LTIMindtree

Proxy Re-Encryption-Based Data Sharing Framework

WHITEPAPER

ABSTRACT

The process of managing data has transformed completely since the evolution of digital era. Earlier, most of the data was on paper or in the form of printed material. However, today, every organization has shifted either towards the usage of digitalized data or is slowly moving in that direction. Though data-managing becomes much easier through digitalization, it also opens a variety of new security risks that can pose harm. The main goal of the proposed framework is to eliminate these risks and also to provide access management facility, offering the user absolute control of their data.

I. INTRODUCTION

Since the past decade, the world is gradually moving towards digitizing everything. It is because of this digitization that the value of data has been growing exponentially and is considered one of the most expensive commodities in the world. With this digitization of data comes several problems like data security, data storage, access control/ownership of data, among others.

Though there are many ways to keep the data secure, there have always been loopholes or points of vulnerability which can be used to gain unauthorized access to data. The proposed framework can help avoid these loopholes and make data more secure while sharing or when stored in a database. The proposed framework uses the concept of re-encryption to achieve this, elaborated in the subsequent sections of the paper.

II. TRADITIONAL METHOD OF SECURING DATA

One of the most common methods used for securing data is encryption. In this method, the data to be secured is encrypted using one of the several encryption algorithms like RSA, AES, 3DES, among others. One of the main issues with traditional encryption techniques is that as long as the key is secured, the data is secured. However, in scenarios which require sharing of this encrypted data, the receiver must have this key to decrypt the same. Given this aspect of sharing the key with the receiver makes the whole process vulnerable.

III. INTRODUCTION TO RE-ENCRYPTION

Re-encryption is a form of public-key encryption with a few changes. Traditionally, in case of public-key encryption, if the encrypted data is shared with someone to be decrypted, the receiver would require the sender's private key - thus requiring an exchange of keys. In the case of re-encryption, a re-encryption key gets generated. This key can be shared instead of sharing private key. The same key can then be used to re-encrypt the data in terms of receiver's public key which the receiver can easily decrypt using his/her private key. This process can be elaborated further by the following process:

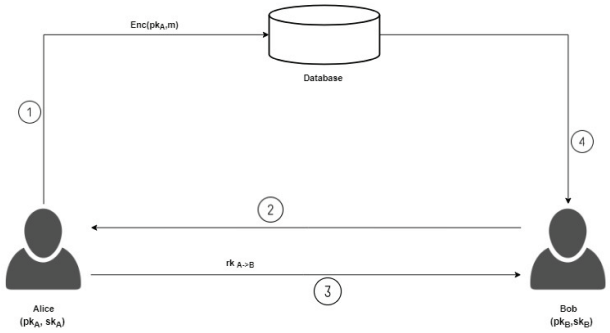


Fig. 1. *Re-encryption Process*

1. Alice encrypts the data using her public key and uploads it on a database.
2. Bob sends a file request to Alice.
3. On accepting this request, Alice generates a re-encryption key and sends it to Bob.
4. Bob can get the encrypted data from the database and re-encrypts it using the re-encryption key provided by Alice and decrypt it using his private key.

Some points need to be noted here as follows:

- The re-encryption scheme is unidirectional. This key enables the transformation of encrypted data or ciphertext only in one direction, i.e., from Alice to Bob in this case [1].
- The re-encryption key generated here can be re-encrypt any data which is encrypted using Alice's public-key.
- The process of re-encryption can be multi-hop, i.e., the re-encrypted data can be re-encrypted again [1].
- In the same scenario, even if someone gets hold of the re-encryption key, it wouldn't be of any use as it would only re-encrypt the data and the receiver's private key is still required to decrypt the data.

With the help of the above flow, one can see that this process automatically eliminates one of the significant vulnerability points i.e. exchanging of keys between the parties. At the same time, it also presents a vulnerability that as stated above - the re-encryption key generated here can re-encrypt any data which is encrypted using Alice's public-key. It's not a good idea to share the re-encryption key with Bob as that will enable him to access that data also which he was not provided access by Alice.

Hence, to overcome the vulnerability cited above, there should be a system which acts as a mediator, enforcing access control policies specified by the user without actually knowing the actual data and without sharing the re-encryption keys with any of the recipients.

IV. THE RE-ENCRYPTION FRAMEWORK

As mentioned in the previous section, this framework proposes a system which acts as a mediator that enforces the access control policies as provided by the user in addition to the normal re-encryption process explained before. This mediator system could merely be a server exposing some API for this operation or a smart contract deployed on a blockchain platform enforcing the access policies as specified by the user.

This process can be further understood with the help of the following flow:

