Empowering Confidential Data Sharing with Shamir Secret Sharing

Vaishnavi Kunduru, Venkata Sai Bharat Chaganti, Akshay Jain, Sanjana Desai
University of Texas at Dallas,
kxv200027, vxc210006, axj190052, sxd200114

1. Introduction

There is an enormous amount of information that is produced every day, and massive amounts of data will be communicated on a regular basis. Some of the data that is transmitted includes sensitive information, and it is essential that this information be safeguarded at all costs. There is a vast variety of private information that is only meant to be shared with a select few individuals. Sharing passwords is one of the most widespread practices in today's world.

There are many risks associated with Password sharing such as unwanted privacy breaches, hacking and phishing etc. Privacy Breaches have become very common. We wanted to overcome such issues by sharing the password securely with a limited group of people. The main novelty of this application is that in case something happens to the user, it can entrust a group of people to access the contents of the file.

2. Objectives

In this project we have created an application that helps us to share sensitive information with the trusted ones with the help of password protection as well as encryption so that no other entity can access the information. The user can save any information such as passwords, bank account number etc., which will be stored securely in Google Cloud in a encrypted text file

3. High-Level Design

In this project, an app is developed that allows users to securely share sensitive information which has been implemented in 5 stages. The program will enable the user to enter any information such as passwords, bank account number etc, which will be stored securely in Google Cloud in an encrypted text file. The information is encrypted using AES-256 with GCM mode using a combination of user entered password and salt value.

Users can open the file as often as they want. The file stored in the cloud will be checked for any unauthorized modification. If the file is modified, an error warning to the user will be displayed. Users will be allowed to change the contents of the file as often as they want. During the update process, the encrypted file will be downloaded from the Google cloud, checked for any unauthorized modification. Users also have the option to change the password used, and store the contents with a new password.

The highlight of the program is that the user can entrust a set of people to access the encrypted information such that not a single person has access to the information but may allow a subset of users such as any ‘t’ out of ‘n’ trusted users to open the file. The app uses Shamir secret sharing to secretly share the sections of AES key with n people such that, any t of them can come together to decrypt the file.

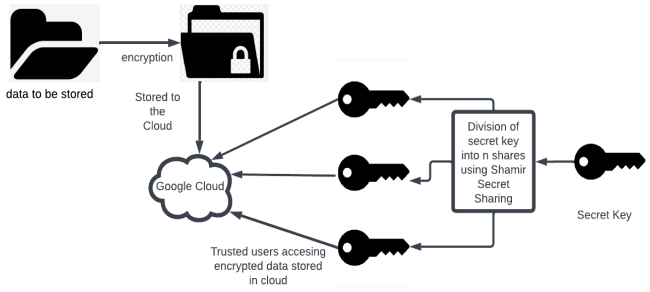

Figure 1: block diagram for the proposed system

For the user to be able to upload the file to the drive, access to the internet is required

1. Software Required:
    a. Windows 10
    b. Python 3.10
2. Hardware Required:
    a. I5 7th Generation Processor
    b. 4GB Ram

4. Implementation Design

1. Secret Sharing Key

Secret sharing methods are perfect for distributing sensitive and vital data. The loss of any of these bits of information might be fatal, thus they must be kept private. Therefore, a safe secret sharing technique is implemented that shares the key with n trusted users and distributes shares such that at minimum t trusted users are required to decrypt the information.

2.AES-GCM algorithm

Advanced Encryption Standard with Galois Counter Mode(AES-GCM) ensures data secrecy and authenticity in communication applications. AES-GCM block cipher mode enables fast authenticated encryption and data integrity. Today's poor privacy protection makes data hackable hence

we are in need of AES. We have used AES-GCM's 256 bit length key to provide a high-security system.

Approach:

The application allows the user to input the sensitive data which will be encrypted using AES-256 key with GCM mode derived by a user entered password. The encrypted data is stored in a file uploaded at a Google drive which can be accessed by the user. Before decryption, the file will be checked for any modifications in the data and based on the result, successfully decrypt the file to restore the data.

The application also provides functionalities such as it provides the user the ability to change the password used for encryption and update the encrypted data as per the new password. Also, the user can distribute the secret password among n people using Shamir secret sharing and allowing m out n people to decrypt the file at a given time.

