiple ways to define the domain of e, as seen in, for example, Costello, 2012, yielding different "types" of pairings. If we look at the definition of the mapping e

$$e : G_1 \times G_2 \rightarrow G_T,$$

then the type of the pairing depends on the choice of G_2. The only difference between the pairing types is the technical implementation. The literature defines three different types of pairings:

Type 1 pairing: These pairings are obtained if we set $G_1 = G_2$. Most pairing-based protocols are written with pairings of this type because it has some mathematical benefits when defining a scheme. The first advantage is that there are no problems with hashing into the groups, and there is a trivial isomorphism $\phi : G_2 \to G_1$. The drawback is that the speed of computing the pairing is troublesome. Therefore, real implementations of cryptosystems do not employ this type of pairing.

Type 2 pairing: A type 2 pairing has two distinct groups G_1 and G_2 (meaning $G_1 \neq G_2$) and an additional isomorphic map $\phi : G_2 \to G_1$, which can be computed efficiently. The downside for this type of pairing is that it is unknown how to hash into G_2. Furthermore, it is impossible to generate random elements of G_2, except with scalar multiplication with the generator P_2. This is often not desirable.

Type 3 pairing: A type 3 pairing also has $G_1 \neq G_2$ but there is no isomorphism $\phi : G_2 \to G_1$ (at least none that can be computed efficiently). A pairing of this type is far easier to compute than pairings of the other types, and additionally, it is possible to hash into G_2. Due to this fact, most implementations of protocols based on pairing-based cryptography rely on this type. However, the downside when using this pairing type is the sacrifice of the map ϕ, which can be problematic, if the security proof of a scheme relies on the homomorphism.

Cryptographic algorithms rely on the hardness of well-investigated problems. With the above definitions, it is possible to define such a hard problem, namely the Bilinear Diffie-Hellman Problem (BDHP). The BDHP is based on the Diffie-Hellman Problem (DHP), proposed by Diffie and Hellman, 1976. The hardness assumption of the DHP is the Discrete Logarithm Problem (DLP). We recall the definition of the DLP (written multiplicative). Having a group G with order n, which is generated by the generator g, and an element $P \in G$, find an integer $x \in [0, n-1]$ such that $g^x = P$. For some chosen groups, this problem is thought to be sufficiently hard.

So, the DHP is defined as knowing the generator g of G and two elements g^a, g^b, find g^{ab}. If the DLP is hard, then the DHP is also hard to solve because it reduces in polynomial time to this problem. One group for which the hardness assumptions hold are points on elliptic curves over finite fields.

Due to the conditions on the bilinearity, the BDHP is given as having a bilinear map $e : G_1 \times G_1 \to G_T$ and elements P, aP, bP, cP, compute $e(P, P)^{abc}$. If one can solve

the DLP, it is also possible to solve the BDHP. This problem definition is thought to be as hard as DLP. The BDHP is the same for all different types of pairings.

3.2 Proxy Re-Encryption

This section summarizes the definitions of Proxy Re-Encryption (PRE) and states all operations of a PRE scheme. Furthermore, we give more details on Multi-Use Proxy Re-Encryption (MUPRE) schemes and the "Multi-Use" property.

Blaze et al., 1998 introduced the concept of PRE. In their work, they define an essential environment and notions for so-called Atomic Proxy Cryptography. Usually, in this setting, there are three players involved: Alice, who wants to share her data, Bob, who is the receiver of Alice's data, and a semi-trusted proxy.

PRE enables the proxy to transform Alice's ciphertext into another ciphertext, which only Bob, with his secret key, can decipher, so that no sensitive data, such as cleartexts or key material, is exposed to the semi-trusted proxy. So, PRE can be seen as a natural extension to asymmetric cryptography.

Alice is often referred to as the delegator and Bob as the delegatee of data. In this context, "semi-trusted proxy" means that the proxy is actively trying to get access to Alice's data or key material, but performs the underlying protocol correctly.

3.3 Multi-Use Proxy Re-Encryption

Up to this point, all definitions were tackling the use case that Alice wants to share her data with Bob. Schemes like that are called 'single-hop' schemes. A logical extension of this setting would be the addition of a new player Charlie.

If Alice wants to share her data with Bob, she enables the proxy to create a re-encrypted ciphertext for Bob. In the context of MUPRE, Alice's original ciphertext is called the first-level ciphertext, and Bob's re-encrypted ciphertext is called second-level ciphertext. Let us say Bob also wants to share Alice's data with Charlie, then again, Bob invokes the re-encryption operation on the proxy, yielding a third-level ciphertext for Charlie. Only Charlie's secret key is then, of course, able to decrypt this third-level ciphertext. Of course, it should also be possible to further re-encrypt the ciphertexts up to the l-th level. Fulfilling this condition implies that the scheme has the so-called "multi-hop" or "multi-use" property. Such schemes

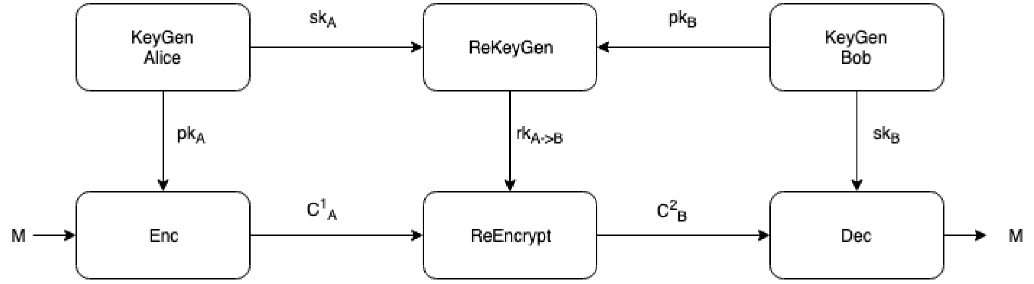


Figure 1: Data flow of a PRE scheme

are called Multi-Use Proxy Re-Encryption. We will stick to this naming convention throughout the thesis as our proposed scheme uses MUPRE. A formal definition of MUPRE can be seen in Definition 2.

Definition 2. *A MUPRE scheme with message space $\mathcal{M}$ has the following operations.*

KeyGen(1^κ) $\rightarrow$ (sk, pk): *This operation outputs a fresh, unique key pair, consisting of a public key and a secret key* (sk, pk)*, depending on the input parameter κ.*

Enc(pk, M) $\rightarrow$ C^1: *On input of a public key* pk *and some cleartext message $M \in \mathcal{M}$, the encrypt operation creates a first-level ciphertext C^1, which the respective secret key then can decrypt.*

Dec(sk, C^l) $\rightarrow$ M: *The input to this operation is a secret key* sk *and an l-th level ciphertext. If the given secret key can decrypt the ciphertext, it yields the message $M \in \mathcal{M}$ or $\perp$ otherwise.*

ReKeyGen(sk_A, pk_B) $\rightarrow$ $rk_{A \rightarrow B}$: *On input of Alice's secret key sk_A and Bob's public key pk_B, the operation creates a re-encryption key $rk_{A \rightarrow B}$.*

ReEncrypt($rk_{A \rightarrow B}$, C^l_A) $\rightarrow$ C^{l+1}_B: *Provided with a re-encryption key $rk_{A \rightarrow B}$ and Alice's l-th level ciphertext C^l_A, the semi-trusted proxy creates a $l+1$-th ciphertext C^{l+1}_B for Bob.*

Regarding the re-encryption key generation, we defined the input to the **ReKeyGen** operation as Alice's secret key and Bob's public key. This definition is a generalization as the input can differ from that. We will discuss the input to this function in more detail later in the thesis when we define the non-interactive property (see Section 5).

Figure 1 examines the data flow between the individual operations. In this figure, the dependencies of the operations are shown when performing a single re-encryption. For MUPRE schemes, the **ReEncrypt** step can, of course, be applied multiple times.

3.4 Hardware-Backed Keys

In this section, we will describe the concept of hardware-backed key material. Alongside reasoning why such keys exist, we will frame the prerequisites that have to be met to be able to use these keys and how they are implemented.

When integrating cryptographic building blocks in a system, designers face specific problems, namely, among others, the secure creation, protection, and usage of cryptographic key material. In every system which relies on cryptographic primitives, these aspects have to be considered. Some challenges are more evident than others, for example, the storage of secret keys. Of course, only the person who created the key and later plans to use it shall access the secret key. However, to use the key, it has to be persistently stored somewhere. A straightforward solution is to store the key on the device's hard drive, which created the key. The apparent downside to this approach is that malicious software that runs on the same device would be able to access the secret key, hence virtually no key is stored in this manner. A more sophisticated approach to securely store keys is a dedicated file format, like the format defined in PKCS#12 by Moriarty et al., 2014. Usually, PKCS#12 is used to bundle a secret key with one or more corresponding X.509 certificates together, then encrypted with a passphrase. Unfortunately, to access these files, systems usually need to access the passphrase via code or configuration files, where the passphrase is stored in plaintext.

Nevertheless, there are still limitations using dedicated file formats, although they are reasonably secure. One such downside is that it is impossible to bind the key material to a specific piece of hardware, meaning that the key can be used on any device if the passphrase is known and the file was exchanged. Furthermore, if the key material is used in a cryptographic operation, the secret key's plaintext has to be loaded into the RAM of the device to perform its task. This necessary step opens a vast amount of different attack surfaces, e.g., side-channel attacks or malware vulnerabilities. Hardware manufacturers recognized these problems and introduced the concept of the so-called Trusted Execution Environment (TEE).

The TEE is a dedicated, isolated area inside the main processor of a device. Sometimes the TEE is implemented as a separate chip. In this dedicated area, only specific pieces of software can be executed, alongside a TEE kernel. These programs are called "Secure Applications" and use the full functionality of the processor. Opposed to the TEE is the Rich Execution Environment (REE), where the kernel of the operating system and user applications are executed. Although the TEE and

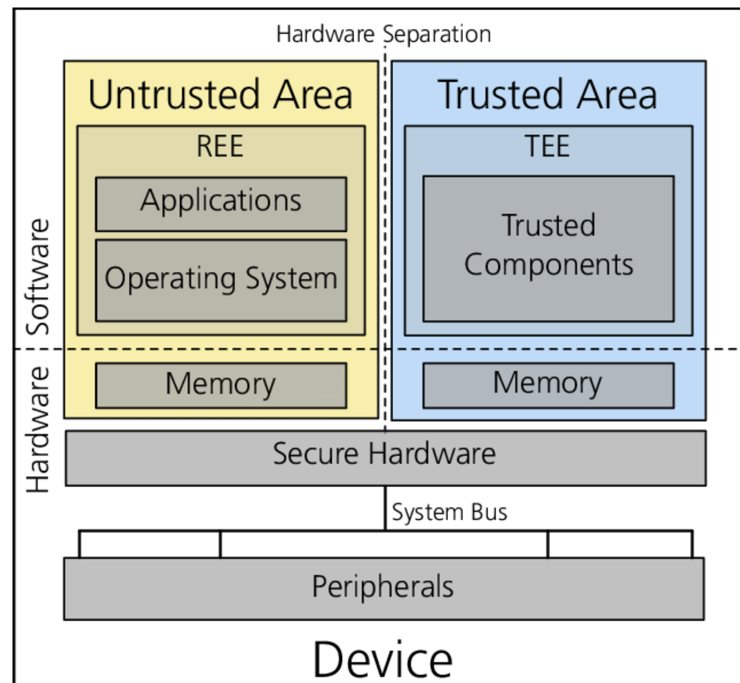


Figure 2: The separation of the TEE and REE

.

REE share the same processor, the REE does not have the same capabilities as the TEE. Furthermore, the TEE has access to more memory regions than the REE.

Similar to the context switching mechanics in operating systems, there exists a context switch to enter the TEE via well-defined manners. Communication between the TEE and the REE is done via shared memory, and these context switches. This clear partitioning between the two areas depends on hardware design and software. Figure 2 shows the separation of the two worlds. The picture is obtained from González, 2015.

Prominent examples for TEEs are the ”TrustZone” from ARM and the Software Guard Extensions (SGX) from Intel. On devices that support a TEE, it can be utilized to create cryptographic keys securely. Keys are generated inside the TEE and stored in areas where the access of the REE is prohibited. Such keys are called hardware-backed. Also, cryptographic operations using the hardware-backed keys are performed in the TEE so that the key never leaves the TEE. Even if the system is infected with malicious malware or the kernel of the operating system is exploited,

it is not possible to compromise the key material due to the hardware and software protection of the TEE, implying that a key is bound to the TEE (and ergo the device), which created the key material. Even if attackers get physical access to the device, it is not possible to extract the key material from the device.

The security of all operations we listed above also depends on the correct authorization of incoming requests. In other words, it is necessary that only authorized processes can use their respective keys. To give an example, the "TrustZone" ensures this with a secure boot chain. During the booting process of a device, every piece of software, including the REE and the secure applications, are verified by validating a signature computed over the code that is loaded. If the secure boot fails for whatever reason, the device does not start. This secure boot chain implies that if the device did start correctly, the TrustZone can trust the REE that it handles all incoming requests correctly.

In this section, we described the concept of hardware-backed key material. Alongside reasoning why such keys exist, we gave the prerequisites which have to be met to be able to use these keys and how they are implemented.

4 Multi-User Data Sharing System and Recovery from Device-Loss Strategies

This section describes the proposed multi-user data sharing system, which is the main focus of the thesis. The system is cloud-based, with a novel user-friendly solution for recovery from key-loss utilizing MUPRE with trust decisions that the user chooses on their own. The first section will list the challenges the system faces. The second section states all actors within the system, their operations, and how they interact. A discussion on trusted recovery users can be found in section three. The fourth section discusses the different approaches of key-authenticity, which can be used in the system. The fifth section states all trust assumptions between the actors.

4.1 Challenges

In this section, we will state all the challenges our system faces. Furthermore, all objectives which have to be met are listed.

Confidentiality A significant demand for our system is the confidentiality of the documents uploaded by users. As users may upload sensible data to the cloud storage, it shall be, of course, only possible for privileged actors in the system to get access to these documents. Additionally, users shall have absolute sovereignty over what happens to their documents. It shall only be possible for entities to read data from a different user if the said user previously agreed.

Integrity Alongside confidentiality, integrity is a fundamental basic concept in information security. During all steps in the system, it shall not be able to alter data from users undetected. Only a user themself, or any entity that got bestowed with the user's rights, shall be able to modify the data.

User Experience As we focus mainly on building a system that may be used by real users, user experience is one of the most significant aspects of our system. All cryptographic operations should be embedded in the system such that users do not feel a negative impact on the performance. A simple, understandable interface

shall be shown to users where it is easily possible to store and securely share their data. Furthermore, as the underlying cryptography is quite complex, there may be performance impacts. These problems need to be addressed.

Sharing of Documents One of the central functionalities of our system is the sharing of documents between users. As already stated, it shall only be possible for authorized users to access data uploaded to the cloud storage. The possibility to bestow access rights of documents to distinct users shall be limited to the data owner. Additionally, to server-side access validation, we incorporate cryptography such that it is also mathematically unfeasible to gain unauthorized access to documents.

Recovery after Device-Loss Users of our system shall have the possibility to recover the access to their documents after losing all their devices and, therefore, their key material. Additionally, this recovery process shall imply minimal trust decisions by the user that lost all their phones.

Protection of the Key Material The key management is one of the core aspects of our proposed system. Our approach enables the possibility that the users' private key material can be bound to the hardware's device where it was created, which renders it unextractable. For more information about hardware-backed keys, we refer to Section 3.4.

4.2 Data Sharing

This section explains how the actors of the system interact. For a quick overview of the actor's workflow and how they interact, cf. Protocol 1.

① **Register at Service** Multiple **users** want to store their data securely. Therefore they register at a **cloud storage system**. For all users who register at the service for the first time, some space at the cloud storage is allocated. This space can be used to store the users' documents later on. This device will be the so-called Primary Device (PD). The PD creates a hardware-backed MUPRE key pair and transmits the public key to the service. After this step, the registration is complete.

② **Upload Documents** After the registration, users can start to upload their data to their reserved space on the cloud storage system. The data is encrypted with the key material of the user's PD.

③ **Register Secondary Device (SD)** Users may want to register multiple devices at the cloud storage alongside their PD. After creating fresh key material, the cloud service notifies the PD over some secure channel, where the registration of a new SD is authenticated or not. This workflow implies that it is necessary to have physical access to the PD to register a new SD. In a real-world application, this authentication step can be extended by providing another piece of information, e.g., a password, resulting in a two-factor authentication.

Such additional devices are called Secondary Device (SD). The PD of the user creates a re-key, which can transform ciphertexts from the user's PD to the newly registered SD. When signed in with the SD, uploaded ciphertexts can then be decrypted after a single re-encryption. If a user wants to upload data from their SD, the data is encrypted with the public key from the PD nevertheless. There are multiple approaches on how to tackle key-authenticity of Primary and Secondary Devices. Key-authenticity is discussed in more detail in Section 4.5.

④ **Share Data (Pairing)** Users are also able to share their data with different users. Imagine a user that wants to share their medical data with a medical authority in a confidential matter. The authority initially has to perform a request to the user to get bestowed with decryption rights to gain access to the medical data. If the user agrees, they create a re-key that can transform ciphertexts from the user's PD to the authority's device that sent the request. This can either be a PD or an SD. Then the user installs the re-key at the cloud service. After this step, the medical authority can decrypt the requested data.

⑤ **Download Data** Of course, it is necessary to provide users the functionality to download the documents for which they got access rights. In case the documents are requested from the PD of the user, which uploaded them in the first place, the approach is straightforward. The data is served in encrypted form, and the PD is easily able to decrypt the data as the PD has access to its private key.

If users want to access their data from their SD, they can't decrypt the encrypted documents. So, the cloud storage needs to get active. The cloud storage uses

the previously created re-key and re-encrypts the requested document. The re-encrypted ciphertext is served to the SD, where it now can be decrypted.

A similar approach is taken when users want to download documents from distinct users. The cloud storage re-encrypts the data with the previously installed re-key, which was created during the pairing process. As the device requesting the data may be an SD, there may be the need for more than one re-encryption.

Recall the example from the previous paragraphs. Some user bestowed a medical entity with the right to read their documents. The medical entity wants to download said documents on one of its SDs. The cloud storage searches for a path of re-keys, enabling it to re-encrypt the documents from the original user's PD to the medical entities' SD. If such a path is found, the cloud storage re-encrypts the data and serves the l-th level ciphertext to the entities' SD.

