University of Bucharest
Faculty of Mathematics and Computer Science

Study programme: Computer Science

Bachelor's thesis

A survey of secret sharing schemes

Coordinator: Assistant Professor Adela Georgescu
Student: George Bodea

Bucharest, July 2022

Contents

Abstract

A secret sharing scheme is a mechanism for distributing information so that only a qualified group of participants can access sensitive data. Such schemes are used as cryptographic primitives in constructing authentication protocols, multiparty computation and more. Better understanding of secret sharing schemes contribute to increased information security.
In this survey, we will briefly present the definitions and observations of previous research papers and implement in code the most known scheme. The implemented code is also the basis of the application that illustrate how can the scheme be used in practice. The purpose of this work is to introduce the basic scientific notions to a reader that has no knowledge of secret sharing schemes or cryptography in general, although elementary understanding of mathematics and computer science is required.

Rezumat

O schemă de partajare a secretelor este un mecanism de distribuire a informațiilor astfel încât numai un grup calificat de participanți să poată accesa date sensibile. Astfel de scheme sunt utilizate ca primitive criptografice în construirea protocoalelor de autentificare, calcul multipartit și altele. O mai bună înțelegere a schemelor de partajare a secretelor contribuie la creșterea securității informațiilor. În acest studiu, vom prezenta pe scurt definițiile și observațiile din lucrările de cercetare anterioare și vom implementa în cod cea mai cunoscută schemă. Codul implementat este, de asemenea, baza aplicației care ilustrează modul în care schema poate fi utilizată în practică. Scopul acestei lucrări este de a prezenta noțiunile științifice de bază unui cititor care nu are cunoștințe despre schemele de partajare a secretelor sau despre criptografie în general, deși este necesară o înțelegere elementară a matematicii și informaticii.

Chapter 1

Introduction

1.1 Background

Secret sharing is an important system of distribution and reconstruction used in many applications today. In order to hide confidential data (highly classified documents, nuclear missiles launching codes, personal and/or sensitive information, access code for a bank vault) we can use different approaches to achieve the security that we want, within a reasonable span of time. In the examples above, putting the access key in just one location is not feasible. In order to protect the most important of secrets, one should put parts of the key for finding the secret in more than one storage location. But, not all the parts must be necessary to reconstruct the secret. In this way, if one location is corrupted by an attacker or even destroyed due to natural causes, the secret can still be revealed. Moreover, when numerous participants must collaborate to obtain the secret, we have to keep in mind that in real case scenarios, even some of those individuals participating can be corrupted and swayed in favor of destroying the secret or making it impossible for the rest of the group to obtain the secret, therefore we must choose wisely how many parts of the key can reconstruct the secret, or even, how important is each part of the key among others. Even more aspects come into play but depending on the type of situation and what exactly we want to achieve, we have to make some compromises, either on security or performance.

1.2 Genesis

The concept of a secret sharing scheme has been introduced formally by Shamir [19] and Blakley [3] in 1979, in different papers, although the first paper is more referenced due to the more simplistic approach. This major discovery led to the creation of various flavours of schemes, such as the general access scheme first explained by Ito, Saito, and Nishizeki [15], ideal and linear schemes studied by

Brickell [6] and some more practical schemes such as multi-linear schemes and hierarchical schemes.

1.3 Motivation

The motivation for this paper is to provide an oversimplified guide on the topic of secret sharing schemes. The inspiration for this idea came immediately after learning about this cryptographic branch. In spite of the fact that it provides interesting ideas and represent an important part of cryptography, not so many tutorial-like papers have been produced to arouse interest in a reader. Most research papers are esoteric and convoluted in explaining the main ideas. Moreover, this work provides support for a developer looking to create an application that focus specifically on developing security tools or boost internal security of other applications.