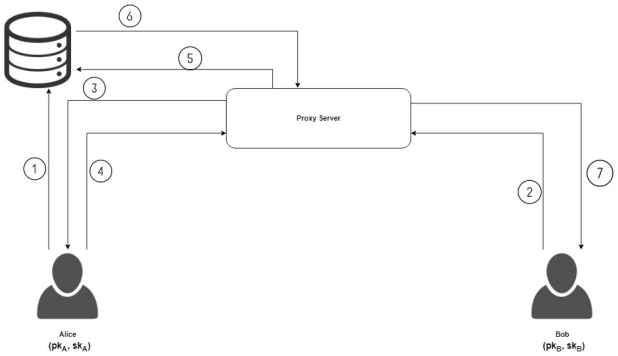
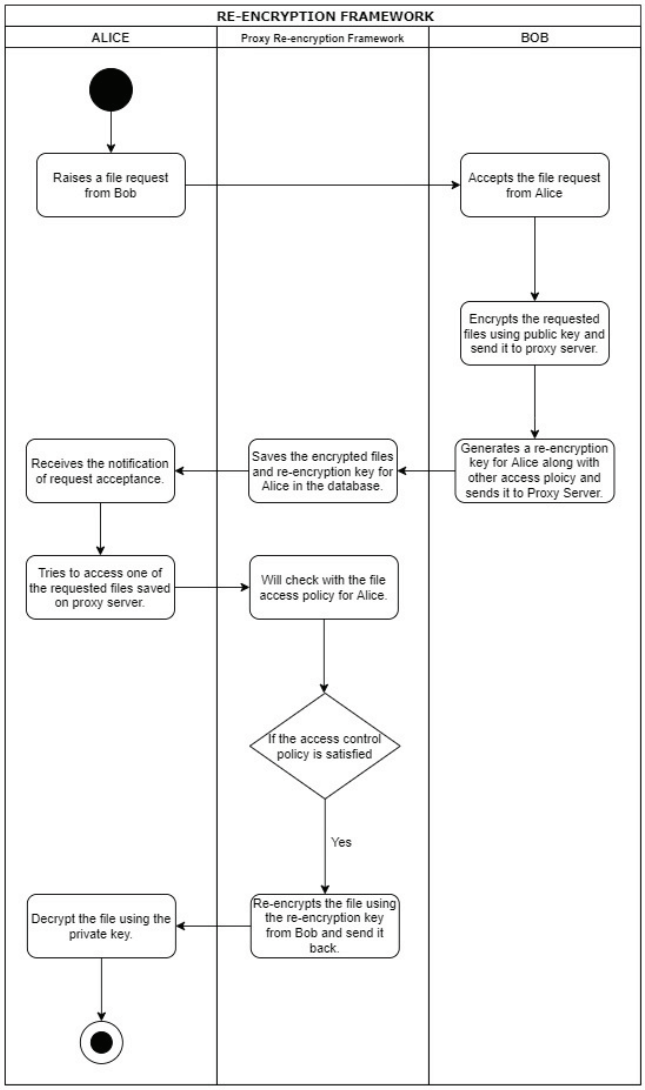


Fig. 2. *Proxy Re-Encryption Flow*

1. Alice encrypts the data using her public key and uploads it on a database.
2. Bob sends a file access request to access some files of Alice to the proxy server.
3. The proxy server will check with the existing policies that if Bob is allowed access to that file. If no such policy exists, it will send the request to Alice to either accept or reject the request.

4. If Alice accepts the request, a new policy will be created stating the files allowed, tenure for which the access is allowed, and the re-encryption key will be sent to the proxy server, and Bob will be notified that the file requested is available.
5. The proxy server will save this policy in the database along with the re-encryption key.
6. Now, when Bob tries to access this file, the proxy server will pull Alice's encrypted file along with re-encryption key if the access policy is still valid.
7. The proxy server will re-encrypt the file using the re-encryption key retrieved from the database previously and send the re-encrypted data to Bob.

The activity diagram of the above explained flow is as follows:



Some of the advantages of the proposed framework are as follows:

- The data uploaded in the database is in encrypted form, so even if there is a breach in the database, the actual data would not be exposed.
- In the same scenario, even if someone gets hold of the re-encryption key, it wouldn't be of any use as it would only re-encrypt the data and will still require the receiver's private key to decrypt the data.
- As mentioned before, there isn't any actual exchange of private keys happening.
- This whole process is pluggable and can be implemented on top of any system requiring exchanging of data between the parties.
- The concept of policy enforcing and storing can also be implemented using Blockchain Smart Contract, thus increasing the availability and security of the system.
- Using this framework gives a user absolute ownership of the data. Though the data is stored in a database collectively, all of it is encrypted, so unless a user provides access to data by themselves, no one can access it.

V. POSSIBLE APPLICATIONS

As mentioned earlier, this framework can be applied anywhere there is a requirement of data exchange between two different parties. Some of the possible scenarios of its application are as follows:

- In case of sharing medical documents, every time a person goes to a hospital or clinic for consultancy or treatment, he/she needs to share the past medical records and document with the doctor to proceed. This framework can be used here, where the user can choose what, and with whom to share the medical documents.
- In case of Digital Identity, in the present scenario, there is always a need to prove one's identity to initiate a process such a visa application, purchasing a new internet connection or anything else. This framework can be used by the users to share their identity document with the vendor/organization when required. Users can also have the ability to revoke access in case the user changes his/her mind, or once the process is complete.

There are many other use cases as well, but the essence of all of them remain the same i.e., 'Sharing of Data.' This framework can also be used to keep track of data access by someone else which can prove to be a handy feature for auditing purposes, as well as to avoid the misuse of data.

VI. FURTHER IMPROVEMENTS

Though the framework solves many security issues which one may encounter while sharing data currently, there are some points which can further be improved:

- A significant issue is key management. From an end-user point of view, one can understand that the cryptographic keys involved are long and almost impossible for a user to memorize. There needs to be a way to manage these cryptographic keys (public key and private key); it can be through a hardware wallet or any other method which ensures that the keys remain safe, enabling the user to use them with simplicity.
- Another aspect of this is that the policy data can be updated only by the person uploading it. In the process explained above, there is a vulnerable point, i.e. the access policy stored in the database can be susceptible to tampering in case of a data breach.

References

[1] DAVID NU~NEZ, "UMBRAL: A THRESHOLD PROXY RE-ENCRYPTION SCHEME"

Santanu Mukherjee
Blockchain CoE Lead, CAG,
LTIMindtree Ltd.
Bengaluru, India
santanu.mukherjee@ltimindtree.com

Vishal Chaurasia
Blockchain CoE,
LTIMindtree Ltd.
Bengaluru, India
vishal.chaurasia2@ltimindtree.com