Figure 3 shows a simplified process flow of all the operations mentioned above in the system. This diagram shows how the operations listed in 4.2 are executed if Bob wants to download Alice's documents on his SD.

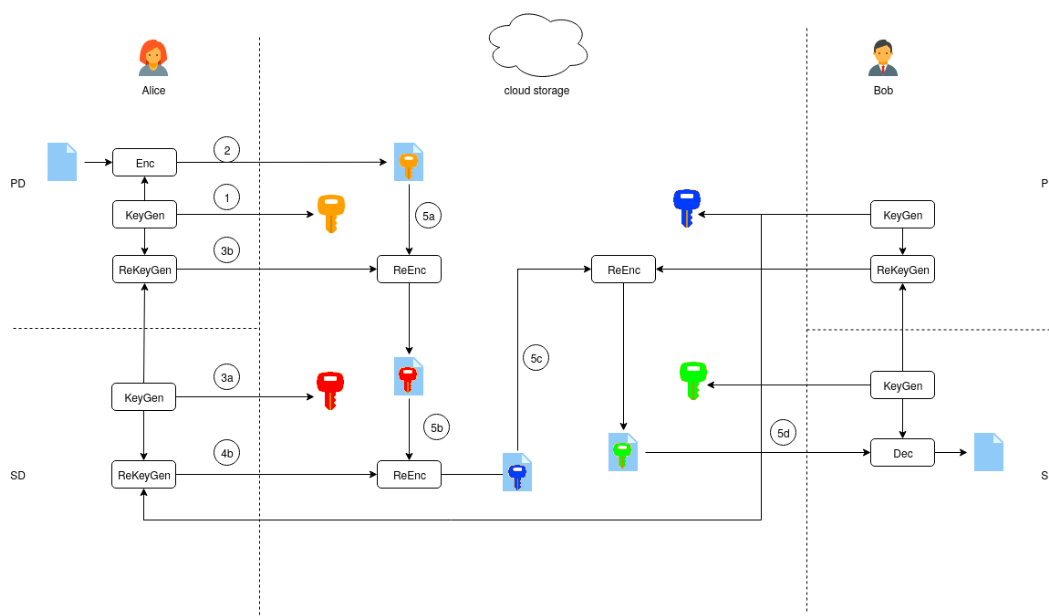


Figure 3: Complete operation flow of the system

① **Register User**

Alice: on her PD

1. Alice creates a key pair $(\mathrm{sk}_{A1}, \mathrm{pk}_{A1}) \leftarrow \mathrm{KeyGen}(1^\kappa)$
2. Alice sends her public key pk_{A1} to the cloud storage. Her device is registered as PD

② **Upload Data**

Alice: on any device

1. Alice obtains the public key of her PD pk_{A1}
2. Alice invokes the encryption function and creates a first-level ciphertext $\mathrm{C}^1 \leftarrow \mathrm{Enc}(\mathrm{pk}_{A1}, \mathrm{M})$ of her document M
3. Alice sends the first-level ciphertext C^1 to the cloud storage

③ **Register SD**

Alice: on her secondary device

1. Alice creates a key pair on her device $(\mathrm{sk}_{A2}, \mathrm{pk}_{A2}) \leftarrow \mathrm{KeyGen}(1^\kappa)$
2. Alice sends her new public key pk_{A2} to the cloud storage. Her device is registered as SD
3. Alice sends her new public key pk_{A2} to her PD

Alice: on her primary device

4. Alice verifies the registration of the SD
5. Alice creates a re-key $\mathrm{rk}_{A1\rightarrow A2} \leftarrow \mathrm{ReKeyGen}(\mathrm{sk}_{A1}, \mathrm{pk}_{A2})$
6. Alice sends the re-key $\mathrm{rk}_{A1\rightarrow A2}$ to the cloud storage

④ **Share Data**

Bob: on any device

1. Bob sends the public key of his PD pk_{B1} to Alice

Alice: on any device k

2. Alice verifies the request from Bob
3. Alice creates a re-key $\mathrm{rk}_{Ak\rightarrow B1} \leftarrow \mathrm{ReKeyGen}(\mathrm{sk}_{Ak}, \mathrm{pk}_{B1})$
4. Alice sends the re-key $\mathrm{rk}_{Ak\rightarrow B1}$ to the cloud storage

⑤ **Download Data**

Alice: on any device k

1. Alice requests document M from cloud storage which is encrypted under pk_*

cloud storage:

2. cloud storage searches for a path of re-keys which are able to translate M from pk_* to pk_{Ak}. This path is denoted as $rk_{* \to k}$
3. cloud storage re-encrypts ciphertext to an l-th level ciphertext $C_A^{l+1} \leftarrow \mathrm{ReEncrypt}(rk_{* \to k}, C^l)$ and sends it to device k

Alice: on device k

5. Alice decrypts the ciphertext and obtains the document $M \leftarrow \mathrm{Dec}(sk_k, C_A^{l+1})$

Protocol 1: Protocol of basic operation in the multi-user data sharing system

4.3 Recovery from Device-Loss

The system's incorporated key management, which utilizes MUPRE, does not only enable a user-friendly way to share data between multiple devices and users but also provides a novel way to recover data in the case that users forfeit access to their devices and in turn their key material. With classical public-key cryptography, there would be an inherent problem, as the loss of the key material usually results in a complete loss of the associated data. With our key management system, it is possible to restore all data access in a confidential and user-friendly way.

There are three different strategies to recover from device-loss. Depending on the situation, a different strategy has to be applied. The user that lost their device can solve two of those scenarios. In the third scenario, another *trusted recovery user* has to get active. The workflow of the actors during the recovery strategies can be found in Protocol 2.

ⓐ Loss of SD Let us suppose Alice loses her SD and in turn the device's key. First of all, Alice has to deregister the device from the system. This step ensures

that no documents can be downloaded to the lost device, which could lead to data leakage otherwise, if the device was stolen. Since Alice's documents are encrypted with the key material of the lost device, there is no need to perform recovery operations as Alice still has access to the documents via her PD.

Another aspect that has to be considered are the connections from the lost device to devices from distinct users. These connections can either be *incoming* or *outgoing*. To get back to the same state of the system before the device-loss, some of these connections have to be re-generated.

The more trivial problem are the outgoing connections. These connections are the re-key paths where Alice bestows decryption rights to distinct users via the lost device's key material. Due to the existing re-key network, it is, in fact, not necessary to generate new connections from another device. If Bob could read Alice's data via the lost SD, then this path is still intact, and all documents from Alice can be re-encrypted like before. Of course, this approach implies that it is indispensable that the proxy does not serve the intermediate re-encryptions, as they could be decrypted with the lost SD.

Conversely, if the lost device was used to read data from distinct users (incoming connections), the incoming re-keys have to be deleted. Alice then has to request decryption rights with another device. Now distinct users can decide whether they want to share their data again or not. These requests can either be done explicitly by Alice or may be performed implicitly by the system.

This scenario is illustrated in Figure 4. The diagram shows the messages sent between Alice, who lost her SD, the cloud service, and Bob, who bestowed decryption rights to Alice's lost device.

ⓑ Loss of PD with Existing SD Imagine that Alice forfeits access to the key material of her PD, but registered one or more SDs some time beforehand. Alice chooses one of her SDs, which then becomes her new PD. Her old PD is deregistered, and no documents are served to this device anymore. However, all newly uploaded documents are still encrypted with her old PD. She still has access to her documents due to the existing network of re-keys. Furthermore, if Alice owns other SDs, re-keys which translate ciphertexts from her new PD to the other SDs are created.

Again, the incoming connections from the lost PD have to be restored. This process is the same as for the loss of an SD.

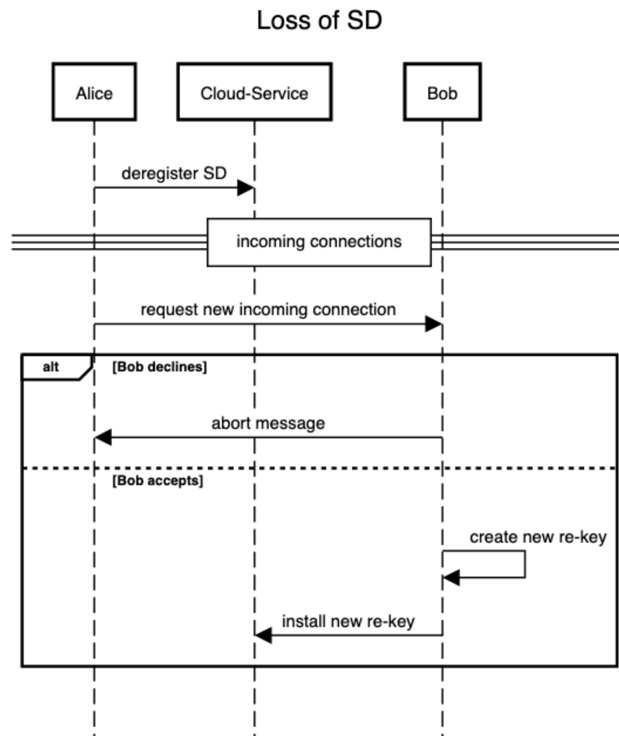


Figure 4: Messages sent to recovery from SD loss

ⓒ Loss of PD without Existing SD The most interesting case is if Alice loses her PD but does not have any other device registered at the cloud service. As the other two recovery strategies are not applicable, a third party needs to get active, namely a *trusted recovery user*. There are multiple ways of how a trusted recovery user can be implemented in the system. The concrete implementations are further discussed in Section 4.4.

Let us say, Bob acts as a single recovery user for Alice. This means that Alice generated a re-key that can transform ciphertexts with respect to her lost PD to ciphertexts that Bob would be able to decrypt with one of his devices. After Alice realizes she lost her PD, she has to register a new device at the cloud service and creates a fresh key pair for this device. How she obtains the new key pair and ensures key-authenticity is discussed in Section 4.5. She then authenticates herself to Bob, sends him the public key of her device, and requests data recovery.

First, Alice convinces Bob that the request is indeed coming from her. In a real-

world application, a trusted-user may be a family member or a close friend. Bob then creates a re-key that can translate ciphertexts from Bob's device to Alice's new device and sends this key to the cloud service. This authentication (password authentication at the service, authentication at the trusted user) convinces the cloud service that the requesting party is indeed Alice.

The cloud service invokes the recovery process, which consists of multiple re-encryptions. All documents encrypted under the key material of Alice's old PD are re-encrypted twice. Firstly the documents are re-encrypted for Bob's key material and secondly to Alice's new PD. This process is illustrated in Figure 5. Bob's intermediate ciphertexts are only used for the re-encryptions and are deleted afterward. Alice is now able to login with her new PD at the cloud service, enabling her to upload documents. Furthermore, she has restored the access to her previously stored documents.

After the new device is successfully registered as PD, the recovery of the incoming and outgoing re-key connections of Alice's old PD has to be started. This operation is done in the same way as in the other two recovery strategies.

ⓐ **Loss of SD**

Alice: on her PD

1. Alice deregisters her lost SD from the cloud storage

ⓑ **Loss of PD, with Existing SD**

Alice: on her SD (new PD), with key pair $(\mathrm{sk}_{A2}, \mathrm{pk}_{A2})$

1. Alice deregisters her lost PD from the cloud storage
2. Alice creates a re-key for every other SD she owns $\{\mathrm{rk}_{A2 \to A_i} \leftarrow \mathrm{ReKeyGen}(\mathrm{sk}_{A2}, \mathrm{pk}_{A_i}) : \forall i \in \mathrm{SDs}\}$
3. Alice sends the re-keys to the cloud storage. Her PD is updated.

ⓒ **Loss of PD, without Existing SD**

Alice: on new device $*$

1. Alice creates a key pair $(\mathrm{sk}_{A*}, \mathrm{pk}_{A*}) \leftarrow \mathrm{KeyGen}(1^\kappa)$
2. Alice sends her new public key pk_{A*} to the cloud storage and invokes the recovery process

cloud storage:

3. cloud storage sends Alice's public key pk_{A*} to her recovery user Bob

Alice: via some channel

4. Alice authenticates herself to Bob

Bob: on any device k

5. Bob creates a re-key $\mathrm{rk}_{Bk \to A*} \leftarrow \mathrm{ReKeyGen}(\mathrm{sk}_{Bk}, \mathrm{pk}_{A*})$
6. Bob sends the re-key to the cloud storage

cloud storage:

7. cloud storage re-encrypts Alice's documents with the previously stored re-key from her old PD to Bob's device k $\mathrm{C}^2_B \leftarrow \mathrm{ReEncrypt}(\mathrm{rk}_{A1 \to Bk}, \mathrm{C}^1)$
8. cloud storage re-encrypts the intermediate ciphertexts such that Alice can decrypt them with her new PD $\mathrm{C}^3_A \leftarrow \mathrm{ReEncrypt}(\mathrm{rk}_{Bk \to A*}, \mathrm{C}^2_B)$
9. cloud storage deregisters Alice's old PD

Protocol 2: Protocol of the device-loss recovery strategies

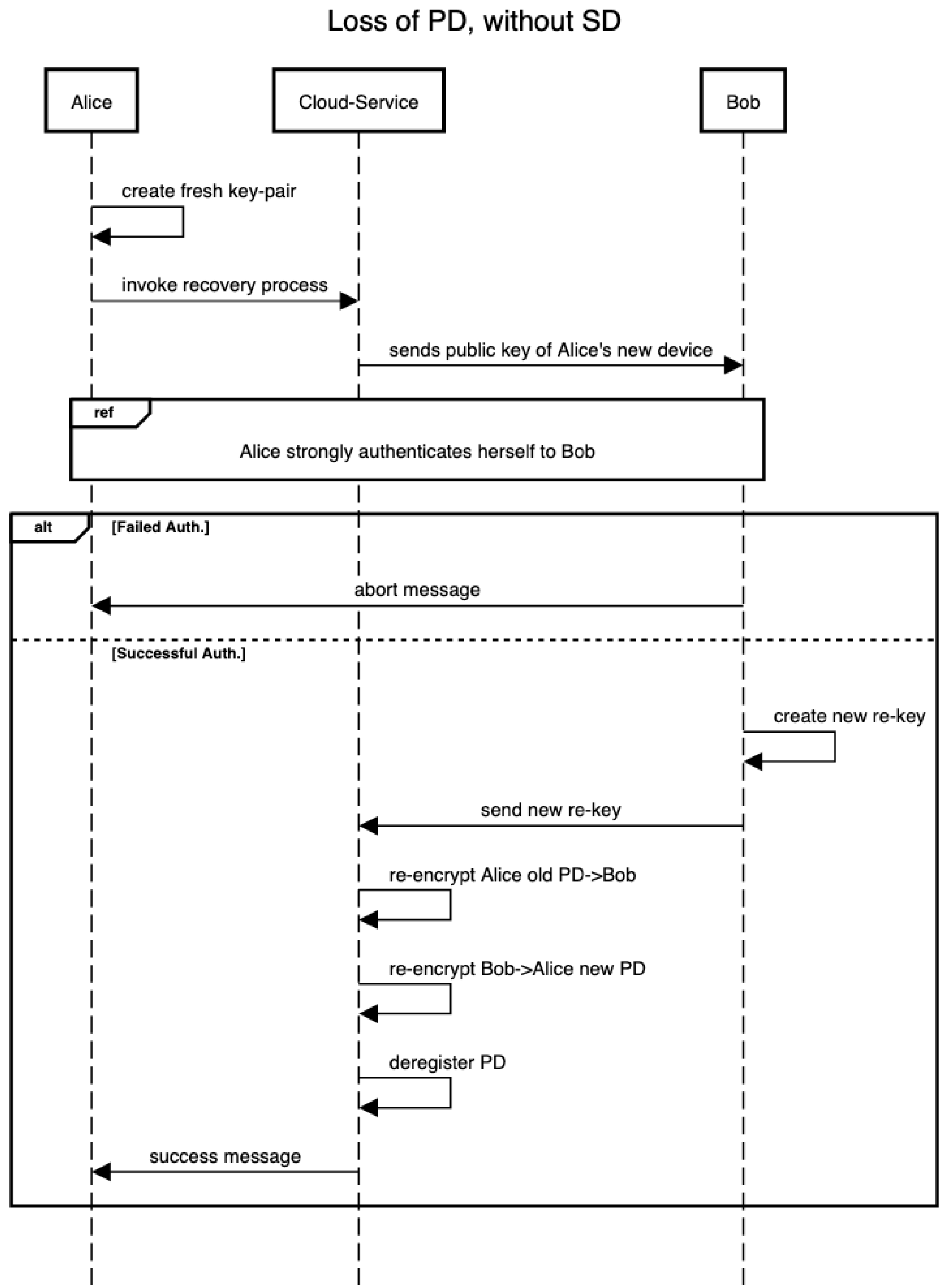


Figure 5: Messages sent to recover from PD loss when there is no SD registered

4.4 Trusted Recovery Users

This section will discuss how trusted recovery users can be implemented in the system in more detail. Trusted recovery users are needed for the recovery process if a user forfeits access to their PD and does not have an SD registered at the service.

As a trusted recovery user works mainly as an intermediate step in the recovery process, there are multiple ways they can be integrated into the system. Nevertheless, certain restrictions have to be considered. Recovery users also work as a "second-factor" authenticator for the user who lost their PD. The fact that it is necessary to convince the recovery user to start the recovery process also makes it harder to misuse the recovery strategies by attackers. So, this authentication step has to be taken into account, as it should only be possible for the user, who lost their PD to start the recovery process. The different versions of recovery users differ in availability and the needed trust assumptions.

Distinct User As described in the example in Section 4.3, a recovery user may be a distinct user of the data sharing system, e.g., a family member or a close friend. The authentication in this scenario is straightforward, as it is easy to identify oneself to an acquaintance, especially if said person is a family member. Furthermore, it is hard to convince the recovery user to start the recovery process for an attacker.

The recovery with distinct users can also be implemented in different ways. One possibility is that every user has to choose one recovery user at registration. Then the registration process defined in Section 4.2 would be extended by the choice of the recovery user. Furthermore, the registree creates a re-key from their PD to one of their recovery user's devices, which is then installed at the cloud service. This re-key is only used if the recovery process is invoked, so the recovery user cannot read the registree's documents.