1.4 Tough situation

One real case when a secret sharing scheme can be useful is when we have a number of U.S. officials burdened with the task of taking immediate action during a DEFCON-1 scenario. DEFCON is a state of readiness of the U.S. military which indicates how serious should the army respond, ranging from DEFCON-5 (low level of threat, normal readiness) to DEFCON-1 (maximum level of threat, prompt response needed). At the highest level of threat, nuclear war is imminent or already happening and as a matter of course, the nuclear missiles have to be launched. Since this is a desperate, last action plan with extreme consequences, the security of the codes to be launched is of upmost national and even global importance. Entrusting a single person with the codes is, by far, too risky. Splitting the responsibility is the only way in which we get to a situation were a decision is not made impulsively or in ill faith. Now, depending on the type of officials that have the requisite authority we can decide on at least two types of schemes: one that splits the launching code in equally important parts or one that is based on ranking. For example, if all the persons entrusted are generals, then each one of them will have the same jurisdiction over the decision. If the approval of the president is required, then a better fit would be to give the president a vital part of the launching code, therefore having a hierarchical structure of power.

1.5 Contribution

My contribution stands in presenting briefly the definition that had already been introduced, but in a more informal way along with known properties and a few

comparisons for each type of applicable scheme, using reputed academic sources to introduce the reader in as many as 11 schemes. Recommendations for when to use them in general, based on the scheme's properties, are also provided along with concrete examples of uses in cryptography. Even more, a real application that represents one paramount scheme will be included for analysing. The implementation characteristics are to instruct the reader on how to start in developing another type of scheme based on the one in cause. Even though detailed surveys about a few schemes had already been written, one solid example would be Secret-sharing schemes: A survey [1], the level of complexity is high, although they provides more rigorous theoretical explanations.

1.6 Exercise

The application embodies the concept of one the most known referenced scheme, Shamir's threshold secret sharing scheme. It has a straightforward graphical interface, in the sense that, if the user knows how Shamir's scheme works, the manual on how to use it is just a formality, except for the usages of external cloud databases like: Google Firebase, Clever Cloud and Azure Cosmos DB. The application asks the user to choose between two options that represent the two major steps of the algorithm related to the scheme. In each of the steps, the user has to introduce an input. After the input is introduced, the user has to select the preferred databases in which information for the second step is needed. The user can use the offline method of distributing or reconstructing a secret, use the cloud databases or both, each option with advantages and disadvantages.

1.7 Overview

In the next chapters a few conventions will be introduced as there are very common notions on the topic of secret sharing schemes along with descriptions of the schemes themselves. Later on, we will explain the implementation of the code that will lead us to the last part being the application. Chapter 2 will present three important notions that are recurrent in many of the prominent research papers studied. They will set the stage in explaining the schemes. Chapter 3 is the most theoretical oriented part of the paper and explains what is a secret sharing scheme, what are some of the attributes that come with the scheme, some recommendations for when to use them and some uses in cryptography. Proceeding forward we will encounter many canonical types of schemes, although some have been cataloged only later on from the moment they have been introduced. The reason for not going in depth with all the properties is simply because each of those schemes constitute lengthy papers on their own that require a strong understanding of mathematical knowledge and computer science. To understand what is the purpose of that specific scheme we will introduce examples. In the last subchapter we will draw some comparisons on the

relationship between some of the concepts shown. In Chapter 4, the practical project will be dissected to make it easy for the reader to capture the whole architecture and functionality of the application and give clear instructions on how to use this tool.

Chapter 2

Preliminaries

In this chapter we will present used notions throughout cryptography and will set the stage for explaining why and how we use secret sharing schemes. Not all of the papers pertaining to a specific category of schemes use all of the notions presented here but, by convention, we refer to them so to have a common ground when explaining all of the schemes.

2.1 Qualified and unqualified sets

A qualified set of participants can reconstruct the secret. An unqualified set of participants can not learn any bits of information about the secret. We consider Γ the access structure and the set Π the unqualified subset of participants. Naturally, any set that exists in Γ is not included in Π, and vice versa. For example, if $\Gamma = \{\{P_1\}, \{P_1, P_2\}\}$, then $\Pi = \{\{P_2\}\}$ is possible but definitely not $\Pi = \{\{P_2\}, \{P_1\}\}$.

2.2 Probabilistic polynomial in time adversary

A probabilistic polynomial in time (PPT) adversary has the means of breaking a cryptographic scheme in polynomial time. Throughout this paper, we consider the attacker as a PPT attacker, as this is the classic adversary in cryptography.

2.3 Access structure

An access structure of a secret sharing scheme is the collection of all qualified sets. Usually, they are represented using set building notation. For instance, for the set

$\{P_i \mid i \neq 1\}$ given as access structure we can say that any set of participants that does not contain the participant P_1 is included in Γ.