The implementation was carried out in 5 stages:

I. Encrypting and decrypting the data:

This was the first phase we started working with. In order to begin the encryption process, a random 16-byte salt is generated with the help of CryptoDome library. The application then accepts a secret passphrase which will be used for encryption and decryption purposes. The application then accepts sensitive data from the user which is to be encrypted and stored in the file. Additionally, the program will store additional authentication data to detect any possibility of changes in file.

Once all the required information is stored, the application uses PBKDF2 protocol to generate a key based on the randomly generated salt and user entered passphrase. The key is then used to generate the encrypted text using the AES-GCM algorithm and the encrypted data is then stored into a file which the user can decide to upload on the cloud.

For decryption purposes, the user reads the encrypted data and decrypts it based on the user entered passphrase. The application asks the user to enter the password used for encryption and decrypts the file if and only if the user enters the correct passphrase thus ensuring the authenticity of the user.

II. Storing and downloading the data from the Google drive:

After the required encryption and decryption, we tried to store the files on the google drive. So, in this phase the encrypted file that is being created is stored directly on the google drive of the user who inputs the sensitive data. To store the file on the google drive we first enabled the google drive APIs from the developer console. After enabling the API's, creating the required credentials and installing the libraries such as pydrive and google Auth we were able to upload the file on the drive. When the user enters the input for the first time the authentication is carried out and after that only the file is stored on the drive. After uploading the file if the user wants to check or download the file with the help of the same libraries, we created the function for downloading the file from the drive. In this the file whose file_id and name are known only that file can be downloaded from the drive. So, with this we can make sure that only the user who knows about the file can download the file.

III. Checking for the unauthorized Modifications in the data:

In this phase we check the contents of the file for any modification. Firstly, the authTags are checked for any modifications. Also, if any additional change is detected in the data entered by the user at the time of encryption, it alerts the user of the change. The integrity check is done before the file is processed for decryption

IV. Changing Password:

Once encryption and decryption has been done to the message file, in the interface menu we implement the password change option. When the user is verified, they can update the secret passphrase as desired. Since the key generation for encryption depends on the user's password, we will modify the key and the attributes corresponding to it. This prompts a new encryption update to the message stored by the user in the cloud.

V. Secretly sharing the password to loved ones:

In this phase, the 256 bit AES key that is generated by the AES-GCM algorithm is distributed among the shareholders by splitting the key into shares. Generally, the key is divided into n shares and t is the minimum threshold (minimum number of shares that are used to reconstruct the secret again) both of which (n and t) is determined by the user. If the number of shares is less than the t values, then the reconstructed cannot be done and the file cannot be decrypted. The shares are generated using polynomial interpolation.

Phase1: Generating the shares

Given n, t values, we construct a random polynomial P(x) with a degree of t-1 and setting the constant term equal to the secret key. To generate n shares, we randomly pick n points and distribute them among the loved ones.

Phase2: Reconstruction of secret key

Minimum of t shares are required for the reconstruction. After getting the shares, we use an interpolation algorithm to reconstruct it. After generating the reconstruction polynomial then finding P(x) for x = 0 generates the secret.

5. Results and Inference

We have our current system under multiple test instances. Firstly, the user needs to select one of the options from the list of available ones. The user options are provided in the below snapshot.

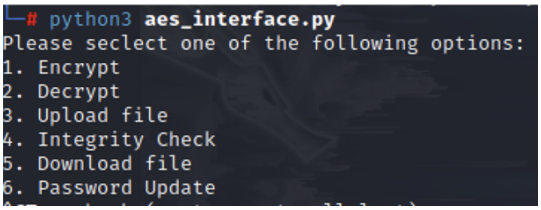

Figure-2: User Interface Menu options.

- Interface to enter confidential information is provided below.

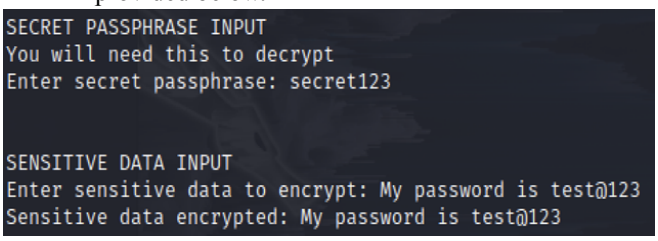

Figure-3: Interface to enter passphrase information.