A different possibility is that there is not a single, dedicated recovery user, but every user who was bestowed with decryption rights can act as a recovery user if specified by the recovering user. With this approach, there could be multiple recovery users. On the one hand, this approach's apparent advantage is the availability of recovery users, as the start of the recovery process is not bound to a single user. On the other hand, it is necessary that the recovery user also has access to the other user's documents. Furthermore, if a user did not share their documents, it is also not

possible to recover their documents. Also, it may get easier to impersonate a user if there are multiple potential recovery users.

Additionally, trusted (possibly commercial) third party services can work as recovery users. For every user, the recovery service creates a fresh key pair. The user then has to create a re-key from their PD to the key pair of the recovery service and install the re-key at the cloud service. Also, a strong authentication system for the user at the recovery-system has to be set up. In this scenario, the third party service would act as the second-factor authenticator for the recovery process. A single password authentication would be too weak. Of course, it is prohibited for these services to be affiliated with the cloud service, as this would break one of our fundamental assumptions. If the cloud service and the recovery service collude, it is easily possible to obtain the documents of all users who rely on the recovery service.

Recovery users implemented as a single distinct user are a straightforward solution, which meets all identified requirements. If availability is necessary, the solution can be further tweaked to increase availability at the cost of increased trust assumptions. The recovery user as second-factor in a two-way authentication is also reasonably secure.

Threshold of Users A more favorable trade-off between availability and trust assumptions is to incorporate a secret-sharing mechanism that was originally proposed by Shamir, 1979. During the registration process, the user chooses n recovery users and creates a re-key for each of them. Every re-key is split into n shares. Every i-th share is encrypted for the i-th recovery user. The resulting ciphertexts are stored at the cloud system. Once the user invokes the recovery process, they need to convince a predefined threshold of t users to download their respective shares. These t users decrypt their shares and transmit them to the cloud service. The cloud service can then reconstruct at least one re-key of the transmitted shares and perform the re-encryptions. All messages which are sent in this scenario can be seen in Figure 6. The messages shown in this diagram are sent between Alice, who lost her PD, the cloud service, and one of Alice's recovery users, Bob. In fact, Alice's messages to Bob in this diagram are sent multiple times to each of her n users.

The main advantage in this scenario is that the re-key used during the recovery process is not accessible unless a threshold of t users collude with the cloud service. If t is chosen reasonably large, the possibility for this to happen is minimal. Also, the availability of t users is realistic.

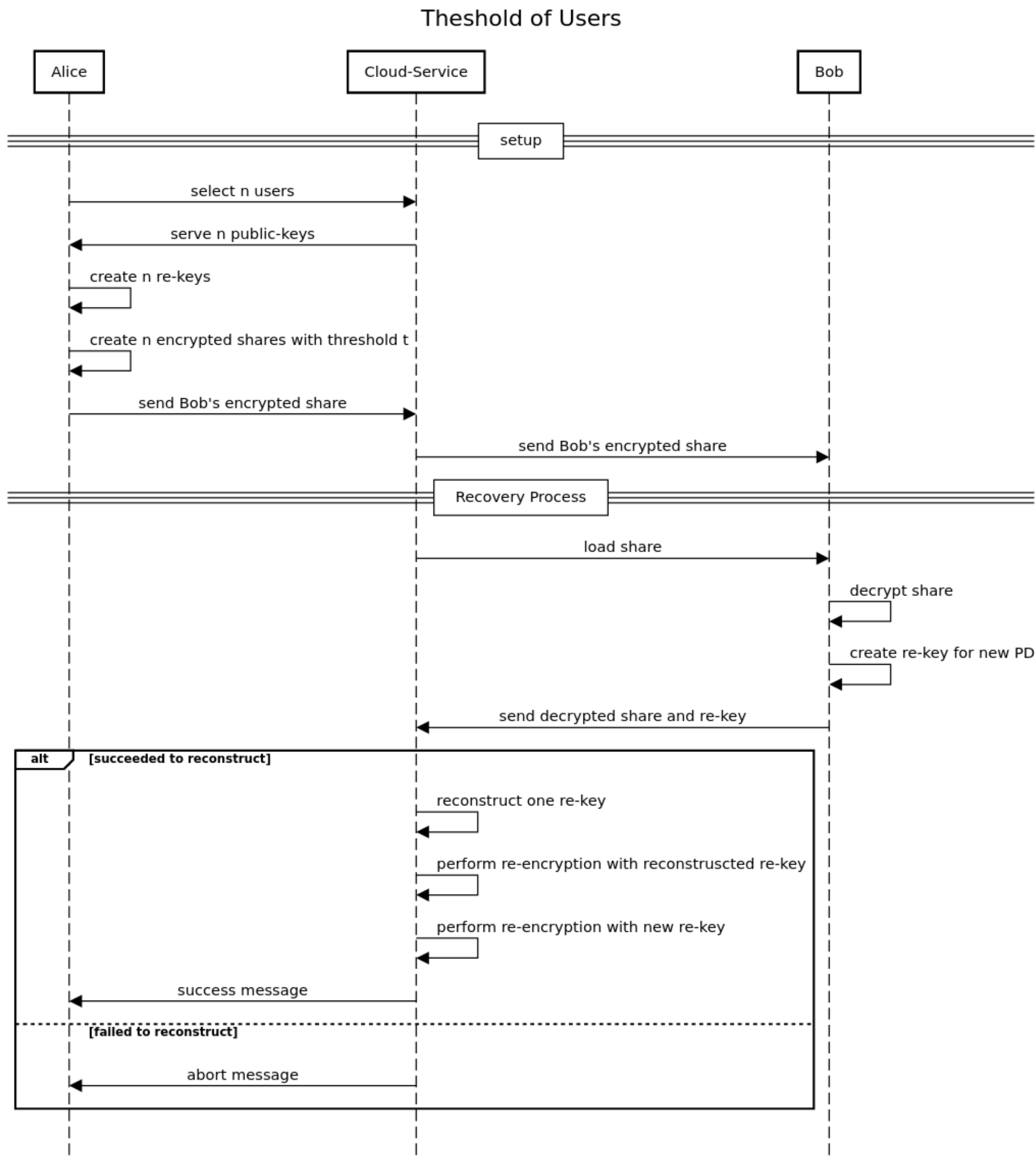


Figure 6: Messages sent to recover with a threshold of trusted users

4.5 Key-Authenticity

During virtually all steps, actors have to interact with the keys of different devices. These interactions include, among others, communication between the devices which belong to the same user, and communication between devices that belong to different users. Therefore some sort of key-authenticity has to be implemented in the system. There are different approaches to how key-authenticity can be ensured. In this section, we will give some solutions regarding how key-authenticity can be integrated into the system. Furthermore, we will discuss the advantages and disadvantages of the different approaches.

Manual Key-Authenticity A straightforward solution is a user-driven manual key-authenticity without any PKI and digital signatures. When pairing a device with another user's device, irrelevant whether the devices are PDs or SDs, the users verify the involved keys themselves. This verification can be done quickly if the users compare their public keys' fingerprint via, e.g., a phone call or scanning QR-codes.

During the registration process of devices, each device creates a hardware-backed key. If the new device is an SD, then the registration is authorized with the PD. Users can verify the key of the SD by merely comparing the fingerprint of the public key, which is displayed on the SD, and the authorization request on the PD.

On the one hand, the low implementation effort, in comparison with other solutions, is advantageous. On the other hand, there are multiple downsides when using this approach. First of all, users have to get actively involved in the key verification process. Furthermore, a key's authenticity is dependent on the judgment of users. Also, the cloud service does not have any secure means to authenticate keys.

However, the biggest shortcoming of this approach is the inability to authenticate devices during the recovery process with an existing SD. It would be beneficial to ensure the recovery request's authenticity with some sort of key-authenticity, which is not possible with this approach. There would be the need for additional secret information on the device, to ensure that the request is indeed coming from the SD.

The manual key-authenticity is easy to implement, but the downsides outweigh the positives.

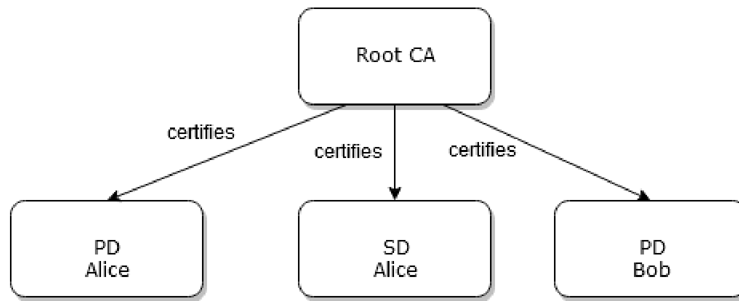


Figure 7: Flat hierarchy of certified public keys

Flat PKI Structure Of course, it is also possible to use traditional, centralized Public Key Infrastructure (PKI). A new actor would be added to the system, namely a third party root Certificate Authority (CA). Furthermore, we introduce new key pairs for every device. These keys are called signing-keys. These signing-keys are used to certify the re-encryption (encryption) key pairs of the same device by signing the public key's fingerprint of the encryption key pair. Signing-keys could, for example, be Elliptic Curve Digital Signature Algorithm (ECDSA) key pairs and have a human-readable identifier to let users quickly identify the key pair. Of course, the key pair used for signing can be created in the device's hardware.

The root CA certifies every signing public key during its device's registration process. This approach would lead to a flat hierarchy of certified public keys, as seen in Figure 7. In this example, Alice owns an additional SD alongside her PD. Bob only registered his PD.

During each operation, a key's authenticity needs to be validated. The parties request the corresponding signing public key, validate the signature on the encryption public key, and then check whether the root CA certifies the signing public key.

During the pairing process, users may further validate the key by a manual key-authenticity check (as described in the previous section). This can be achieved by assigning a naming scheme for the signing key pairs. One such naming scheme could be, for example, that Alice's PD is called 'Alice-PD'. Every SD Alice registers, later on, could be incrementally named, like 'Alice-SD1', 'Alice-SD2', and so on. With such a naming scheme, users can verify the other parties' key easily. Of course, the root CA has to ensure that these identifiers are not issued multiple times.

In contrast to the manual key-authenticity, the cloud service can easily verify the authenticity of keys and requests with this approach. During the recovery

process with an SD, the proxy may enforce the additional requirement that the request is signed with the signing key of the SD. This added security measure leads to a two-factor authentication during the recovery process with an SD, as now, alongside the user's password, physical access to an SD is needed to invoke the recovery process. For the recovery with a trusted recovery-user, the second-factor validation is outsourced to the recovery-user, as there is no access to the devices' key material.

Nevertheless, one shortcoming of this approach is that if we want to add a naming scheme for the signing key pairs, the flat PKI structure implies that the effort to identify the SDs of users is on the CA. The CA does not have the means to verify if a key-authentication request is indeed coming from the user's SD. Therefore it would be hard to bind the keys to a single user. One possible solution to this problem is described in the next section.

Hierarchical PKI Structure Looking at the flat PKI structure defined in the previous section, there is no easy way to identify that an SD belongs to a dedicated user. This problem can be solved with a hierarchical PKI structure.

We again use the concept of signing-keys for each device, as described in the last section. Each signing-key certifies its corresponding encryption key. However, in contrast to the flat PKI structure, the signing-key from the PD works as intermediate CA for the signing-keys of the user's SDs. During the registration of an SD, the device's signing public key is transmitted to the PD, where it is certified by the signing-key of the PD, yielding a PKI structure, which can be seen in Figure 8. In this example, Alice owns an additional SD alongside her PD. Bob registered two SDs.

When using this PKI structure, the loss of the PD leads to some problems. If the user invokes the recovery process with an SD, the cloud service can still verify the authenticity of the device by traversing the certificate chain. The SD becomes the user's PD, and its signing-key can then certify the newly registered SDs, which increases the length of the validated certificate chains by one.

The more interesting case is the loss of the PD when the user does not own an SD. As the user does not have access to any of their devices, it is also impossible to authenticate the keys of the new device, which should act as the recovery device. Furthermore, it is also not possible to back-up the signing-key of the PD somewhere before the device-loss, as this would break one of our fundamental requirements defined in Section 4.1, namely the hardware backing of keys. To solve this problem,

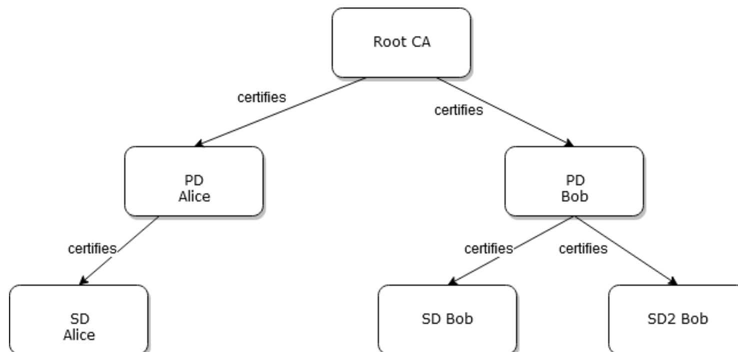


Figure 8: The hierarchical structure of certified public keys

the user needs to convince the root CA to issue a new key pair for the recovery device.

The user creates signing-keys and encryption-keys in the device's hardware and convinces their trusted recovery user to create a re-encryption path from the old PD to the recovery device. Then the root CA creates a challenge ciphertext that is encrypted under the encryption-keys of the lost PD. If the user can decrypt the challenge and transmit it back to the CA, then the authorization with the trusted user was successful, and the CA certifies the new PD's keys. The messages sent between the parties can be seen in Figure 9.

In this section, we gave some solutions regarding how key-authenticity can be integrated into the system. Furthermore, we discussed the advantages and disadvantages of the different approaches.

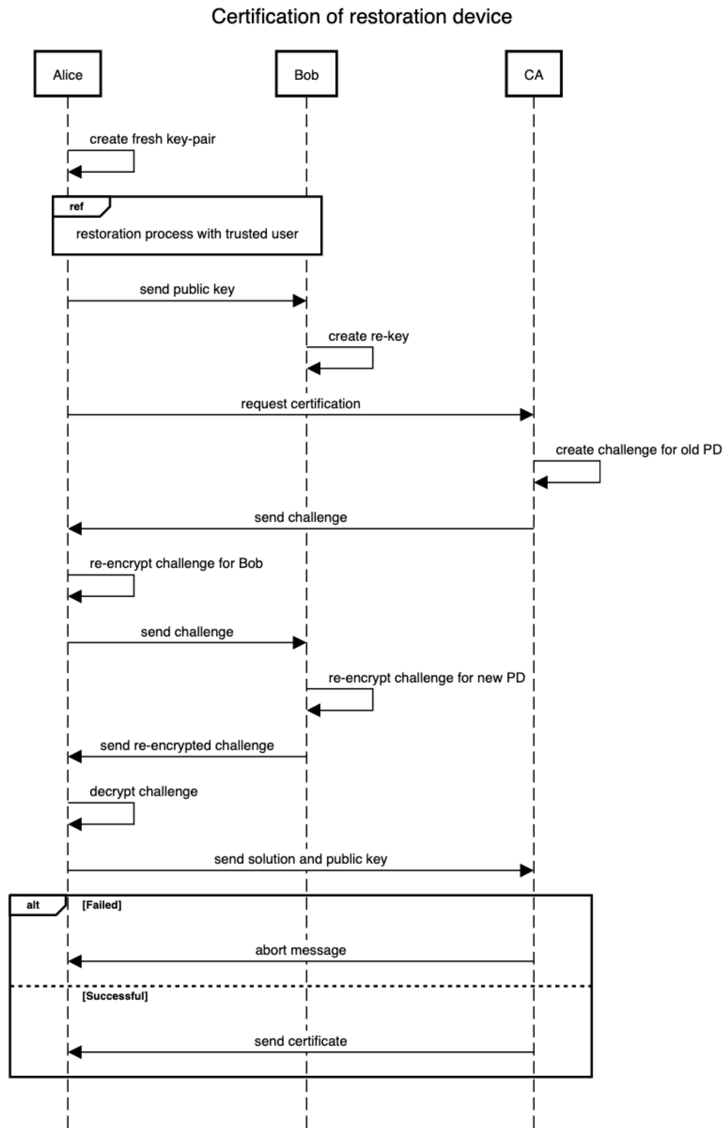


Figure 9: Certification of Alice's recovery device with her trusted recovery user Bob

4.6 Trust Assumptions

One of the multi-user data sharing system's main goals is to minimize trust assumptions between the actors while preserving the confidentiality of the data. This section will state the trust assumptions fundamental to the system and how these assumptions are preserved.

As we are in a semi-trusted setting due to the nature of PRE, every user trusts that the cloud service follows the protocol as intended, but it may be curious about the data it operates with. We incorporate MUPRE to ensure the confidentiality of the users' data.

The cloud service's curiosity implies that every user has to trust the users that they granted decryption rights to not collude with the cloud service. Otherwise, it would be possible for the cloud service to send users' data to corrupted users, where the documents could be easily decrypted. Users have to consider which users they trust enough to share their data. Key-authenticity, as described in Section 4.5, helps the users to decide whether to accept a pairing request or not.

Of course, users have to trust their recovery users. First of all, they act as the second-factor of the two-factor authentication in the recovery process. So, a user in recovery trusts their recovery user to authenticate them at the cloud service when needed. Furthermore, the user trusts the recovery user not to authenticate any other person who tries to impersonate the user. We propose to rely on the recovery process with a threshold of users, as described in Section 4.4. When using the approach mentioned above, authentication errors become reasonably small when the parameters involved are chosen correctly. Additionally, the trusted user(s) shall not collude with the cloud service. The trusted user cannot read the user's documents during the recovery on their own, but they may be able to if they collude with the cloud service.

Furthermore, the user trusts their device's hardware that ensures the non-extractability of the keys generated within it. Also, the secure usage of these keys has to be handled correctly by the hardware. Additionally, users trust that the authentication for using the keys is handled correctly by the operating system and the device driver of the secure hardware.

This section stated all trust assumptions which are fundamental to the system and how these assumptions are preserved.

5 Evaluation of PRE Schemes

For the system we proposed in Section 4, we need an underlying cryptosystem, namely MUPRE. For information regarding MUPRE, have a look at Section 3.2. This section will give details on the requirements of the cryptographic building blocks that have to be met to build our multi-user data sharing system. Furthermore, we will investigate which proposed schemes meet all our requirements and finally evaluate which scheme is best suited.

5.1 Properties of Proxy Re-Encryption Schemes

This section summarizes some of the main properties of MUPRE schemes and the requirements regarding these properties, which are essential for the evaluation process of PRE schemes to be used in the proposed system.