Chapter 3

Secret sharing schemes

This chapter considers the definitions and comparisons of distinct types of secret sharing schemes. Separating schemes is very important for finding out which one of those schemes is used in what scenario. Each one of those schemes have something that makes them unique and having that specific property in mind can lead to creation of other structures.

Additionally, we can also see the limitations and deficiencies so to better measure the overall security in regards to what we want achieve. Some of the most common schemes have examples to make it easy for the reader to grasp the concepts explained.

3.1 Secret sharing scheme

A secret sharing scheme (SSS) is a mechanism in which a dealer distributes a secret in the form of shares to a number of participants in order for a qualified set to reconstruct the secret. The scheme is traditionally defined as satisfying two properties:

- Correctness: any qualified set can reconstruct the secret
- Privacy: an unqualified set can not reconstruct the secret

The dealer is a designated participant which splits the secret in shares and allocates them to other participants.

3.2 Threshold secret sharing scheme

A threshold (k, n) SSS involves a dealer who distributes one share to each of the n participants such that:

- any k shares will reveal the secret

- any $k-1$ or less shares will not reveal the secret

where $1 \leq k \leq n$ [19]. The most well-known threshold SSS is Shamir's threshold (k, n) SSS [19].The algebraic idea behind Shamir's SSS is to create a polynomial function f, defined over a finite field, that yields the secret when evaluated to $f(0)$. We can portray a 2 dimensional space where the secret is precisely on the Y axis, and the shares on the $X-Y$ axis define the polynomial function.

For example, having k=2 we can determine one and only one straight line which intersects the Y axis in exactly one point, which is the secret. Having just one share gives no information on what the polynomial function might be, since there are an infinite, equally probable choices.

Definition of the polynomial function of grade $k-1$ will be given by the k shares using Lagrange's interpolation which states that it exists a unique polynomial $P(x)$ of grade $k-1$ and having a set of points $(x_1, y_1)...(x_k, y_k)$ where $P(x_i)$=y_i with $1 \leq i < j \leq k$ and $x_i \neq x_j$. The function f, defined over a finite field F, will be given by the Lagrange's interpolation formula [19]:

$$g_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{k} \frac{x - x_j}{x_i - x_j}$$

$$f(x) = \sum_{i=1}^{k} y_i g_i(x)$$

SSSs have been introduced independently by Shamir [19] and Blakley [3], whilst very similar in concept, Blakley's version is hyperplane based and thus, less space efficient.

We use the threshold SSS when we want participants with exactly the same authority to contribute in reconstructing a secret. Designing an application using this scheme is fairly fast and not overly complicated. The greatness weakness is that if a share is corrupted, the reconstruction of the secret, using that share, is compromised. Taking this fact into consideration, this particular schemes comes with high risks when not all the participants are to be trusted or the secret is stored for a long period of time because the PPT adversary could corrupt shares. One other important note is that, we put our trust in the dealer who is assumed to be incorruptible, although that is unlikely to happen.

One of the many application for this scheme is in when we implement the ElGamal cryptosystem [9]. ElGamal [10] is an asymmetric encryption system that uses the proprieties of modulus calculation in order to create a public key using a generator and an encryption exponent that will be used, in this case, to generate the shares. This threshold cryptosystem works very similar to the threshold scheme but it

uses the most of ElGamal modulus properties for the calculation of the decryption exponent that will be used to retrieve the secret.

3.3 General access structure secret sharing scheme

A general access structure SSS is very flexible in describing what the qualified parties are composed of. In fact, a general access SSS can realise any given monotone access structure [15].

A share can have more importance than others. In a real world scenario, a team leader opinion can "weight" more than the opinion of any other member in the team. For instance, a group of 3 team leaders has more influence than a combination of 3 junior engineers. Therefore, we can add more value to the shares or create specific combinations of shares, either by a pattern or manually. Although, in some cases, the size of the shares can grow exponentially in relation to the secret.

As a counter example, in the SSS introduced by Shamir [19], all of the share holders are equal in decisional power. Therefore, each share has the same importance. For the same reason, the combination of specific n shares does not have more influence over any another combination of n shares.