- Process involving writing a file to the cloud infrastructure is below.

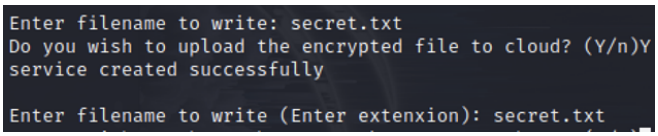

Figure-4: Interface to store encrypted files into the cloud base.

- Secret key sharing and key reassembling and data decryption process is shown as below.

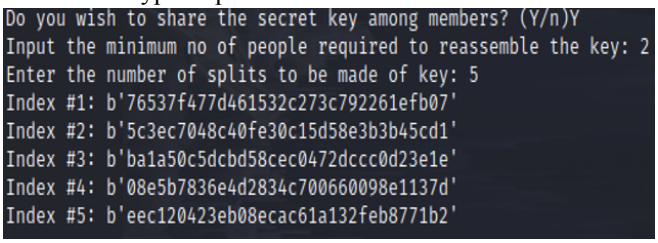

Figure-5: Division of secret key with desired individuals.

- Computation to the decryption process of sha encryption is shown below.

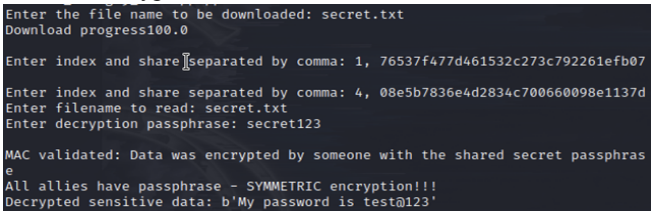

Figure-6: Decryption process on password retrieval.

6. Challenges

There were several challenges encountered during the design of our model:

- **Cryptographic key protection:** One of the key obstacles related to sensitive information exchange is key administration, as it demands the allocation and development of distinct passphrases within all the entities. This task requires precise handling to verify protecting the keys from unauthorized access and restricted to authorized recipients.
- **Adaptability:** Confidential information exchange using Shamir can become cumbersome and complicated with substantial data exchange consisting of multiple stakeholders resulting in reduced productivity and causing incompetence in the information exchange process.
- **Compatibility:** To enable effective cryptographic key exchange, all relevant entities must implement adaptable encryption techniques. When a different encryption technique or procedure is employed by an entity, it results in system incompatibilities compromising the encrypted data sharing.
- **Security:** Although confidential data transmission can be highly encrypted, It demands reliance among all participating entities. If one party is breached, it causes a security breach of the whole machine, Threatening the confidentiality of sensitive information to unauthorized individuals.

7. Future work

This proposal can be prolonged in multiple directions:

- Shamir Secret Sharing can be integrated into existing systems and applications to enhance their security and confidentiality..
- An Alternative way is to improve the safety of Shamir confidential shares. This may require designing unique cryptographic approaches or enhancing, or mitigating potential security risks in employing Shamir Secret Sharing.
- The Shamir secret sharing is combined with the encryption workflow in the current system. The client can utilize the feature to isolate the two workflows.
- During password alteration and file upload process with freshly secured data, it does not remove the former distribution of the file metadata that require file removal from the cloud storage.
- Distributed Key Generation technique enhances accessibility as We just require entry to multiple cloud storage services.

References

[1]https://en.wikipedia.org/wiki/Shamir%27s_secret_sharing

[2]https://iopscience.iop.org/article/10.1088/1742-6596/1019/1/012008/pdf

[3]Gupta, Kishor Datta & Dasgupta, Dipankar & Poudyal, Subash & Rahman, Md Lutfar. (2020). Shamir's Secret

Sharing for Authentication without Reconstructing Password. 10.1109/CCWC47524.2020.9031270.

[4]P. Čuřík, R. Ploszek, and P. Zajac, “Practical Use of Secret Sharing for Enhancing Privacy in Clouds,” Electronics, vol. 11, no. 17, p. 2758, Sep. 2022, doi: 10.3390/electronics11172758.

[5]https://medium.com/bootdotdev/shamirs-secret-sharing-step-by-step-42eb8cafa102

[6]https://pycryptodome.readthedocs.io/en/latest/src/protocol/ss.html