Type: Existing PRE schemes can be further categorized in certain types. One such type is e.g., identity-based PRE. In contrast to conventional public-key cryptography, ID-based cryptography (IDBC) is a type of public-key cryptography, where public keys are known string representations of some identity. A prominent example being the email address of some person or organization. Another example for types of PRE schemes, is conditional PRE proposed by Weng et al., 2009. In conditional PRE, a proxy can only re-encrypt a ciphertext if certain restrictions are met, which are set by the delegator.

Nevertheless, as the proposed system does not depend on additional cryptographic properties, such as the examples in the above paragraph, we focus on classical PRE, which is precisely the setting, as described in Section 3.2.

Direction: According to Ivan and Dodis, 2003, PRE schemes can have two directions. Firstly, a scheme is said to be *bidirectional* if a re-key can be used to create l-th level ciphertexts under both of the key pairs used to generate the re-key. To clarify this, let us say Alice wants to share her data with Bob. She has to create a re-key and re-encrypt her ciphertexts for Bob's key pair. If the proxy can use the very same re-key to re-encrypt Bob's data for Alice, then this scheme is bidirectional. On the contrary, if only one direction is possible, so, if the re-key can only be used

by the proxy to re-encrypt Alice's data but not Bob's, then the scheme is said to have the *unidirectional* property.

In their original paper, Blaze et al., 1998 also specified those properties but in a less rigorous way and called them differently. On the one hand, they called bidirectional schemes *symmetric* because of the symmetric trust implications of such schemes (if A trusts B, then B also has to trust A). On the other hand, unidirectional schemes, where a trust assumption does not imply the respective complement, are called *asymmetric*.

For the proposed system, it is necessary to have a **unidirectional** scheme. In most systems, it is desired to have trust relationships like the ones provided by unidirectional schemes. To give a concrete example, imagine the use case defined in the previous chapter, where user Alice wants to restore her data with the help of her trusted user Bob. Alice had to create a re-key before the restoration process and persistently store it at the proxy's backend. Now, the proxy would have the ability to re-encrypt Bob's data and give Alice access to Bob's data, so every trusted user would sacrifice their privacy to restore another user's data. This restriction is not desirable.

Moreover, if there is a bidirectional use case, it is also possible to achieve bidirectionality with a unidirectional scheme. Let us again imagine the setting defined in the last paragraph. If Alice creates a re-key in a way such that Bob becomes Alice's trusted user, then Bob can recover Alice's data, but Alice does not get access to Bob's data. Furthermore, if Bob wants Alice to read his data, he can create another unidirectional re-key for Alice. This method has the same effect as if the scheme would be bidirectional with the additional restriction that Bob is not forced to reveal his documents, but instead has to give his explicit consent.

Non-Interactive: Originally introduced by Blaze et al., 1998 as *active* and *passive* Proxy Re-Encryption, the non-interactive property clarifies the input to the re-key generation. For interactive (active) schemes, the input to the re-key generation function, as defined in Section 3.2, are the private keys of Alice and Bob, meaning that if Bob wants to share his data with Alice, Alice is forced to actively get involved in the creation process, as she is the only one that can provide her private key.

Non-interactive (passive) schemes do not require Alice to participate in the re-key creation process. Here the re-key generation function depends on the private key of Bob and the public key of Alice. Nowadays, as seen in Ateniese et al., 2006, the literature refers to passive schemes as *non-interactive*.

For our proposed multi-user data sharing system, it is a necessity for the chosen scheme to be **non-interactive**. The first problem which comes to mind if using an interactive scheme is the re-key generation. If let us say Alice wants to share her data with Bob (or register Bob as her trusted user for that matter), then Alice has to create a dedicated re-key. Bob has to send Alice his private key actively. Not only would this approach be bothersome for users as the receiving party has to participate in the key generation, but more importantly, Alice would get access to Bob's private key.

The reasons mentioned above lead to the unambiguous conclusion that the scheme has to be non-interactive.

Collusion-Safe: In most scenarios, when working with proxy re-encryption, the proxy is defined as semi-trusted. In other words, the proxy tries to obtain as much information from the delegator as possible, be it the encrypted data or even the private key.

Collusion-safeness, which is sometimes called master secret security, implies that even when one recipient and the proxy collude, they cannot recover another user's secret key.

For our proposed system, collusion-safeness is not a fundamental requirement, meaning that if a scheme does not have this property, it is not an exclusion criterion. Nevertheless, if there are multiple schemes to choose from, we may determine our final decision whether the schemes have the collusion-safeness property or not.

Basic Security Notions: Over the last paragraphs, we talked about properties that are exclusive to PRE schemes. In the upcoming section, we investigate the imperative need for some basic security notions. Also, other cryptography fields use such notions, but for our case, these notions have to be extended to take re-encryption capabilities into account.

The term security is somewhat ambiguous, and to evaluate cryptographic schemes of any kind, we need to clarify what we mean if we talk about security. Of course, it should not be possible to obtain information of the underlying plaintext given any ciphertext. Still, this definition is too vague, as it does not take, e.g., the capabilities or goals of attackers into account. Most of the upcoming explanations and clarifications are inspired by Bellare et al., 1998. Our primary focus lies on the different attack models from individual attackers. Furthermore, for all attack models, we

try to provide ciphertext indistinguishability, which ensures that attackers cannot distinguish pairs of ciphertext, although they know the corresponding plaintexts.

Regarding these attack models, three different baseline attacks have been introduced, namely the **Chosen-Plaintext Attack (CPA)**, the **Non-Adaptive Chosen-Ciphertext Attack (CCA1)**, and the **Adaptive Chosen-Ciphertext Attack (CCA2)**. The power of the attackers increases from CPA to CCA1 to CCA2; hence, if a scheme is secure against a more powerful attacker, it has a higher security notion. The attack, which is mitigated by CPA secure schemes, is called a chosen-plaintext attack. Both CCA1 and CCA2, mitigate a chosen-ciphertext attack. These notions are best explained by their respective games, where an attacker and a challenger are involved.

CPA: In this game scenario, the attackers gets access to a so-called encryption oracle. This oracle yields ciphertexts from given plaintexts, although the attackers do not get any information about the oracle (black box view). The attackers then create as many plaintext-ciphertext pairs as they like. After this setup phase is done, they choose two plaintexts where the corresponding ciphertexts are unknown and send them to the challenger. The challenger then encrypts these plaintexts, chooses a random bit, and presents the ciphertext to the attackers, depending on the selected bit. If the attackers cannot guess the chosen bit with a higher probability than fifty percent, then a scheme is said to be CPA secure.

CCA1: The CCA1 game is somewhat similar to the CPA game. In contrast to the CPA game, the attackers get access to a decryption oracle, which can decrypt any given ciphertext. The attackers may use this oracle as many times as they deem necessary. After that, the attackers again choose two plaintexts and send them to the challenger who randomly encrypts one of the plaintexts. The challenger provides the ciphertext to the attackers, whereby the attackers are no longer allowed to use the decryption oracle. A CCA1 secure scheme ensures the impossibility to deduce information about what plaintext the challenger chose in the previous step.

CCA2: In contrast to the attack model of CCA1, the attackers may also use the decryption oracle after obtaining the challenge ciphertext. Of course, it is not allowed to decrypt the challenge ciphertext with the oracle. This small change in the settings increases the difficulty in a not negligible way.

As for the proposed multi-user data sharing system, we need at least a scheme that provides CPA security, as there may be risks otherwise. Since we use public-key cryptography, a universal encryption oracle is naturally given.

For the more powerful chosen-ciphertext attack, attackers need not only the inherently accessible encryption oracle but also a decryption oracle. There are only two actors who perform decryptions in our setting, namely the users that decrypt their documents and users, which act as delegatee of proxy re-encryption rights. One can assume that users will not act as decryption oracles, either for internal or external attackers, which is especially true for the users, which decrypt only their respective ciphertexts.

Nevertheless, as the multi-user data sharing system is easily extendable or could rise in complexity over time, the security requirements may increase to some higher level, so, if feasible, a scheme with a higher security notion should be preferred. One such example would be if we also grant writing permission when users share their documents. This behavior could be exploited that users may act as decryption models for other users.

In conclusion, this section summarized the properties of MUPRE schemes and stated the requirements on these properties for the proposed key management system.

5.2 Evaluation Process

In this section, we will evaluate schemes for the usage in our proposed multi-user data sharing system against the requirements defined in Section 5.1. We will go into detail on how we selected the schemes, which we then further examined.

We started the evaluation process with a search for classical PRE schemes. The result was then further filtered for schemes, which provide the multi-use property. In the upcoming sections, we evaluate the remaining schemes in more detail. The results of our evaluation can be found in Table 3.

5.2.1 Initial MUPRE Schemes

In this section, we will discuss schemes that are neither based on lattices nor bilinear pairings. They utilize different hard problems for their security claims.

Blaze et al., 1998 introduce "Atomic Proxy Cryptography" for the first time. Alongside their definitions, which inspired academic research in the field of PRE, they also proposed the very first PRE scheme. Their scheme is fundamentally based on ElGamal and is **bidirectional**. Moreover, it is possible to create higher-level ciphertexts making this scheme **multi-use**. During the re-key generation process, the delegatee needs to get actively involved. Hence, the scheme is **interactive**. It is also not resistant to collusion attacks and, therefore, **not collusion-safe**. Blaze et al., 1998 state that their scheme is **CPA** secure. As the ElGamal based scheme from Blaze et al., 1998 is bidirectional and interactive, it is not suitable for the usage in our system.

Deng et al., 2008 propose a scheme that is based on the Computational Diffie-Hellman (CDH) problem and is provable **chosen-ciphertext secure** in the random oracle model. The authors claim that, since this scheme does not rely on bilinear pairings, which are inherently costly to compute, the scheme is comparatively efficient. During the re-key generation process, the delegator's and delegatee's private keys are needed. Thus, the scheme is **interactive** and not **collusion-safe**. Additionally, the re-key is **bidirectional** and the scheme is **CCA1** secure. In conclusion, this scheme does not meet the previously stated requirements.

Kirtane and Rangan, 2008 present a signcryption scheme that is suitable for PRE. Signcryption simultaneously signs and encrypts messages, ensuring non-repudiation, confidentiality, and integrity of plaintexts. They present a **single-hop** scheme which is **unidirectional** and **non-interactive** and based on RSA. Furthermore, they give a rigorous proof that their scheme, if using correctly chosen building blocks, is weakly **CCA2** secure. The authors also show an extension for their scheme, making it a **multi-use** scheme whereby the ciphertext grows linearly with re-encryptions. The scheme is also **collusion-safe**, so this scheme fulfills all identified requirements.

5.2.2 PRE based on Bilinear Pairings

This section will discuss and evaluate PRE schemes based on bilinear pairings regarding the identified requirements in the previous sections. For more information about the mathematical concept of bilinear pairings, read Section 3.1.

In contrast to lattice-based PRE schemes, which usually rely on the Learning-With-Errors (LWE) problem, schemes based on bilinear pairings are not post-quantum secure. What is more, the created higher level ciphertexts from schemes with the multi-use property usually grow linearly concerning the number of re-encryptions performed.

Shao et al., 2011 provide a **multi-use**, **unidirectional** PRE scheme, which is proven secure against chosen-ciphertext attacks, making it at least **CCA1** secure in the standard model. Furthermore, it is also resistant to collusion attacks (**collusion-safe**) and is **non-interactive**. Like many other schemes with the multi-use property, the ciphertexts' size grows linearly with re-encryptions. Nevertheless, this proposal is sufficient for the usage of our proposed multi-user data sharing system.

Cai and Liu, 2014 introduce an attack which they call "proxy bypass attack". Alongside the proxy bypass attack, the authors propose a new **multi-use** PRE scheme resistant against the attack. Additionally, the scheme is **non-interactive** and **unidirectional**. The scheme also is **collusion-safe** and is at least **CCA1** secure, featuring a linear growth of the ciphertexts. Since fulfilling all identified requirements, this scheme is a candidate for our system.

Lu et al., 2014 provide a **multi-use** PRE scheme with constant ciphertext length on re-encryptions. Furthermore, it is secure against chosen-ciphertext attacks, thus making it **CCA1** secure. The scheme is also **collusion-safe**. As the scheme is **interactive** and also **bidirectional**, it is not possible to use this scheme in our setting.

Shao et al., 2016 propose a **multi-use** PRE scheme, emphasizing the particular use case for "Cryptographic Cloud Storage". The scheme is **bidirectional**, **collusion-safe**, and secure against replay chosen-ciphertext attacks in the random oracle model (**CCA1**). As it is bidirectional, it is not suited to be used in our multi-user data sharing system, although it is worth mentioning that their proposed scheme has a constant ciphertext size. Furthermore, the scheme is **interactive**.

5.2.3 PRE based on Lattices

The security assumptions of the LWE problem are the basis of the discussed schemes, which is reducible from worst-case lattice-problems. The main benefit of such schemes is that they are conjectured to be ”post-quantum secure”, meaning they are immune to known post-quantum cryptanalysis.

Aono et al., 2013 propose a ”Key-Private Proxy Re-encryption under LWE”. Key-Private Proxy Re-encryption in this context means that it is impossible to deduce any information about the identity from the re-key of either the delegator or the delegatee. Additionally, they state that their scheme is **unidirectional** and also has the **multi-use** property. The scheme is **CPA** secure, and, with some modifications, it also implies **CCA1** security in the random oracle model (this is a weaker version of the CCA1 notion, which is said to be in the standard model). One problem when using this scheme is the re-key generation process. The re-key consists of information created by the delegatee’s private key, making the scheme **interactive**. Aono et al., 2013 propose that the delegatee may send the information derived from their private key via a secure channel making the re-key generation ”not interactive”, but unfortunately, this also breaks our identified requirements. Another problem with this scheme is the structure of the re-key. If the delegatee and the proxy collude, it seems possible to learn the delegator’s secret key, making it **not collusion-safe**. Due to the stated issues, we did not choose this scheme.

Nuñez et al., 2015 present a new Proxy Re-Encryption scheme called ”NTRU-ReEncrypt” which is based on the NTRU cryptosystem. Ownership belonging to Security Innovation, Inc, NTRU is an open-source cryptosystem developed in 1996 trying to provide a post-quantum secure alternative to Elliptic Curve Cryptography (ECC) and the RSA cryptosystem. The hardness assumption is based on the “Approximate close lattice vector problem”. More information can be found on the GitHub repository of the NTRU project.

NTRUReEncrypt is a rather efficient scheme and also has the **multi-use** property. Furthermore, under the hardness assumption of the LWE problem, it is also **CPA** secure. Unfortunately, the scheme is **bidirectional**, **interactive**, and **not collusion-safe**, thus breaking fundamental previously identified requirements.

Jiang et al., 2015 propose a "Lattice-based multi-use unidirectional proxy re-encryption" scheme. Like all discussed schemes in this section, it is also based on the hardness assumption of the LWE problem. The authors claim the scheme to be **multi-use**, **unidirectional**, and **non-interactive**. Furthermore, this scheme is resistant to collusion attacks and Jiang et al., 2015 provide a security proof, which shows that their scheme is **CPA** secure in the standard model. This scheme is, therefore, a candidate for our proposed system.

Phong et al., 2016 present two key-private schemes. As only one of them is a **multi-use** scheme, we leave the other one out of the evaluation process. Their proposed scheme is **CPA** secure and **unidirectional**. Nevertheless, we did not choose this scheme because it is **interactive** and **not collusion-safe**.

Fan and Liu, 2016 propose a single-hop scheme which is CCA1 secure and extend it to two different multiple **multi-use** schemes. One of the multi-use schemes is **CCA1** secure, whereas the other falls back to **CPA** security. This weaker security notion is due to the different definitions of the security games from the two proofs. Both schemes are **non-interactive** and **unidirectional**. The authors do not make any claims about collusion-safeness. Therefore, both schemes are candidates to be used in our system.

5.3 Chosen scheme

In this section, we finish the evaluation process of the schemes, given in sections 5.2.1, 5.2.2, and 5.2.3. We will compare all schemes which meet the requirements and choose one of them for our proposed system.

Two of the schemes that meet all requirements are only CPA secure, namely one of the two schemes proposed by Fan and Liu, 2016 and the scheme by Jiang et al., 2015. As all other schemes have a stronger security notion, we decided not to use these CPA schemes.

The scheme given by Kirtane and Rangan, 2008 is the only scheme that is CCA2 secure, although there are some restrictions on the power of attackers. There is one downside if we would choose to use their proposal. Their scheme is, in fact, a unidirectional scheme that is extensible into a multi-use scheme. The unidirectional re-key generation step involves the encryption of parts of the generated re-key.

	unidirectional	collusion-safe	non-interactive	CPA-security (P) CCA-security (C)
	initial schemes			
Blaze et al., 1998	✗	✗	✗	P
Deng et al., 2008	✗	✗	✗	C
Kirtane and Rangan, 2008	✓	✓	✓	C
	paring based schemes			
Shao et al., 2011	✓	✓	✓	C
Cai and Liu, 2014	✓	✓	✓	C
Lu et al., 2014	✗	✓	✗	C
Shao et al., 2016	✗	✓	✗	C
	lattice based schemes			
Aono et al., 2013	✓	✗	✗	C
Nuñez et al., 2015	✗	✗	✗	P
Jiang et al., 2015	✓	✓	✓	P
Phong et al., 2016	✓	✗	✗	P
Fan and Liu, 2016	✓	✓	✓	C/P

Table 3: The result of our detailed evaluation on some MUPRE schemes

If Alice wants to create a re-key to translate her ciphertexts to Bob's, she has to encrypt certain parts of the re-key with a distinct public key from Bob using some asynchronous encryption scheme. To get a fully functional multi-use scheme, they suggest not to use an asynchronous encryption scheme, but rather re-encrypt the parts of the re-key with another PRE algorithm. To still guarantee all security assumptions, this distinct PRE scheme also has to be at least weak CCA2 secure. As this would mean we would have to implement another PRE scheme, we did not choose this scheme as this overhead in contrast to the other schemes is too high.

Due to the method of elimination, only three schemes remain at this point videlicet the scheme proposed by Shao et al., 2011, the CCA1 secure scheme by Fan and Liu, 2016, and the one given by Cai and Liu, 2014. There is no apparent advantage when using one scheme over the other, hence the decision was more problematic than before. Finally, we chose the scheme from Cai and Liu, 2014, just out of the reason that this scheme is less complicated than the other and more comfortable to implement.

In the last section, we finished the evaluation process of the schemes, given in Section 5.2.1, 5.2.2, and 5.2.3. We compared all schemes which met the requirements and chose one of them for our proposed system.