3.4 Proactive secret sharing scheme

A proactive SSS is a sharing scheme that updates (checks, destroys and replace) its shares periodically to avoid the danger of losing or destroying the secret by a perpetrator [14]. If done periodically and with a randomly selection algorithm of the shares, it increases overall security. The maximum number of corrupted of shares is $\frac{n}{2}-1$ [14], in order to still make possible the reconstruction of the secret. But a very important note is that the preemptive detection of the corrupted shares has to be done before the reconstruction of the secret.

In order to communicate with the holders of the shares, an intermediary broadcast channel C is implemented with synchronous properties. The updating process will take place at the same time for every server that it connects (in this setup, a participant is represented as a server). In this layout, the attacker can do any modification to a server (shut it down, listen to other messages, corrupt values) although it cannot block a message from other servers being sent on C. Also, the adversary is computationally bounded and any other cryptographic primitives are not to be considered vulnerable.

Compared to Shamir's SSS, the proactive SSS simply provides better security having a great advantage in a real world scenario, although its implementation is

not as easy, considering that a synchronous broadcast channel must be set, along with algorithms of checking, destruction and replacement of the shares. However, this option still proves insecure in regards to a corrupted share that is already being used in the reconstruction of the secret.

This scheme could be used when implementing the SSH protocol [13]. The Secure Shell Protocol is a network protocol for securely communicating with another machine. It uses asymmetric encryption where both the sender and the receiver of the request construct a public and private key.

3.5 Robust secret sharing scheme

A robust SSS has the ability to correctly reconstruct the secret even if some shares have been corrupted at any time. This scheme is actually a proactive SSS that has better identification algorithms for detecting corrupted shares [14]. If $1 \leq k < \frac{n}{3}$, using Reed-Solomon codes we can reconstruct the secret with a probability of 1 [17].

The Reed-Solomon codes are used extensively in situation where some data will get corrupted or lost, but we would still like to get the correct information, given that some information is intact. A summary introduction to Reed-Solomon is based on the same Lagrange's interpolation idea that, if you have a sufficient number of shares, but not necessarily all, you can construct the polynomial which will give the other points that are needed. For example, let's say we would like to send the data $y_1...y_u$ to a receiver. Before we send it, we will map each $y_i, 1 \leq i \leq u$ to distinct x coordinates which are equal to the indexes i, therefore having the tuples of coordinates $(1, y_1), (2, y_2)...(u, y_u)$. We can determine the polynomial with grade $u - 1$ which will define those points. Now we can calculate, using this polynomial, other $y_{u+1}...y_k$ coordinates, $k \leq u/2$ and their respective x coordinates. The number k represents the number of possible shares that can get corrupted and still can get repaired. In this way, if we loose any k tuples of coordinates, we can reconstruct them since we can determine the polynomial from which they originated by using the other u tuples, in any order. To note here is that, the corrupted or lost shares have to be detected by other means rather than the decoding itself, so this can be done using Berlekamp–Welch algorithm [24] or Euclid's algorithm.

The scheme also assures (with a high probability) that the secret can be reconstructed even if $\frac{n}{3} \leq k < \frac{n}{2}$ [18]. A broadcast channel is established for communication between other participants, similar to the one presented earlier. The strongest feature of this scheme is reliability, although it comes with a heavy cost time spent on convoluted implementation and a slower performance.

One concrete case when this scheme is used is when creating a RSA signature scheme. RSA is a public key or asymmetric cryptographic system that is broadly used for exchanging private information. Shoup [20] presented how a robust scheme can be merged together with a RSA cryptosystem to create unforgeable signatures,

even when corrupted players try to change the results.

3.6 Perfect secret sharing scheme

In regards to a threshold (k, n) scheme, we consider that scheme to be perfect if a set of $k-1$ or less share holders can not obtain any partial information about the secret [21]. Having the set Γ, we say that a secret sharing scheme is perfect if any subset of the set Π cannot partially reveal the secret. This notion can be easily extended for general access structure schemes.

In most of the cases, we want to be sure that the secret is retrieved by a qualified party and not leave clues to some semi-access qualified parties. If complete secrecy is a must, a perfect SSS is the right choice.

A very interesting application for perfect SSS is in graphs. In Csirmaz's scheme [8] the nodes represent the participants and if there is an edge from a set of participants to a specific graph, than that set is in Γ.

3.7 Ideal secret sharing scheme

An ideal SSS is perfect and the set of probable shares has the exact same space as the set of secrets [6]. An alternative definition to this would be that an ideal SSS uses shares with the same size as the secret. Referring back to the presented types of sharing schemes, we conclude that Shamir's [19] SSS is ideal: is perfect and the polynomial function is defined over a field (the coefficients of the polynomial function is in the same set as the secret).

Ideal SSS is an important scheme, or rather an important attribute in the constructions of other SSS due to the properties given. An application for this type of scheme would be ideal multipartite SSS which divides a subset of Γ into other subsets where each individual play the same role as others [11].

3.8 Linear secret sharing scheme

A SSS is linear if the secret's reconstruction from the shares is a linear mapping [1]. All of those elements, including the secret and the shares belong to the same finite field. Most SSS studied are linear SSSs. As an example, we will refer to a (2, 3) threshold scheme. Let's say that the polynomial function f is defined over the field Z_5. We have the participants P_1, P_2, and P_3 and the shares: $sh1=(1, 0)$, $sh2=(2, 3)$, $sh3=(4, 4)$, where P_i holds the share sh_i. Since the number of minimally required

shares is 2, the polynomial function is $f(x)=a_0+a_1 \cdot x$, where a_0 and a_1 are constants and $f(0)=a_0$ is the secret. Doing some simple calculations:

$f(1)=0 \Leftrightarrow a_0 + a_1=0.$
$f(2)=3 \Leftrightarrow a_0 + 2 \cdot a_1=3.$ In a matrix layout:
$f(4)=4 \Leftrightarrow a_0 + 4 \cdot a_1=4.$

$$\begin{matrix} P_1: \\ P_2: \\ P_3: \end{matrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 4 \end{bmatrix}$$

We have represented the relation between shares, the secret and some random element a_1 in a linear transformation, in this case a matrix. Doing some Gauss elimination we can deduct a_0=2, which is the secret, from any two shares. This is a particular example of a Shamir SSS translated as a Linear SSS, where the access structure is defined by the threshold and the number of shares. The following example is more general. The polynomial function is defined over Z_3.

$$\begin{matrix} P_1: \\ P_2: \\ P_3: \\ P_3: \\ P_4: \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 \\ 2 & 2 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} a_0+a_3 \\ 2a_3 \\ 2a_0+2a_1+2a_3 \\ a_2 \\ 2a_0+a_1+2a_2+a_3 \end{bmatrix}$$

We can immediately see that $\{P_1, P_2\}, \{P_3, P_4\} \in \Gamma$. Every set that includes the subsets $\{P_1, P_2\}$ or $\{P_3, P_4\}$ is an access structure. We can also observe that participant P_3 has 2 shares, a feature not found in Shamir SSS.

3.9 Multi-linear secret sharing scheme

As a matter of course, when we would like to distribute more than one secret, we could use linear SSSs multiple times, but that would be inefficient [2]. Multi-linear SSS are a natural extension of linear SSS. A multi-linear SSS distributes shares for more than one secret, unlike linear schemes. Furthermore, if an unqualified subset of Π knows some secrets, they might have some information about the other secrets [5]. But conventionally, we only consider a Multi-linear SSS as a scheme that is perfect [2], therefore any unauthorized set can not gain any information about the secret. In the next example, we will work on the field Z_7:

$$\begin{matrix} P_1: \\ P_1: \\ P_2: \\ P_2: \\ P_3: \\ P_3: \end{matrix} \begin{bmatrix} 2 & 3 & 3 & 0 & 1 \\ 3 & 3 & 2 & 4 & 5 \\ 1 & 4 & 3 & 0 & 1 \\ 4 & 2 & 2 & 4 & 5 \\ 6 & 4 & 4 & 0 & 6 \\ 4 & 5 & 5 & 3 & 2 \end{bmatrix} \begin{bmatrix} a_{00} \\ a_{01} \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 2a_{00}+3a_{01}+3a_2+a_4 \\ 3a_{00}+3a_{01}+2a_2+4a_3+5a_4 \\ a_{00}+4a_{01}+3a_2+a_4 \\ 4a_{00}+2a_{01}+2a_2+4a_3+5a_4 \\ 6a_{00}+4a_{01}+4a_2+6a_4 \\ 4a_{00}+5a_{01}+5a_2+3a_2+2a_4 \end{bmatrix}$$

We can observe that there are 2 secrets: a_{00} and a_{01}. Hence, $\{P_1, P_2\}, \{P_2, P_3\} \in \Gamma$. The position of the shares in the matrix is not important, as long as the shares belong to the same participant.