6 Translation Process

The scheme from Cai and Liu, 2014 utilizes type 1 bilinear pairings. Over the upcoming paragraphs, we discuss how we translated the scheme from Cai and Liu, 2014 into a system that relies on practical type 3 pairings instead of more theoretic type 1 pairings. For more information regarding the different types of pairings, see Section 3.1. Furthermore, we will give the translation of their scheme, which uses type 3 pairings.

Type 1 pairings have some mathematical advantages regarding the creation of cryptographic primitives (e.g., the hashing into subgroups). The downside of such pairings is that they are infeasible hard to compute, thus reducing their adaptability in real-world implementations of cryptographic algorithms. Therefore, we rewrote the scheme of Cai and Liu, 2014. There are multiple ways to achieve this. We will discuss two of them, namely the trivial approach and our more sophisticated solution.

6.1 Trivial Approach

In this section, we discuss how to translate a scheme utilizing type 1 pairings into a type 3 scheme with a trivial algorithm. For mathematical notation, we denote a bilinear map of type 1 e_1 and a bilinear map of type 3 e_3, respectively. Furthermore, all groups are written multiplicatively.

Let us recall the definition of a bilinear pairing e on the groups G_1, G_2 and the target group G_T:

$$e : G_1 \times G_2 \rightarrow G_T$$

If $G_1 = G_2$, then we talk about type 1 pairings.

There is a set of public parameters for all PRE schemes that rely on type 1 bilinear pairings. In most cases, they consist of the generator g_1 of the group G_1 and n different elements $h_i \in G_1, i \leq n$. Sometimes there are more public parameters, such as cryptographic hash functions, but they are not relevant for the translation process.

As in type 3 pairings $G_1 \neq G_2$ and $g_1, h_i \notin G_2, \forall i \leq n$, it is no longer possible to evaluate the type 3 billinear map e_3, because its domain is $G_1 \times G_2$. To solve this problem it is possible to naively add an extra pairing operation for each existing pairing, leading to a doubling of the computed pairings. Furthermore,

every random element h_i has to have a representation in both groups, so also the elements used are doubled. Imagine a pairing e_1, two elements $p_1, p_2 \in G_1$ and an element in the target group $p_t \in G_T$:

$$e_1(p_1, p_2) = p_t$$

In order to translate this in such a way that it is possible to use the type 3 map e_3, there would be the need for two more elements $q_1, q_2 \in G_2$ and two target elements $q_{t1}, q_{t2} \in G_T$. Then the parings can be computed as

$$e_3(p_1, q_1) = q_{t1}$$
$$e_3(p_1, q_2) = q_{t2}$$

The translated scheme would then use the two elements q_{t1}, q_{t2} whenever the element p_t would be used in the original scheme.

To give a concrete example, imagine the following, lightweight "encryption" scheme. It is a shortened version of the scheme proposed by Cai and Liu, 2014 where everything was omitted but the key generation, encryption, and decryption operation, although also these functions are abbreviated.

Setup(): Having a type 1 bilinear pairing, we define the public parameters as (g, g_1, q, G, G_T, e), where g is the generator of G, g_1 is a random element in $G \setminus \{g\}$, G is defined over $\mathbb{Z}_q^*$ and e being a billinear map $e : G \times G \rightarrow G_T$

KeyGen(): $\rightarrow (pk, sk)$: Select $x \in_R \mathbb{Z}_q^*$, set $pk = g^x$ and $sk = x$.

Enc(pk, m)**:** $\rightarrow c$: Given $m \in G_T$, select $r \in_R \mathbb{Z}_q^*$, outputs a ciphertext $c = (c_1, c_2) = (g^r, m \cdot e(pk, g_1)^r)$.

Dec(sk, c)**:** $\rightarrow m$: Parse c as (c_1, c_2). If this does not work, return $\perp$. Otherwise, compute $m \leftarrow c_2 / e(c_1, g_1^{sk})$.

To translate this very basic encryption scheme into a scheme utilizing type 3 pairings, the public parameters have to be extended. Like we stated earlier, for every random element used, there has to be a representation in both G_1 and G_2. Of course, the generator of G_2 should also be part of the public parameters, so the Setup phase becomes:

Setup(): Having a type 3 bilinear pairing, we define the public parameters as $(g_1, g_2, p_1, p_2, q_1, q_2, G_1, G_2, G_T, e)$, where g_1 generates G_1, g_2 generates G_2, p_1 is a random element in $G_1 \setminus \{g_1\}$, p_2 is a random element in $G_2 \setminus \{g_2\}$, G_1 is defined over $\mathbb{Z}^*_{q_1}$, G_2 is defined over $\mathbb{Z}^*_{q_2}$ and e being a billinear map $e : G_1 \times G_2 \to G_T$.

The same is also true for the public key, meaning the key generation is altered in the following manner:

KeyGen(): $\to (pk, sk)$: Select $x_1 \in_R \mathbb{Z}^*_{q_1}$ and $x_2 \in_R \mathbb{Z}^*_{q_2}$, set $pk = (pk_1, pk_2) = (g_1^{x_1}, g_2^{x_2})$ and $sk = (sk_1, sk_2) = (x_1, x_2)$.

We recall that, as stated above, we have to add an extra pairing evaluation for every pairing computation in the type 1 scheme, so the Enc and Dec operation in the translated type 3 pairing are:

Enc(pk, m)**:** $\to c$: Given $m \in G_T$, select $r_1 \in_R \mathbb{Z}^*_{q_1}$ and $r_2 \in_R \mathbb{Z}^*_{q_2}$, outputs a ciphertext $c = (c_1, c_2, c_3) = \left(g_1^{r_1}, g_2^{r_2}, m \cdot \left(e(p_1, pk_2)^{r_2} + e(pk_1, p_2)^{r_1}\right)\right)$.

Dec(sk, c)**:** $\to m$: Parse c as (c_1, c_2, c_3). If this does not work, return $\perp$. Otherwise, compute $m \leftarrow c_3 / \left(e(c_1, p_2^{sk_1}) + e(p_1, c_2^{sk_2})\right)$.

We successfully translated the basic encryption scheme. Unfortunately, there are multiple shortcomings with this naive "algorithm". First of all, the complexity of the scheme rises significantly. Not only the evaluated pairings, which are inherently costly to compute, are doubled, but also the set of used parameters increases, finally leading to massive performance impact. Furthermore, we showed this approach with a reasonably small encryption scheme. When using this translation process on a fully-fledged scheme, it can become quite complicated.

To conclude, the trivial approach is well-suited for lightweight schemes because the algorithm used is easily applicable. On the other hand, the complexity rises significantly on more intricate schemes. Additionally, the performance impact is not negligible. Out of these reasons, we did not use this approach to translate the used scheme.

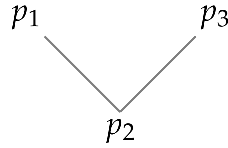


Figure 10: Basic dependency graph for the translation process

6.2 Advanced Approach

In contrast to the last section, this section will state a more sophisticated solution for translating a scheme utilizing type 1 pairings into a type 3 scheme. Notation for this section will be the same as in Section 6.1.

As the trivial approach, given in Section 6.1, had multiple problems, we investigate alternative ways of translating the scheme, aiming primarily at reducing the evaluated pairings alongside a reduction of the number of public parameters used. The algorithm's main idea is to examine which of the random elements h_i are used together to evaluate pairings. If there are elements that are always combined with a respective counterpart, then it would be possible to "move" these points to other groups, provided that the moving element does not break a different pairing, implying that some of the random points h_i would then be sampled from G_1 and some from G_2. Of course, alteration of the generator element of G_1 and G_2 is not possible.

The first step in achieving this is to build a dependency graph between all elements used in the scheme, where the vertices are the elements used, and the edges between them are the pairings. So, imagine the pairings

$$e_1(p_1, p_2) = p_{t_1}$$
$$e_1(p_2, p_3) = p_{t_2}$$

, then the graph would consist of three vertices and two edges, looking like Figure 10.

Of course, not all vertices need to be connected, so there may be multiple subgraphs. On a side note, in actual schemes, all exponents can be omitted as they are irrelevant for the evaluated pairing.

The next step is to determine if any of the graphs are non-cyclic. For every non-cyclic graph, there is at least one possibility to sample the elements from G_1 and G_2 in such a way that there is no necessity to add new pairings. To find such a

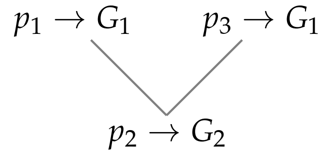


Figure 11: Labeled dependency graph for the translation process

possibility, start at any vertex of the uncyclic graph and chose whether this element belongs to G_1 or G_2. The choice for the first element is indifferent. All connected vertices should then be moved to the respective other group. Recursively go through the graph and label every element with the counterpart of its predecessor. The path through the graph does not matter, as long every vertex is visited. Applying this to the graph in Figure 10, starting at vertex p_1, which is moved to G_1, we get the graph in Figure 11.

So the translated equivalent pairing for type 3 would be

$$e_3(p_1, p_2) = p_{t_1}$$
$$e_3(p_3, p_2) = p_{t_2}$$

If there were no cyclic graphs, then the translation process would already be finished. There was no need to add additional pairings, and also no further elements had to be created.

Imagine again the parings used in the above paragraphs, but this time there is an additional computation

$$e_1(p_1, p_2) = p_{t_1}$$
$$e_1(p_2, p_3) = p_{t_2}$$
$$e_1(p_1, p_3) = p_{t_3}$$

The dependency graph would look like the graph in Figure 12.

A correct categorization of the vertices is not possible for this graph, at least not in the manner we stated earlier. Imagine we start by declaring the element p_1 as an element from G_1. Then there would be a contradiction later on, since both, p_2 and p_3 would belong to G_2, although there is an edge connecting these points, which renders it impossible. Unfortunately, there is no universal approach to solve this, except the trivial algorithm defined in 6.1, which can be used to translate the

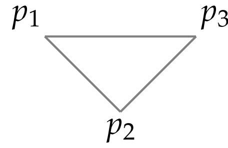


Figure 12: Cyclic dependency graph for the translation process

cyclic parts of the graphs. Sometimes it is also possible to find an alternative better solution for specific schemes, but this has to be checked individually.

For some type 1 schemes, this more sophisticated solution can be applied to translate them to type 3, where no additional costly pairing has to be added. What is more, also the needed public parameters can remain the same. The downside is the lack of a satisfactory solution for the problem when there is a cyclic dependency on certain elements. Regardless of this shortcoming, this approach is still more applicable than the trivial solution in Section 6.1.

To summarize, this section stated a more sophisticated solution on how to translate a scheme utilizing type 1 pairings into a type 3 scheme.

6.3 Translated Scheme

In the following paragraphs we will give the scheme from Cai and Liu, 2014 translated from type 1 to a scheme based on type 3 pairings, utilizing the translation process stated in Section 6.2.

Applying said algorithm on the MUPRE scheme provided by Cai and Liu, 2014, we found out that there is not a single cycle in the dependency graph, so there was no need to add additional pairings to the scheme. Regarding the public parameters, we added the generator g_2 of G_2 and substituted it with the random point $g_1 \in G_1$. We did this because it became apparent that the generator element g was the only element that was ever paired with other elements, so there was no mixing between the random elements $g_1, h_i : 0 < i \leq 3$. Also, the random point g_1 was only paired with g, so we choose that this random element would be sampled from G_2 and as the generator of G_2 was already added as a public parameter, we just set $g_1 = g_2$. This led to the following scheme:

Setup(1^k) $\rightarrow par$: Let 1^k be the security parameter, $(q_1, g_1, q_2, g_2, G_1, G_2, G_T, e)$, be generated by a bilinear group generator on input (1^k), $Sig = (\mathcal{G}, \mathcal{S}, \mathcal{V})$ be a strongly unforgeable signature scheme, and svk the signature verification key of the proxy. Let h_1, h_2 and h_3 be three random elements in G_2. Further, let $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_{q_2}^*$ and $H_2 : G_T \rightarrow G_2$ be two one-way, collision-resistant cryptographic hash functions. The public parameters are: $par = (q_1, g_1, q_2, g_2, h_1, h_2, h_3, G_1, G_2, G_T, e, Sig, svk, H_1, H_2)$.

KeyGen() $\rightarrow (sk, pk)$: Select $x \xleftarrow{R} \mathbb{Z}_{q_1}^*$, set $pk = g_1^x$ and $sk = x$.

Enc(pk, m) $\rightarrow C_{pk}^{(1)}$: Given $m \in G_T$, select $r \in_R \mathbb{Z}_{q_1}^*$, outputs a first-level ciphertext $C_{pk}^{(1)} = (c_{1,1}, c_{1,2}, c_{1,3}) = (g_1^r, m \cdot e(pk, g_2)^r, (h_1^{H_1(c_{1,1})} h_2^{H_1(c_{1,1}||c_{1,2})} h_3)^r)$.

ReKeyGen(sk_i, pk_j) $\rightarrow rk_{i\rightarrow j} (i \neq j)$: To generate a re-encryption key from pk_i to pk_j, do the following:

1. Select $r_{ij} \in_R \mathbb{Z}_{q_1}, K_{ij} \in_R G_T$.
2. Compute $R_1^{(ij)} = g_1^{r_{ij}}, R_2^{ij} = K_{ij} \cdot e(pk_j, g_2)^{r_{ij}}, R_3^{ij} = svk$,
   $R_4^{ij} = (h_1^{H_1(R_1^{ij})} h_2^{H_1(R_1^{ij}||R_2^{ij}||R_3^{ij})} h_3)^{r_{ij}}, R_5^{ij} = H_2(K_{ij}) \cdot g_2^{-sk_i}$
3. Output $rk_{i\rightarrow j} = (R_1^{ij}, R_2^{ij}, R_3^{ij}, R_4^{ij}, R_5^{ij})$. The re-encryption key is sent to the proxy via a secure channel.

ReEncrypt$(rk_{i\rightarrow j}, C_i^{(l)})$ $\rightarrow C_j^{(l+1)} (i \neq j, l \geq 1)$:

1. To re-encrypt a First-level ciphertext $C_i^{(l)}$, denoted by $C_i^{(1)}$, do:
   a) Parse $C_i^{(1)}$ as $(c_{1,1}, c_{1,2}, c_{1,3})$ and $rk_{i\rightarrow j}$ as $(R_1^{ij}, R_2^{ij}, R_3^{ij}, R_4^{ij}, R_5^{ij})$.
   b) Check if $e(g_1, c_{1,3}) = e(c_{1,1}, h_1^{H_1(c_{1,1})} h_2^{H_1(c_{1,1}||c_{1,2})} h_3)$ and
      $e(g_1, R_4^{ij}) = e(R_1^{ij}, (h_1^{H_1(R_1^{ij})} h_2^{H_1(R_1^{ij}||R_2^{ij}||R_3^{ij})} h_3))$ hold. If either of them fails, return $\perp$. Otherwise, do the following:
   c) Select $\mu_2 \in_R \mathbb{Z}_{q_1}^*, X_2 \in_R G_T$, compute $\chi_{2,1} = g_1^{\mu_2}, \chi_{2,2} = X_2 \cdot e(pk_j, g_2)^{\mu_2}$,
      $\chi_{2,3} = (h_1^{H_1(\chi_{2,1})} h_2^{H_1(\chi_{2,1}||\chi_{2,2})} h_3)^{\mu_2})$

d) Let $C = (c'_{1,1}, c'_{1,2}, c_{2,1}, c_{2,2}, c_{2,3}, c_{2,4}, \chi_{2,1}, \chi_{2,2}, \chi_{2,3})$, where $c'_{1,1} = c_{1,1}$, $c'_{1,2} = c_{1,2} \cdot e(c_{1,1}, H_2(X_2) \cdot R_5^{ij}), c_{2,1} = R_1^{(ij)}, c_{2,2} = R_2^{(ij)}, c_{2,3} = R_3^{(ij)}, c_{2,4} = R_4^{ij}$.

e) Let ssk be the signing key of the proxy corresponding to the verification key R_3^{ij}.

f) Run the signing algorithm $S(ssk, C)$, to generate a signature on the ciphertext tuple $(c'_{1,1}, c'_{1,2}, c_{2,1}, c_{2,2}, c_{2,3}, c_{2,4}, \chi_{2,1}, \chi_{2,2})$, and denote the signature as S_2.

g) Output the ciphertext $C_j^{(2)} = (C, S_2)$

2. To re-encrypt a lth-level ciphertext $C_i^{(l)}$, where $l > 1$, do:

a) Parse $C_i^{(l)}$ as $(c'_{1,1}, c'_{1,2}, c_{2,1}, c_{2,2}, c_{2,3}, \chi_{2,1}, \chi_{2,2}, S_2, \cdots, c_{l,1}, c_{l,2}, c_{l,3}, c_{l,4}, \chi_{l,1}, \chi_{l,2}, \chi_{l,3}, S_l)$ and $rk_{i \to j}$ as $(R_1^{ij}, R_2^{ij}, R_3^{ij}, R_4^{ij}, R_5^{ij})$.

b) Check if $e(g_1, c_{l,4}) = e(c_{l,1}, h_1^{H_1(c_{l,1})} h_2^{H_1(c_{l,1}||c_{l,2}||c_{l,3})} h_3), e(g_1, \chi_{l,3}) = e(\chi_{l,1}, h_1^{H_1(\chi_{l,1})} h_2^{H_1(\chi_{l,1}||\chi_{l,2})} h_3)$ and $e(g_1, R_4^{(ij)}) = e(R_1^{(ij)}, h_1^{H_1(R_1^{(ij)})} h_2^{H_1(R_1^{(ij)}||R_2^{(ij)}||R_3^{(ij)})} h_3)$ hold. If either of them fails, return $\perp$. Otherwise, do the following:

c) $\forall k \in [2, l]$, check $V(c_{k,3}, S_k, (c'_{1,1}, \cdots, c_{k,1}, c_{k,2}, c_{k,3}, \chi_{k,1}, \chi_{k,2})) = 1$. Whenever one of them fails, return $\perp$. Otherwise, do the following:

d) Compute $c'_{l,2} = c_{l,2} \cdot e(c_{l,1}, R_5^{(ij)}), c_{l+1,1} = R_1^{(ij)}, c_{l+1,2} = R_2^{(ij)}, c_{l+1,3} = R_3^{(ij)}, c_{l+1,4} = R_4^{(ij)}$.

e) Select $\mu_{l+1} \in_R \mathbb{Z}_{q_1}^*, X_{l+1} \in_R G_T$, let $\chi_{l+1,1} = g_1^{\mu_{l+1}}, \chi_{l+1,2} = X_{l+1} \cdot e(pk_j, g_2)^{\mu_{l+1}}, \chi_{l+1,3} = (h_1^{H_1(\chi_{l+1,1})} h_2^{H_1(\chi_{l+1,1}||\chi_{l+1,2})} h_3)^{\mu_{l+1}}$, compute $\chi'_{l,1} = \chi_{l,1}, \chi'_{l,2} = \chi_{l,2} \cdot e(\chi_{l,1}, H_2(X_{l+1}) \cdot R_5^{(ij)})$.

f) All other elements remain unchanged, let $C = (c'_{1,1}, \cdots, c'_{l,1}, c'_{l,2}, c'_{l,3}, \chi'_{l,1}, \chi'_{l,2}, S_l, c_{l+1,1}, c_{l+1,2}, c_{l+1,3}, c_{l+1,4}, \chi_{l+1,1}, \chi_{l+1,2}, \chi_{l+1,3})$.

g) Let ssk be the signing key of the proxy corresponding to the verification key R_3^{ij}. Run the signing algorithm $S(ssk, C)$ to generate a signature on the ciphertext and denote the signature as S_{l+1}.

h) Output the ciphertext $C_j^{l+1} = (C, S_{l+1})$.