3.10 Ramp secret sharing scheme

A ramp (d, k, n) SSS is similar to a (k, n) threshold scheme with the addition that, less than d shares will not reveal any information about the secret and more than d shares but less than k shares will reveal some information about the secret [4]. We can easily conclude that, if $d{=}k$ then the structure is a threshold (k, n) scheme. We can observe that a new type of access structure is created, a semi-access or semi-authorized set of participants. Furthermore, we can state that ramp SSS is, by design, not a perfect SSS. We use ramp secret schemes in scenarios when we want the shares to be smaller in size but at the cost of security[4].

An application for ramp SSS is improved broadcast encryption [22]. The improvement in the broadcast encryption problem is the amount of data, or as it is described in the research paper, the rate of information that is sent to the participants that have to collaborate in order to obtain the encrypted information.

3.11 Hierarchical secret sharing scheme

In a Hierarchical SSS the ranks of the participants are divided and each access structure in Γ has at least k_0 highest ranking participants, $k_1{>}k_0$ second-highest ranking participants and so on [23]. This scheme gets very close to real world cases in corporations and institutions that put a strong accent on hierarchy.

This type of scheme is used in creating a hierarchical ID-Based cryptosystem [12]. As with the other cryptography systems mentioned in this paper, an ID-based encryption system is public key driven. The way it works is by using a central authority to generate a master public key and master private key. The master public key will be shared with all the users to generate, individually, the public keys of other users based on an unique identifier. The private key of each individual is distributed offline by the central authority.

3.12 Verifiable secret sharing scheme

In standard secret sharing, it is assumed that the dealer is honest and does not send wrong information to each participant. A verifiable SSS considers the case where the dealer is dishonest and some of the participants are corrupted [7]. A simultaneous

broadcast network is established in order for the participants to communicate privately (one-to-one communication) and publicly (broadcast) as to verify if their shares are legitimate (can be used for the reconstruction of the secret) and the dealer is not corrupted.

An application for this would be publicly verifiable SSS where even someone outside the group of participants can verify if the share received by a participant has been correctly distributed [16].

3.13 Comparisons

SSSs can be compared in many ways, by many parameters or designing features. Needless to say, there is no conventional way in order to achieve this. We will especially try to compare schemes that are at opposite poles since the most nuanced differentiation can be observed and see some of the advantages or disadvantages that occur.

Shamir SSS and general access structure SSS is a good starting point. In terms of special features like detecting corrupted shares or verifying the dealer, they behave the same. The difference is in the access structure where Shamir's SSS is limited to a threshold, whereas general SSS provides ways of constructing any secret sharing scheme based on monotone functions. The downside of the general SSS is that the share size can increase exponentially in regards to the secret, therefore they can prove inefficient. The advantage of Shamir SSS is on the share size and simplicity in developing an application based on the scheme.

An ideal SSS and a ramp SSS have little in common. As a matter of fact, they are completely different. The size of the shares in the ramp SSS is lower than the secret, therefore space is saved at the cost of security. In an ideal SSS, the opposite happens: it provides perfect security, therefore only the authorized sets provide information about the secret and the shares have the same size as the secret.

A linear SSS and a multi-linear SSS, besides the obvious difference in how many secrets one scheme can hold, differentiate themselves by how well they preserve secrecy. While a linear SSS does not have to be a perfect SSS (see ramp SSS), the multi-linear SSS is, by convention, a perfect SSS.

Chapter 4

Application

The goal of all of the theoretical explanations is to create a tool that can help us solve a real problem. In this chapter we will put in practice Shamir's SSS. The application provides the user with a friendly graphical user interface that does not require an in depth understanding of the Shamir's SSS and just put in practice its capabilities.

The application also provides a few additional features, like using cloud services and could also be further improved into a more useful tool or even to serve as a template for building other applications. But, for the purpose of learning, the most important aspect to be taken into account is probably not the finished product, the whole application, but the implementation of the code.

In the next subchapters we will present the technologies used in the form of libraries, internal to Python and external, also the structure of files and folders which comprises our contribution, the implementation and a manual for using the application.

4.1 Technologies

The programming language used is Python 3.10.2. Python is a reliable programming language for creating desktop applications with a large community and a varied database of libraries. The code is ran through an interpreter and is fast an easy to use, even for individuals without much programming knowledge or computer science background.

Since the purpose of this application is to illustrate and better understand how to use a SSS, selecting a programming language such as C, C++ or Java would not be as effective in developing an application as Python, given how intuitive Python is. Cloud service technologies have been used for storing information: Google

Firebase, Clever Cloud and Azure Cosmos DB. The cloud databases have been introduced for multiple reasons, such as the ability of the participant to choose between preferred database, having a secure channel for communicating the share to the participant (as a theoretical example, sending the share by WhatsApp is not as secure as sending the share directly to the WhatsApp database), easy setup of the database, demonstration of the fact that the program works when the number of necessary shares are combined.

The application has been compiled using the *pyinstaller* Python library and *NSIS*, a professional open source installer. The imported module are:

- **tkinter** for has been extensively used for designing the application's Graphical User Interface (GUI). It provides a simplistic interface as a desktop application, fit for any type of user. Moreover, it is easily customizable either for further development of functionality or style. It is also one of the most popular GUI libraries for Python.
- **math** for simple mathematical operations.
- **decimal** is vital for mathematical operations that require precision in managing floating point arithmetic and handling extremely large numbers.
- **sys** for accessing internal modules on a higher level than the folder we are located in.
- **json** for converting the access keys for the cloud databases, an input received from the user in the JSON format. The JSON format is widely known and used in many application, therefore as a convention, all access keys are expected as such.
- **firebase_admin** for handling requests as an administrator to the Firebase API.
- **mysql.connector** for handling access to the MySQL database on Clever Cloud.
- **pymongo** for handling requests to the MongoDB in Azure Cosmos DB.
- **traceback** for debug purposes where the error stack tree is printed in the terminal and minimal critical information is showed on the GUI.

4.2 Structure

The application folder comprises mainly of three folders that handle the configuration to each specific cloud service, a file for handling the creation of shares and the reconstruction of the secret, a controller file which binds all the logic functionality

and a file that provides the visual interface of the application. Each cloud service folder has a configuration file which accesses the database using a key and a file logic functionality of uploading and downloading a share, including auxiliary functions.

The file *Shamir_Secret_Sharing_Scheme.py* provides all the functions related to the distribution and the reconstruction of the secret. All the cloud services and the file *Shamir_Secret_Sharing_Scheme.py* are accessed and put together in the *API.py*, such as a controller.

The part that manages the visual interface and communicates with the user is the *GUI.py*, basically the only entry point of the application. The *s-logo.png* file is used as a logo for the application and .gitignore and README.md are used for Github related purposes.

- A-survey-of-secret-sharing-schemes
  - CleverConfig
    - clever_config.py
    - SSSS_Clever.py
  - CosmosConfig
    - cosmos_config.py
    - SSSS_Cosmos.py
  - FirebaseConfig
    - firebase_config.py
    - SSSS_Firebase.py
  - GUI.py
  - s-logo.png
  - .gitignore
  - API.py
  - README.md
  - Shamir_Secret_Sharing_Scheme.py

4.3 Implementation

The core functionally of the application is based on Shamir's SSS. The file related to mathematical algorithms and calculations in regards to generating the shares and the secret is *Shamir_Secret_Sharing_Scheme.py*. The functions presented are the following:

– *create_shares(secret, threshold, shares_number)* is the file's means of access for generating the shares. Calling the functions requires the mandatory arguments: *secret*, *shares* and *threshold*. The first step in creating the shares is constructing or, in this case, simulating a polynomial function. Since we know the common structure of a polynomial of grade N, each consecutive element x having a coefficient and a grade equal to the previous+1, starting with an element of grade 0, all we need to know is the order of the elements, the grade N and the coefficients. The order of the elements is predefined as increasing, therefore the secret is the first element of grade 0. The coefficients are created by calling the function *create_coefficients*. The final share list is created by plugging in the resulted coefficients as arguments when executing *create_points*.

– *create_coefficients(threshold)* is used for generating random numbers between *[0, MAX_BOUND)* that will serve as coefficients. It is important to note that the library *secrets* has not been used without a clear purpose. We need the polynomial to be as random as it can as to not be easily calculable by an attacker.