Dec$(sk_i, C_i^{(l)}) \quad \to m$:

1. To decrypt a First-level ciphertext $C_i^{(l)}$, denoted by $C_i^{(1)}$, do:

a) Parse $C_i^{(1)}$ as $(c_{1,1}, c_{1,2}, c_{1,3})$. If this does not work, return $\perp$. Otherwise, continue the following process:
b) Verify that $e(g_1, c_{1,3}) = e(c_{1,1}, h_1^{H_1(c_{1,1})} h_2^{H_1(c_{1,1}||c_{1,2})} h_3)$. If not, return $\perp$.
c) Otherwise, compute $m \leftarrow c_{1,2}/e(c_{1,1}, g_2^{sk_i})$.
d) Output m.

2. To decrypt a lth-level ciphertext $C_i^{(l)}$, where $l > 1$, do:

a) Parse $C_i^{(l)}$ as $(c'_{1,1}, c'_{1,2}, c_{2,1}, c_{2,2}, c_{2,3}, \chi_{2,1}, \chi_{2,2}, S_2, \cdots, c_{l,1}, c_{l,2}, c_{l,3}, c_{l,4}, \chi_{l,1}, \chi_{l,2}, \chi_{l,3}, S_l)$. If this does not work, return $\perp$. Otherwise, continue the following process:
b) Check if $e(g_1, c_{l,4}) = e(c_{l,1}, h_1^{H_1(c_{l,1})} h_2^{H_1(c_{l,1}||c_{l,2}||c_{l,3})} h_3)$. If not, return $\perp$.
c) $\forall k \in [2, l]$, check $V(c_{k,3}, S_k, (c'_{1,1}, \cdots, c_{k,1}, c_{k_2}, c_{k_3}, \chi_{k,1}, \chi_{k,2}) = 1$. Whenever one of them fails, return $\perp$. Otherwise, do the following:
d) Compute $K_{l-1} \leftarrow c_{l,2}/e(c_{l,1}, g_2^{sk_i}), X_{l-1} \leftarrow \chi_{l,2}/e(\chi_{l,1}, g_2^{sk_i})$.
e) For a from $l-2$ down to 1, compute $K_a \leftarrow C_{a+1,2}/e(c_{a+1,1}, H_2(K_{a+1})), X_a \leftarrow \chi_{a+1,2}/e(\chi_{a+1,1}, H_2(X_{a+1})) \cdot H_2(K_{a+1}))$
f) Compute $m \leftarrow c_{1,2}/e(c_{1,1}, H_2(X_1) \cdot H_2(K_1))$.
g) Output m.

In the above paragraphs, we provided the scheme from Cai and Liu, 2014 translated from type 1 to a scheme based on type 3 pairings, utilizing the translation process stated in Section 6.2.

7 Implementation of Our System

This section will discuss the technical aspects of our proof of concept implementation of the multi-user cloud sharing system. The first part will state all actors and components in the system. The second part will be a discussion about the workflow and the user-experience of the system. We conclude this section with a performance evaluation and focus on how we could improve the system's performance. Furthermore, we will address how we solved the key challenges defined in Section 4.1.

7.1 Actors and Components

This section will list all components which make up our proof of concept implementation and which technology was used to implement them. Figure 13 illustrates all the separate components and how they interact with each other.

7.1.1 MUPRE Library

The system's central component is the Java implementation of the translated scheme proposed by Cai and Liu, 2014. To implement the scheme (c.f. Section 6) we used the ECCelerate library. This library offers the usage of certain elliptic-curve operations for the Java platform, including, among others, pairing-based cryptography.

Public Parameters The chosen MUPRE scheme defines some global public parameters. First of all, the underlying mathematical structures are Barreto-Naehring Curves (see Kasamatsu et al., 2014). Furthermore, there are three randomly sampled elements from G_2. The library statically initializes these elements on startup.

The scheme defines that the proxy that performs a re-encryption has to sign the corresponding resulting l-th ciphertext. The scheme does not specify the used signature algorithm, so we decided to use ECDSA. As for now, the used signature algorithm is not configurable. When the proxy performs a re-encryption, it has to provide a private key to sign the ciphertext. Additionally, during the decryption operation, the corresponding public key has to be provided. If the signature validation fails for any reason, the decryption operation terminates and outputs an error message.

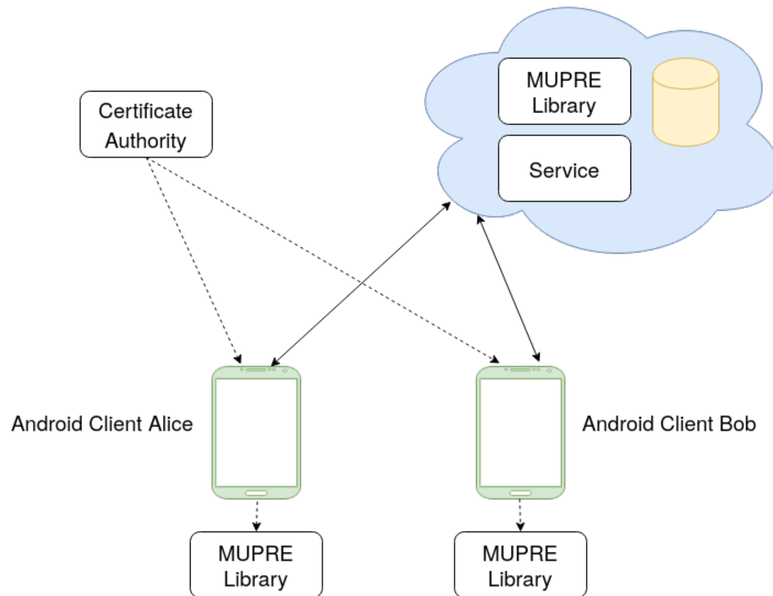


Figure 13: Components of our system

Operations We implemented every operation defined by the scheme. Listing 1 shows how to obtain a fresh MUPRE key pair and then how to encrypt some message with the newly created public key.

```
KeyPair keyPair = keyPairGenerator.generateKeyPair();
MUPREPublicKey pk = keyPair.getPublic();
MUPREPrivateKey sk = keyPair.getPrivate();

Message m = new Message();

MUPRECipher cipherEnc = new MUPRECipher();
cipherEnc.init(PRECipher.ENCRYPT_MODE, pk);
byte[] ciphertext = cipherEnc.doFinal(m.toByteArray());
```

Listing 1: Creation of a MUPRE key pair and encrypting a random message

Furthermore, every relevant object, like keys and ciphertexts, is serializable. The serialization is especially crucial for the creation of re-encryption keys. Listing 2 shows the deserialization of a distinct public key, which is then, in turn, used for creating a re-encryption key.

```
MUPREPrivateKey sk = keyPair.getPrivate();
```

```java
//restore BASE64 encoded public key
MUPREPublicKey pkRestored = new MUPREPublicKey(Base64.decode(
    encodedPK));

//svk is the signing key of the proxy that uses this re-key
MUPREReEncKeyGenerator.generateKey(sk, pkRestored, svk);
```

Listing 2: Deserialize a distinct public key, which is then used to create a re-key

The interface for the decryption operation is the same for first-level ciphertexts and higher-level ciphertexts. The decryption operation decides on its own whether to perform higher-level decryptions or not. The caller has to initialize the decryption object with a private key to perform any decryption, which can be seen in Listing 3.

```java
MUPREPrivateKey sk = keyPair.getPrivate();
byte[] ciphertext = //obtain ciphertext

MUPRECipher cipherDec = new MUPRECipher();
cipherDec.init(PRECipher.DECRYPT_MODE, sk);
Message m = cipherDec.doFinal(ciphertext);
```

Listing 3: Decrypt a l-th level ciphertext

7.1.2 Android Application

The component that is used by users of the system is implemented as an Android application. We decided on an Android application because it provides all necessary features we defined as compulsory, including, among others, the hardware-backed storage of cryptographic keys alongside hardware-backed cryptographic computations. Furthermore, as we implemented the MUPRE library for the Java platform, it is easily integrable into an Android application. The user interface of the application is seen in Section 7.2.

The Android client can be used as PD and as SD. It provides the functionality to upload documents, to download documents for which the client has the correct access rights, and to share documents with distinct users. If the device was registered as PD, it is also able to authorize the registration of SDs.

Secure Storage of key material For the handling of the cryptographic operations, we utilized the built-in "Android Key Store" [1] system. If the device is equipped with trusted hardware (like a TEE), then the Android Key Store system allows the communication with this hardware component. Android supports a predefined list of algorithms that can be used with the Android Key Store system. Unfortunately, the scheme proposed by Cai and Liu, 2014 is not among them. Therefore, we adopted a hybrid approach for the storage of the MUPRE private key created on the device.

Alongside the creation of the MUPRE key pair, an AES key is generated inside the device's hardware. As this symmetric key is created within the hardware of the device, it is not extractable. We then encrypt the MUPRE private key with the AES key, with GCM for authenticated encryption. The encrypted private key is then stored on the device. For all operations that depend on the MUPRE private key, e.g., the creation of re-keys or encrypting documents, the key is decrypted inside the device's hardware. There are some downsides with this approach. For example, during the key usage, the plaintext of the key resides in the RAM of the device. Unfortunately, as long as there is no native support from Android for MUPRE algorithms, we are limited to this approach.

Nevertheless, as the encrypting AES key is bound to the device's hardware, the MUPRE private key is also bound to the device. When using the Android Key Store, only the application which also created a hardware-backed key can perform cryptographic operations with it. Therefore, even when the device is corrupted, it is impossible to decrypt the MUPRE key, which is stored somewhere on the device.

7.1.3 Cloud Service

This component combines two different roles. Firstly, it acts as the cloud storage, which is the back end of the system. Secondly, it takes the proxy's role in the MUPRE scheme. The cloud service is initiated with Java Spring 5, a framework based on Java to create web services. The rationale behind the usage of Java Spring is our proficiency with this framework, and also the fact that we can easily integrate our implemented MUPRE library. With the help of Java Spring, we created a REST API, which can communicate with the Android application described in the last section.

[1] Further information about the Android Key Store system can be found by following this link

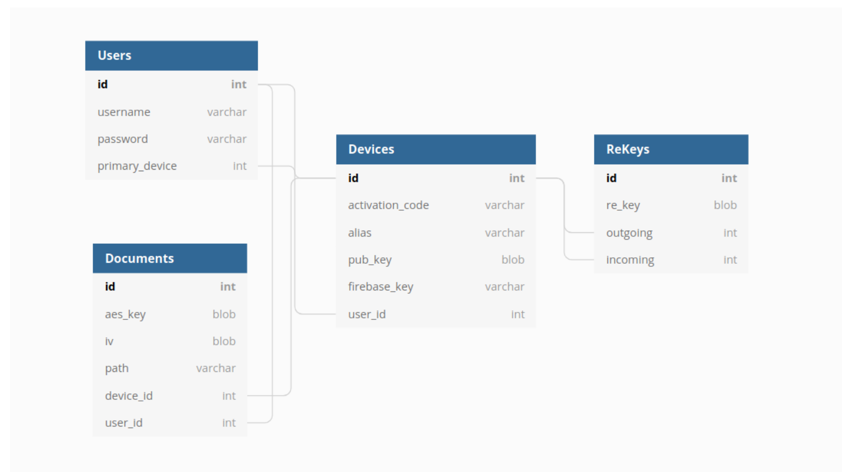


Figure 14: Database definition of the cloud service

One of the central aspects of the cloud service is the user management and in turn the device management. During the registration of the PD, an internal representation of the user is created, alongside a representation of the PD. Every newly registered device is also mapped in the backend. These representations are persistently stored in a relational database. The illustration of the database can found in Figure 14. A user can log in with every device at the same time.

Furthermore, the cloud service allocates some space for the documents of its users. If a user uploads a document, the cloud service stores the encrypted document in this allocated space, and creates a database entry for this file. This entry contains certain meta information for later usage, e.g., the user that uploaded the document. With this information, the cloud service is also able to perform access control. It only serves documents to users if they have sufficient access rights. This can either be the case if the user themself wants to access their documents or if an owner of the document bestowed access rights to the requesting user.

Access rights are modeled as a directed graph of re-encryption keys. As seen in Figure 14, there exists a database table for this purpose. The graph's vertices are devices, and the edges are registered re-encryption keys, which were uploaded beforehand by users. If a user requests a document, the cloud service searches for a path within this directed graph. The path starts from the device which uploaded the document to the device which requests the document. If such a path exists, the cloud service checks whether the document has to be re-encrypted or not, and then

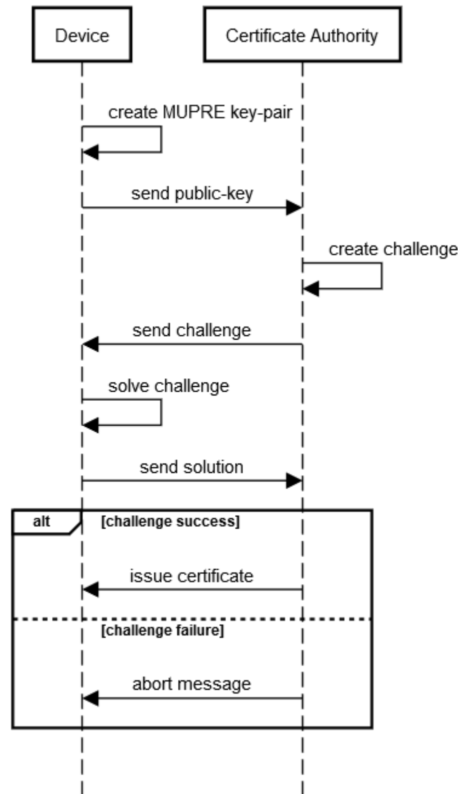


Figure 15: Messages sent for certification of the MUPRE public key

serves the (re)encrypted document. In the other case (no path exists), the cloud service sends an error message.

7.1.4 MUPRE Certificate Authority

The last component of our proof of concept implementation is a CA, which issues certificates over MUPRE public keys. This CA acts as a root of trust within our system and enables some points discussed in Section 4.5. We used the same framework for this component and the cloud service described in the last section, namely Java Spring 5. The cloud service provides a REST API to request the certification of MUPRE public keys. Furthermore, it manages a persistent database

to keep track of all issued certificates.

Every time a device is registered, it has to create a MUPRE key pair. After creating the key pair, the device requests the CA to certify the public key. To prove that a device owns the corresponding private key, we implemented a challenge-response protocol. All messages sent within this protocol can be found in Figure 15. This diagram shows a top-level view of all messages that are sent between a device that wants to certification for its MUPRE public key, and the MUPRE CA.

7.2 Process

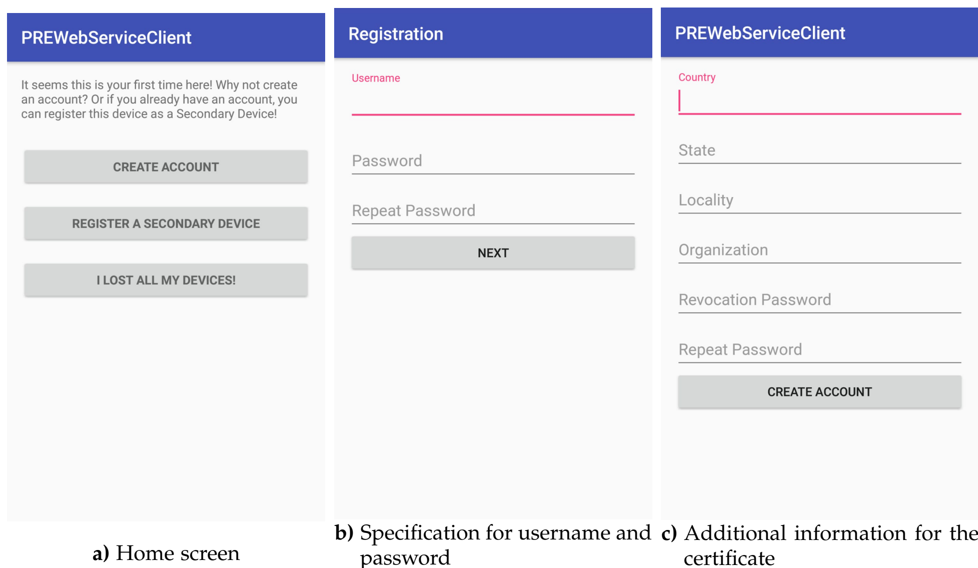
In this section, we will describe the typical workflow from the point of view of a user. Furthermore, we will address the user experience by discussing the Android application's user interface that serves as a client for the system. The section is organized as follows: The first part shows the steps a user has to take to be registered at the system. The second part and the third part will summarize how the upload and sharing of documents are handled. The section concludes with a discussion on the download of documents.

7.2.1 Registration of a PD

The first step for every user is the registration at the service (as seen in Section 4.2). When opening the Android client, the shown interface depends on whether the device is already registered at the system or not. If the user did not already register the device, the home screen is rendered, as seen in Figure 16a. This window is the starting point for multiple actions. The user can create a new account, register an SD, and invoke the restoration process if the user has lost all their devices.

When clicking on "create account", the user is asked to input a unique username and a password, followed by a window where the user enters additional information incorporated in the certificate. All of the windows can be seen in Figure 16.

After the user inputs all their information, the device creates a fresh MUPRE key pair. More information on the key-generation can be found in Section 7.1.2. This fresh key pair has to be certified by the root CA in the system. The application communicates with said CA and performs the necessary steps for certification, as described in Section 7.1.4.



a) Home screen

b) Specification for username and password

c) Additional information for the certificate

Figure 16: Registration Process

After the public key's successful certification, the client transmits all necessary information to the cloud service. The cloud service performs some checks on the request. Among others, it verifies whether the CA issues the certificate of the public key. Some space at the server is allocated for the new user's documents, and all data from the request is persistently stored. The new device is uniquely identifiable by the public key's fingerprint and by a string identifier of the device. This device identifier is generated from the username and a device name. For the first PD, the device name is set to "PD". For example, Alice's device after the registration is called "AlicePD". This device name is stored at the cloud service and the device.

If the registration process was completed, the user is greeted with a welcome message and the possibility to login by providing their password, as seen in Figure 17. It is not necessary to give the username as the device is bound to a specific user.

7.2.2 Registration of an SD

The workflow for the registration of an SD is similar to the registration of a new user account. One of the differences for the user is that a device name has to be

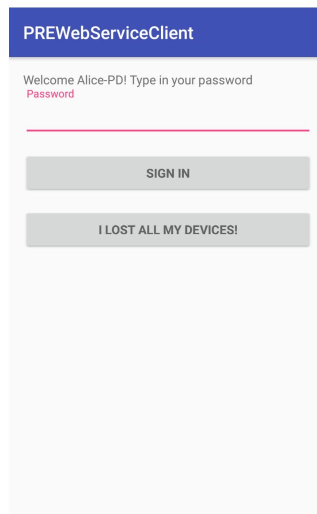
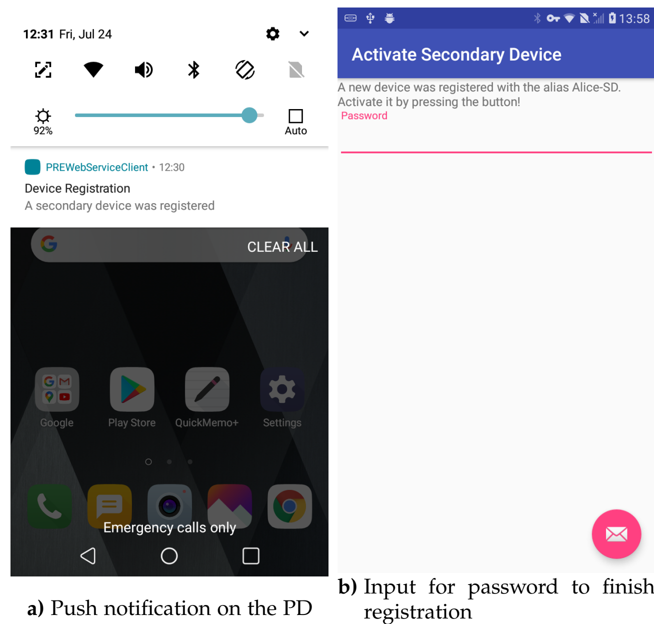


Figure 17: The welcome screen of the application after registration

provided. Furthermore, after the CA issued the certificate for the key pair of the SD, and the key is sent to the cloud service, the user has to authenticate the SD. This authentication is done by the PD, which is notified by a Push-Notification, which can be seen in Figure 18a. The Push-Notification transmits the device name of the SD, and the user has to re-enter their password to complete the authentication, as seen in Figure 18b. The user could discard the registration of the SD if they did not want to register a new device. This leads to a two-factor authentication, where it is necessary to know the password of a user, and also have physical access to their PD to register an SD.

7.2.3 Handling of Documents

In this section, we will summarize the upload and download of documents, including the cryptographic steps which have to be performed. For this proof of concept implementation, we decided to focus on the sharing of text documents. Every step is easily extensible for arbitrary octet streams of data, but for the sake of simplicity, we limited our scope. This specific part of the Android application can be seen in Figure 19.



a) Push notification on the PD

b) Input for password to finish registration

Figure 18: User Interface for the registration of an SD

List After a successful login, the application requests the names of all documents that the device has access to. This includes, of course, all documents belonging to the user, but also the documents from distinct users that shared their data. How the cloud service determines the access rights can be found in Section 7.1.3. When the response from the cloud service is parsed, the application renders the interface, which can be seen in Figure 19a. This window consists of a list of documents that can be downloaded. In the given example, Alice has access to four documents. Three of those documents belong to her, and one is owned by Bob, which shared the document with her, indicated by Bob's username in front of the document's name. From this window, the user can invoke multiple processes. The user can, for example, delete documents where they have the correct rights or could download any of these documents.

Upload Furthermore, it is possible to upload new text documents to the cloud service. Creating and uploading a document is done by clicking the floating action button, as shown in Figure 19. For simplicity, a prompt is opened where the user can specify a document name and input the document's content. Figure 19b shows the prompt. After the user enters the necessary information, the document

is encrypted and uploaded. The encryption process itself is a hybrid encryption approach, where the asymmetric part is the rewritten scheme given in Section 6.3, and for the symmetric part, we utilize AES in GCM. For every document, the device creates an AES key and an Initialization Vector (IV) and encrypts the document. The AES key is derived from a randomly sampled element $P \in_R G_T$. The point P is then encrypted with the MUPRE scheme, and all resulting ciphertexts and the IV are sent to the cloud service where they are persistently stored.

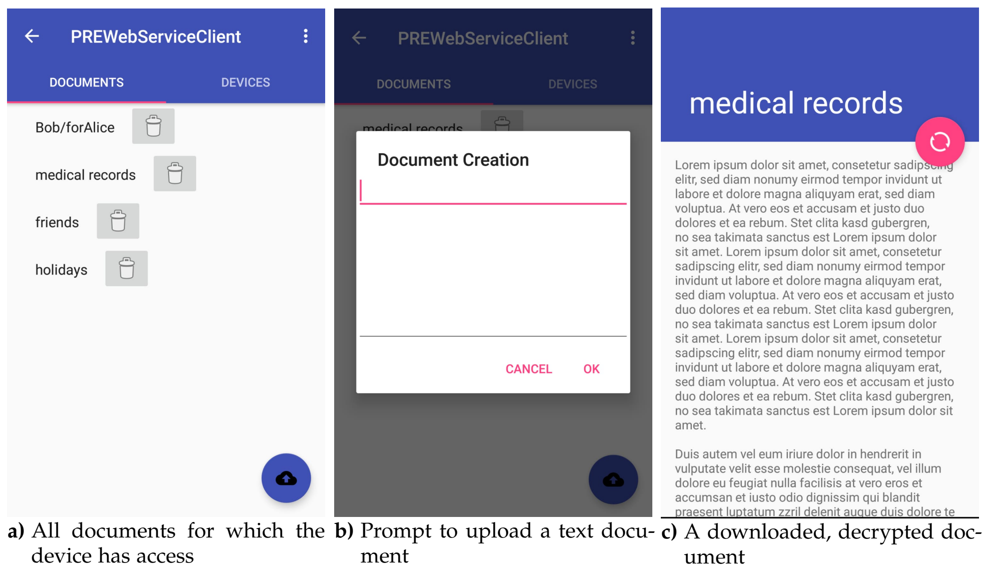
Download If the user wants to download a document, they have to click on the name of the document. The Android application requests all information needed from the cloud service. After the cloud service validates the request, it transmits the AES ciphertext of the document and the MUPRE ciphertext of the point P. Furthermore, the cloud service may perform some re-encryptions, depending on which device the document was originally encrypted. With the embedded MUPRE private key on the device, it can obtain the point P and derives the AES key, which can be finally used to decrypt the document. After a successful decryption, the document is rendered, as seen in Figure 19c.

Because the decryption and the re-encryption are expensive computations, the original download of a document can take some time, especially if multiple re-encryptions have to be performed. To counteract this problem, we implemented a mechanism to cache the downloaded documents, so only the first opening of a document will take some time. If a document is downloaded for the first time, it is encrypted with a separate AES key, which is embedded in the secure hardware of the device. It is possible to reload the document by clicking on the reload button, which can also be seen in Figure 19c.

7.2.4 Sharing of Documents

This section focuses on the sharing of documents and how to pair devices. To communicate between devices, we utilize the Push-Notification technology natively supported by Android. The user interface for this process can be examined in Figure 20.

Show When we have a look at Figure 20a, we see all pairings where the device is involved. In this example, Alice created two outgoing connections, and one incoming connection was created for her. For every outgoing connection, she



a) All documents for which the device has access **b)** Prompt to upload a text document **c)** A downloaded, decrypted document

Figure 19: Document handling

bestowed decryption rights to another device. The two devices for which outgoing re-encryption keys were registered are Alice's SD and Bob's PD. Of course, Alice's SD is able to read the data encrypted for her PD (see 4.2). Furthermore, Bob can read Alice's documents on all of his devices. Contrary, the incoming connection enables her to read Charlie's documents.

Create To create a new pairing, it is necessary to perform a pairing request to the targeted device where the public key of the requesting key is embedded, such that it is possible to invoke the re-key generation. The user has to click on the floating action button, which can be seen in Figure 20a to start this process. This action opens a prompt where the user is asked to enter the username of a distinct user. If the given username is registered at the service, the cloud service notifies the requested user that some user wants to get access to their documents. Imagine Bob wants to obtain decryption rights for Alice's data. After Bob enters Alice's username, Alice is notified that Bob requested read access, as seen in Figure 20c. Alice can either discard the request or accept it by entering her password. If she accepts it, her device creates a new re-key with her device's private key and Bob's public key, which was sent to her device via the Push-Notification. Figure 20d shows the interface for Alice to enter her password. The fresh re-key is then

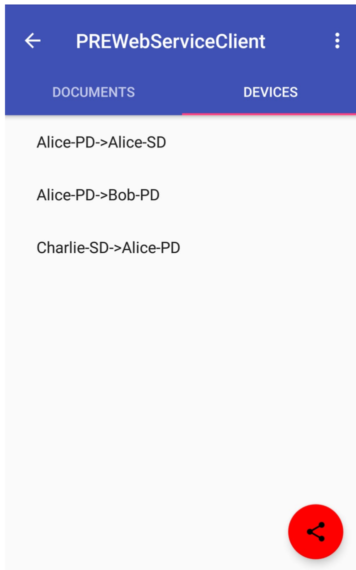
installed at the cloud service.

7.2.5 Recovery after Device-Loss

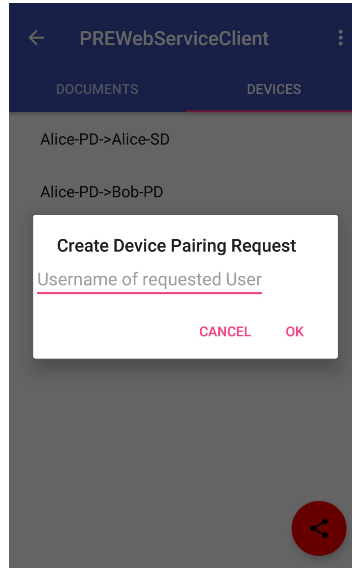
This section describes the user interface for recovery after a user lost all of their devices. Same as for the sharing of documents, we use Push-Notifications to communicate between devices. Figure 19 shows the user interface for this process.

If a user lost all their devices, they have to install the client on a new device. This device will become the new PD after the recovery process finishes successfully. The recovery process is invoked by clicking on the button "I lost all my devices!", which can be seen in Figure 16a. If, for example, Alice lost all her phones and started the restoration process, she will see the screen given in Figure 21a. There she has to input her username, her password, and the username of her trusted user Bob. Furthermore, as the device she is currently on is not registered, she has to specify a new alias.

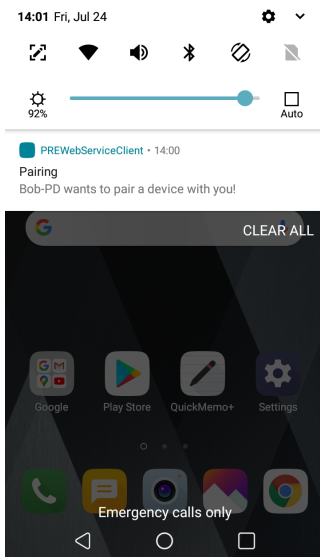
After the cloud service verifies the request, Bob receives a push notification, as seen in Figure 21b. If he clicks on the push notification, the screen in Figure 21c is opened on his device. If Bob is convinced that the request is coming from Alice, he inputs his password and invokes the recovery process described in Section 4.3.



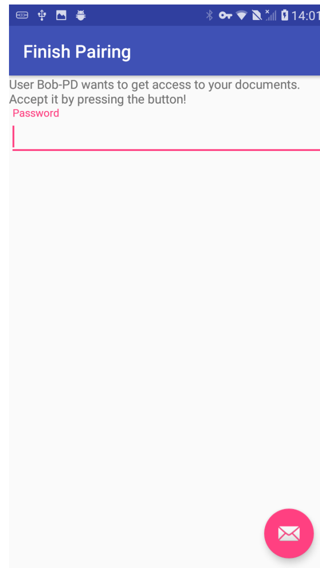
a) All pairings from this device



b) Prompt to send a pairing request to another user

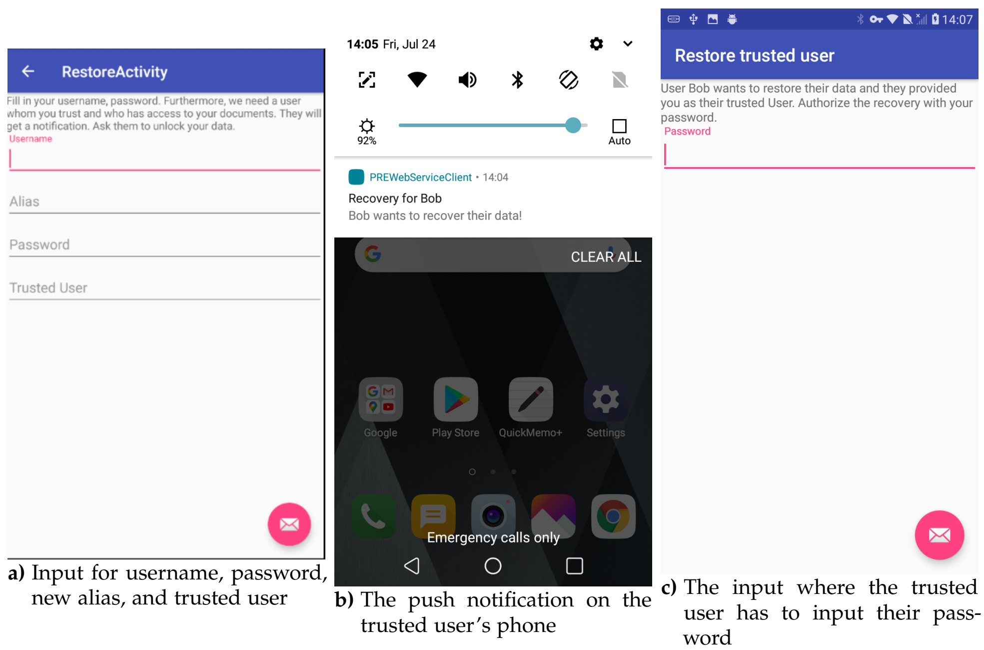


c) The push notification on the device of the other user



d) Input for the distinct user to enter password

Figure 20: User Interface for sharing of documents



a) Input for username, password, new alias, and trusted user

b) The push notification on the trusted user's phone

c) The input where the trusted user has to input their password

Figure 21: User Interface for recovery

7.3 Discussion

In this section, we state the results of our performance evaluation, followed by a discussion on how to improve the system's performance. We will further elaborate on how we met the key challenges of the system defined in earlier sections.

7.3.1 Performance Evaluation

During the development of our proof of concept implementation, we performed multiple benchmarks to assess the service's performance. We mainly focused on cryptographic operations because we suspected that these operations would have the biggest impact on the service regarding speed.

We started by evaluating the throughput of all operations of the MUPRE scheme (as seen in Definition 2), which are necessary for data sharing and recovery on different devices. Our results can be found in Table 4. In the referenced table, we listed the time it took on average (10000 computations) in milliseconds to perform said operations of the MUPRE scheme. This includes key generation, re-key generation, and encryption. Furthermore, we tested the re-encryption and decryption steps on ciphertexts up to the fifth level. We chose this limitation because the expected level of a ciphertext used in the system is at most 4. There may be some cases where the level will get higher than that, but this only happens if a user lost their device, and the system needs to recover access to the documents.

Table 4 shows two representative benchmark results for a PC, running with Ubuntu 18.04, and a mobile phone, namely a Google Pixel 2 with Android 8.0.0. Interestingly, when performing the benchmarks on mobile phones, there was a fall-off in nearly every operation. When performed on mobile devices, we expected that the operations are slower, but not by such a margin. Whereas for the PC, all operations take a reasonable amount of time regarding user experience. Even for the heavy computations as decryption and re-encryption, the same mobile phone operations perform significantly worse. The decryption and re-encryption time increases even linearly.

As the client for users is implemented as an Android application and the backend of the cloud service runs with Java Spring, some of the tested operations will only be performed on a mobile device and some only on a PC. The Android application will create keys, create re-keys, encrypt plaintexts, and decrypt l-th level ciphertexts, whereas the cloud service will only perform the re-encryption

level l	KeyGen	ReKeyGen	Enc	ReEnc $l \rightarrow l+1$	Dec
PC (Intel i5-2500@3,30GHz x4, 16Gb RAM)					
1	6.9	184	110	472	206
2	-	-	-	632	357
3	-	-	-	633	524
4	-	-	-	664	672
5	-	-	-	615	808
Mobile Phone (Google Pixel 2, Android 8.0.0)					
1	8.8	3622	2468	9740	4961
2	-	-	-	13501	7284
3	-	-	-	13656	10912
4	-	-	-	14082	14191
5	-	-	-	14532	17832

Table 4: Execution Times (in milliseconds) of the implemented MUPRE scheme

of ciphertexts. Key generation, re-key generation, and encryption are reasonably fast for our needs. However, re-encryptions are computed when users want to download their documents on different devices, which naturally implies that the decryption on the mobile phone is done during the same workflow. For users of the application, both of these operations are done sequentially during a single request they perform. The re-encryptions per se do not have that much of a negative impact on their own, as they are computed on a PC. Still, the decryptions on the mobile have a noticeable impact on the overall user-experience. In the upcoming paragraphs, we discuss how to address this problem and present some solutions to better the system's response time.

In subsequent work, we have investigated possibilities to improve performance. By re-implementing, the same translated MUPRE scheme based on RELIC [2] in C, we were able to improve the performance as reported in Hörandner and Nieddu, 2019 dramatically.

Re-Encryption Chain Length As seen in Table 4, the time for both, the re-encryption and decryption, rises directly proportional to the level of the ciphertext.

[2]RELIC is a framework that helps you build cryptographic libraries based on C. More information can be found on their Github page.

This implies that if we can limit or decrease the chain length of re-encryptions, there will be a positive impact on the system's performance. Recalling Protocol 1, there is a natural limitation on the chain length of re-encryptions. In this protocol, re-encryptions are only computed by the cloud service when a user wants to download documents on a device different than their PD. There are, in fact, three cases, 1) Alice wants to download data to her SD, 2) Bob wants to download data from Alice to his PD, and 3) Bob wants to download data from Alice to his SD. As there are no transitive trust assumptions, further re-encryptions are never done in this context. The maximum level of the created ciphertexts for the three different cases are 2, 3, and 4, respectively, depending on whether Alice's SD is involved in the re-encryption chain or not. To summarize, when performing Protocol 1, the maximum chain length of re-encryptions and the maximum level of a ciphertext is 4.

It gets more complicated when performing recovery processes as defined in Protocol 2. On the one hand, the loss of an SD does not impact the level of the ciphertexts used. On the other hand, the loss of the PD increases the needed amount of re-encryptions if the download of the data is requested by a device, where the connection was created after the recovery process was performed. If, for example, Alice loses her PD but did register an SD, an additional re-encryption has to be performed every time a user, irrelevant if its Alice or some distinct user, wants to download one of Alice's documents with such a new connection. Furthermore, if Alice did not register an SD before her loss of the PD, the needed amount of re-encryptions increases by 2, in the same case. As such, device-loss recovery strategies are not performed regularly, the ciphertext levels will not get infinitely large. Still, even one recovery may impact the service's performance in a noticeable fashion.

Refreshing Ciphertext Level The maximal length of a re-encryption chain is limited to 4 re-encryptions, except a user lost their PD and performed one of the recovery strategies from device-loss discussed in Section 4.3. We defined two different types of connections that a lost device may have. The first ones are the outgoing keys, which are the keys where the user bestowed decryption rights to other devices, and the lost device acts as an intermediate step in a re-encryption. The other keys are called incoming keys. With these keys, it is possible to re-encrypt ciphertexts such that the ciphertext may get decrypted with the key pair of the lost device.

For all recovery strategies, we specified that the existing network of re-encryption

keys stays intact with the addition of new incoming connections to the device used for recovery. This approach implies that the re-encryption chain of documents that are downloaded with a device, where the connection was created after the device-loss, may exceed the natural limit of 4 re-encryptions, negatively impacting the system's performance.

To give an example, imagine Alice lost her PD and registered her old SD to be her new PD. If now Bob requests access to Alice's documents, then all ciphertexts have to be re-encrypted from her old PD to her new PD for Bob to be able to read Alice's data. This increases the level of the ciphertexts by one in contrast to the original levels defined in the previous paragraph, implying that there even may be a 5-th level ciphertext that has to be decrypted on any of Bob's SDs. The same is true for the case that Alice did not register an SD, with the difference, that the ciphertext level increases by two, yielding potential 6-th level ciphertexts. As Alice may lose her PD again, there is no limit on the growth of the ciphertext.

This problem can be tackled by refreshing the ciphertexts which were encrypted for the lost PD. During the recovery process, the user may be allowed to select some (or all) ciphertexts. These ciphertexts are re-uploaded to the cloud storage with respect to the new PD. The recovery process would be much more costly, as the defined documents stored at the cloud storage have to be served to the new PD, where they would be decrypted, then encrypted again (with a hybrid approach), and finally persistently stored in the cloud. The obvious advantage is that there would be fewer ciphertexts created at a higher level than 4.

Caching and Pre-Computing Ciphertexts Another possible solution to tackle the linearly growing time consumption of performing multiple re-encryptions sequently is to pre-compute re-encryptions. Instead of re-encrypting documents on-demand, the cloud service caches ciphertexts. For example, if a user registers an SD, the cloud service re-encrypts every document of the user, such that the resulting ciphertexts can be decrypted with the newly registered SD. This approach could also be extended for sharing data with distinct users. Every time a user shares their documents with a distinct user, the documents get re-encrypted for all devices of the requesting user. Also, after a successful recovery process, all ciphertexts would be pre-computed. The apparent advantage is that irrelevant which device requests documents, there is no need to perform any costly computations at the cloud service at the time of requesting the data.

Additionally, pre-computing could also be applied for decryption. Clients may download the documents for which they have access rights in a background task

and decrypt them without the user's knowledge and store them somewhere on the mobile device. The cached documents may or may not be symmetrically encrypted on the hard drive of the mobile phone.

The downside of pre-computing decryptions and especially re-encryptions is the additional computational effort. During almost all operations defined in Protocol 1 and 2, extra computations have to be performed that may never be used. What is more, pre-computing naturally requires more storage space on the cloud service. The same holds for caching decryptions.

7.3.2 Discussion of Challenges

In Section 4.1, we defined the key challenges our system faces, whereas, in this section, we will state how our proof of concept implementation tackled each of those challenges.

Confidentiality Of course, a major requirement of our system is the confidentiality of the user's data. To guarantee confidentiality, we utilized a hybrid-encryption approach for protecting sensitive data. For symmetric encryption we used state-of-the-art implementations of AES with GCM. The security guarantees of AES are more than well-investigated. Furthermore, for asymmetric encryption, we used a modified scheme proposed by Cai and Liu, 2014. For a discussion on necessary security implications of the MUPRE scheme, see Section 5. Every communication between devices and the cloud service utilizes TLS, and we also implemented key-authenticity in the system, as discussed in Section 4.5. All of those concepts combined ensure the requirements for the confidentiality of the system, as previously defined.

Integrity The integrity of the data is protected by the same concepts that warrant confidentiality, namely the hybrid encryption approach with AES in GCM and the integrated integrity protection of the scheme proposed by Cai and Liu, 2014. When operating AES with GCM, AES becomes an authenticated encryption scheme, where it is no longer possible for any party that is not authorized to alter or insert data undetected. Additionally, the scheme by Cai and Liu, 2014 also detects whether the ciphertext was compromised by any other party than the proxy that performed the corresponding re-encryption during decryption.

User Experience Next to the key concepts of information security is the user experience of the system. Our main goal regarding user experience was that the costly cryptographic operations should not negatively impact the system's usage. A discussion on the performance of the system can be found in Section 7.3.1.

Sharing of Documents Additionally to the confidentiality and integrity protection, the MUPRE scheme from Cai and Liu, 2014 also enables to share documents with distinct users. If a user wants to share the access to their documents, they have to create a re-key that the cloud storage can use to re-encrypt the data. To create such a re-key, it is necessary to have physical access to the PD of a user and know their password. This approach implies that the threat of unauthorized sharing of documents is negligible.

Recovery after Device-Loss Our system utilizes client-side storage of cryptographic key material. Nevertheless, it is possible for a user to recover the access to their documents if the user lost all their devices and, therefore, their key material. All recovery steps imply minimal trust decisions by the user. We address the recovery strategies in Section 4.3.

Users of our system shall have the possibility to recover the access to their documents after losing all their devices and, therefore, their key material. Additionally, this recovery process shall imply minimal trust decisions by the user that lost all their phones.

Protection of the Key Material As the key management of our system is one of the key concepts, we addressed the protection of the key material in Section 7.1.2. Using the "Android Key Store" system, we can ensure that all requirements on the integrity of the key material are met, in case that the Android device on which the application runs is not rooted and that the security guarantees of the Android Key Store system hold.

8 Conclusion

In this thesis, we proposed a cloud-based data storage system utilizing Multi-Use Proxy Re-Encryption. The proposed system enables users to register multiple devices, synchronize their data across these devices, and share data with distinct users confidentially and securely. Alongside the data storage, we introduced a key management system, allowing users to create and store their key material in their devices' secure hardware. With our key management system, users can recover access to their data after device-loss, including a recovery strategy when all devices of a user are lost.

As already mentioned, Proxy Re-Encryption was used to implement the sharing of the users' documents and is also the main building block for the recovery strategies. In our work, we evaluated multiple Multi-Use Proxy Re-Encryption schemes with respect to previously identified requirements in the context of our solution. We decided to use the scheme proposed by Cai and Liu, 2014 since it is a unidirectional, non-interactive scheme with collusion-safeness, and it provides CCA-security. Because this scheme was originally proposed with type 1 pairings, we also stated the process of how to translate schemes relying on bilinear pairings from type 1 into a scheme using type 3 schemes, as they are more practical to implement. We further applied our algorithm and transformed the chosen scheme into a type 3 scheme.

Additionally, we implemented a proof of concept solution to showcase our proposed system, alongside the translated MUPRE scheme. We created a fully functional backend infrastructure with Java Spring that enables potential users to register new devices, upload and securely store their documents, and share their documents with distinct users. Furthermore, we created an Android application that can communicate with the backend of the system and also utilizes the secure hardware of the underlying Android device to manage the keys generated on the device. Of course, the proof of concept implementation allows the user to recover access to their documents in case one or all devices of the user are lost. Finally, we evaluated the performance of the system with particular attention on the MUPRE scheme.

References

Yoshinori Aono, Xavier Boyen, Le Trieu Phong, and Lihua Wang (2013). "Key-Private Proxy Re-encryption under LWE." In: *Progress in Cryptology – INDOCRYPT 2013*. Ed. by Goutam Paul and Serge Vaudenay. Cham: Springer International Publishing, pp. 1–18. ISBN: 978-3-319-03515-4 (cit. on pp. 51, 53).

Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger (2006). "Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage." In: *ACM Transactions on Information and System Security (TISSEC)* 9.1, pp. 1–30 (cit. on p. 45).

Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu (2011). "Password-Protected Secret Sharing." In: *Proceedings of the 18th ACM conference on Computer and Communications Security*, pp. 433–444 (cit. on p. 10).

Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway (1998). "Relations Among Notions of Security for Public-Key Encryption Schemes." In: *Annual International Cryptology Conference*. Springer, pp. 26–45 (cit. on p. 46).

Matt Blaze, Gerrit Bleumer, and Martin Strauss (May 1998). "Divertible Protocols and Atomic Proxy Cryptography." In: vol. 1403, pp. 127–144 (cit. on pp. 4, 19, 45, 49, 53).

Dan Boneh and Matt Franklin (2001). "Identity-Based Encryption From the Weil Pairing." In: *Annual international cryptology conference*. Springer, pp. 213–229 (cit. on p. 7).

Dan Boneh, Amit Sahai, and Brent Waters (2011). "Functional Encryption: Definitions and Challenges." In: *Theory of Cryptography Conference*. Springer, pp. 253–273 (cit. on p. 7).

Yi Cai and Xudong Liu (Aug. 2014). "A Multi-use CCA-Secure Proxy Re-Encryption Scheme." In: *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pp. 39–44 (cit. on pp. 50, 53–56, 60, 63, 64, 67, 83–85).

Craig Costello (2012). *Pairings for Beginners* (cit. on p. 17).

Robert H. Deng, Jian Weng, Shengli Liu, and Kefei Chen (2008). "Chosen-Ciphertext Secure Proxy Re-encryption without Pairings." In: *Cryptology and Network Security*. Ed. by Matthew K. Franklin, Lucas Chi Kwong Hui,

and Duncan S. Wong. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–17. ISBN: 978-3-540-89641-8 (cit. on pp. 49, 53).

Whitfield Diffie and Martin Hellman (1976). "New Directions in Cryptography." In: *IEEE transactions on Information Theory* 22.6, pp. 644–654 (cit. on p. 18).

Yevgeniy Dodis, Leonid Reyzin, and Adam Smith (2004). "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data." In: *Advances in Cryptology - EUROCRYPT 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 523–540 (cit. on pp. 3, 9).

Xiong Fan and Feng-Hao Liu (2016). "Various Proxy Re-Encryption Schemes from Lattices." In: *IACR Cryptol. ePrint Arch.* 2016, p. 278 (cit. on pp. 52–54).

Kevin Edward Fu (1999). "Group Sharing and Random Access in Cryptographic Storage File Systems." PhD thesis. Massachusetts Institute of Technology (cit. on p. 6).

Javier González (Mar. 2015). "Operating System Support for Run-Time Security with a Trusted Execution Environment." PhD thesis (cit. on p. 22).

Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters (2006). "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data." In: *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 89–98 (cit. on p. 7).

Felix Hörandner and Franco Nieddu (2019). "Cloud Data Sharing and Device-Loss Recovery with Hardware-Bound Keys." In: *International Conference on Information Systems Security*. Springer, pp. 196–217 (cit. on pp. 4, 80).

Zheng Huang, Qiang Li, Dong Zheng, Kefei Chen, and XiangXue Li (2011). "YI Cloud: Improving user privacy with secret key recovery in cloud storage." In: *Proceedings of 2011 IEEE 6th International Symposium on Service Oriented System (SOSE)*. IEEE, pp. 268–272 (cit. on p. 10).

Anca-Andreea Ivan and Yevgeniy Dodis (2003). "Proxy Cryptography Revisited." In: *NDSS* (cit. on p. 44).

M. M. Jiang, Y. P. Hu, B. C. Wang, F. H. Wang, and Q. Q. Lai (2015). "Lattice-Based Multi-Use Unidirectional Proxy Re-Encryption." In: *Security and Communication Networks* 8.18, pp. 3796–3803 (cit. on pp. 52, 53).

Andrew Teoh Beng Jin, David Ngo Chek Ling, and Alwyn Goh (2004). "Biohashing: Two Factor Authentication featuring Fingerprint Data and

Tokenised Random Number." In: *Pattern recognition* 37.11, pp. 2245–2255 (cit. on p. 9).
Burt Kaliski (2000). *RFC2898: PKCS#5: Password-Based Cryptography Specification Version 2.0*. USA (cit. on p. 3).
Kohei Kasamatsu, Satoru Kanno, Tetsutaro Kobayashi, and Yuto Kawahara (Feb. 2014). *Barreto-Naehrig Curves*. MEMO. URL: `https://tools.ietf.org/id/draft-kasamatsu-bncurves-01.html` (cit. on p. 64).
Varad Kirtane and C. Pandu Rangan (2008). "RSA-TBOS Signcryption with Proxy Re-encryption." In: *Proceedings of the 8th ACM Workshop on Digital Rights Management*. DRM '08. Alexandria, Virginia, USA: ACM, pp. 59–66. ISBN: 978-1-60558-290-0 (cit. on pp. 49, 52, 53).
Lu, Lin, Shao, and Liang (2014). "RCCA-Secure Multi-use Bidirectional Proxy Re-encryption with Master Secret Security." In: *Provable Security*. Ed. by Sherman S. M. Chow, Joseph K. Liu, Lucas C. K. Hui, and Siu Ming Yiu. Cham: Springer International Publishing, pp. 194–205. ISBN: 978-3-319-12475-9 (cit. on pp. 50, 53).
Alfred Menezes (2009). "An Introduction to Pairing-Based Cryptography." In: *Recent trends in cryptography* 477, pp. 47–65 (cit. on p. 17).
Kathleen Moriarty, Magnus Nystrom, Sean Parkinson, Andreas Rusch, and Michael Scott (July 2014). *PKCS #12: Personal Information Exchange Syntax v1.1*. RFC 7292. RFC Editor, pp. 1–29. URL: `https://tools.ietf.org/html/rfc7292` (cit. on p. 21).
Abhishek Nagar, Karthik Nandakumar, and Anil K Jain (2010). "Biometric Template Transformation: a Security Analysis." In: *Media Forensics and Security II*. Vol. 7541. International Society for Optics and Photonics, 75410O (cit. on p. 9).
David Nuñez, Isaac Agudo, and Javier Lopez (Apr. 2015). "NTRUReEncrypt: An Efficient Proxy Re-Encryption Scheme Based on NTRU." In: *10th ACM Symposium on Information, Computer and Communications Security (AsiaCCS)*, pp. 179–189. ISBN: 978-1-4503-3245-3 (cit. on pp. 51, 53).
Le Trieu Phong, Lihua Wang, Yoshinori Aono, Manh Ha Nguyen, and Xavier Boyen (2016). *Proxy Re-Encryption Schemes with Key Privacy from LWE*. Tech. rep. IACR Cryptology ePrint Archive 2016/327 (cit. on pp. 52, 53).
Adi Shamir (1979). "How to Share a Secret." In: *Communications of the ACM* 22.11, pp. 612–613 (cit. on pp. 3, 9, 10, 36).

— (1984). "Identity-Based Cryptosystems and Signature Schemes." In: *Workshop on the theory and application of cryptographic techniques*. Springer, pp. 47–53 (cit. on p. 7).

Jun Shao, Peng Liu, Zhenfu Cao, and Guiyi Wei (2011). "Multi-Use Unidirectional Proxy Re-Encryption." In: *Proceedings of IEEE International Conference on Communications, ICC 2011, Kyoto, Japan, 5-9 June, 2011*. IEEE, pp. 1–5. DOI: 10.1109/icc.2011.5962455. URL: https://doi.org/10.1109/icc.2011.5962455 (cit. on pp. 50, 53, 54).

Jun Shao, Rongxing Lu, Xiaodong Lin, and Kaitai Liang (2016). "Secure bidirectional proxy re-encryption for cryptographic cloud storage." In: *Pervasive Mob. Comput.* 28, pp. 113–121. DOI: 10.1016/j.pmcj.2015.06.016. URL: https://doi.org/10.1016/j.pmcj.2015.06.016 (cit. on pp. 50, 53).

TRESORIT (2014). *tresorit - WHITE PAPER*. URL: https://tresorit.com/files/tresoritwhitepaper.pdf (cit. on p. 11).

Jian Weng, Robert H. Deng, Xuhua Ding, Cheng-Kang Chu, and Junzuo Lai (2009). "Conditional Proxy Re-encryption Secure Against Chosen-ciphertext Attack." In: *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*. ASIACCS '09. Sydney, Australia: ACM, pp. 322–332. ISBN: 978-1-60558-394-5 (cit. on p. 44).