– *create_points(all_coefficients, shares_number)* takes the coefficients which represent the polynomial and are used for creating the x and y values which are the coordinates of the points. The x values created in a number equal to the *shares_number*, are used for creating the y coordinates by taking into consideration the list of coefficients which represent the polynomial. The *share_list* consist of a list of tuples which hold the coordinates, namely the shares.

– *create_x_coordinates(shares_number)* manages the creation of x values. One very important aspect of Shamir's SSS is that each share has to be unique. If two shares are alike, that is, if the x coordinates are the same, calculating the coordinates of the secret using Lagrange's interpolation would result in an undefined answer because division by 0 is not possible. Even more, if we try to calculate the secret using a system of equations, we obviously see that we have missing information since two expressions will be the same, therefore having two same shares will put us in the same conjuncture as having just one share. We would also need the coordinate x as a random value, for security purposes. A predictable generation of x coordinates will give an attacker the opportunity to find out the rest of the shares just by looking at one or a few shares and the algorithm itself. Taking all of this into account, the spawn of values will be as follows: we use the *secrets.randbelow* function to get a random value between *[0, MAX_BOUND)*, we check to see if it is not already in the set of *unique_shares* number and we add it, or we just keep generating a number. To be noted here is that *MAX_BOUND* is a predefined value and we guarantee that the number of shares will always be less or equal to it, because we check the number of shares at the input stage.

– *create_y_coordinates(all_coefficients, x_coordinates)* simulates the evaluation of the polynomial in cause by calculating the y coordinates and multiplying the coefficients with the coordinates x at consecutive powers. Since the order of the polynomial is increasing, the starting power is 0.

– *reconstruct_secret(share_list)* takes each share and calculates the sum in the Lagrange's interpolation formula. *Decimal* is detrimental for correctly creating a floating point number with a precision of digits given, otherwise the secret reconstructed would not be the one plugged in the input stage. The conversion function *to_bytes* transforms the integer in bytes and using *decode* the bytes are represented as an ASCII string. The byte order is described as the last argument "little", referring to the little endian order. This ensures that the conversion process can recreate the secret as a string, rather than an integer.

– *g_i(i, share_list)* is a function that calculates the product in the Lagrange's interpolation formula. We have to make sure that the applied formula is correct, therefore we take all of the x coordinates except the x_i in cause. If x_i would be equal to x_j, an error will be thrown because we can't divide be 0. The same would hold true if the shares were the same.

The *API.py* is the intermediary between all of the cloud databases, the *GUI.py* and *Shamir_Secret_Sharing_Scheme.py*. All of the logical functions can be access trough this file, hence why we import it in the GUI .

– *input_api(secret, shares_number, threshold)* takes the secret as a string, encodes it into bytes in the ASCII format, and then transforms it into an integer, since the secret has to be a number in order to perform mathematical equations on the polynomial. Normally, the secret in the number representation will be extremely big for long inputs, in practice, performance of the application will be influenced moderately by this, especially when the threshold is a small number. The shares and threshold are checked to see if they meet the logical requirements for the scheme.

– *check_threshold_shares(shares_number, threshold)* verifies that *shares_number* and *threshold* are not out of bounds and also some checks to avoid abnormal situations. First we have to make sure that *threshold* is not bigger than *shares_number*, so to make the retrieval of the secret possible. Otherwise, we would have to plug in more shares than those that were generated. The *threshold* and *shares_number* have to be bigger than 0, otherwise the polynomial would have the grade -1, which would mean the polynomial is undefined. The *shares_number* has to have at least a share, so at least one person should receive the share in order to apply Lagrange's formula. The last condition prevents the program from going in an infinite loop. This has to do with how is the *create_x_coordinates* function defined. In order to create unique shares, we have to have a maximum value set to plug it into *secrets.randbelow*. Only if

the generated number is unique, it is then added to the *unique_number* set. If *MAX_BOUND* was less than *shares_number*, at some point the *unique_numbers* would have a number of *MAX_BOUND* numbers, therefore generating more numbers between 0 and the limit is futile, since each one of those generated numbers would already be in the set. In case the conditions are respected, which means that an input is not acceptable, an error will be raised and caught later on.

In the *API.py* we also have, for each of the cloud technology, a distribution and retrieval function. All of the distribution and retrieval functions, in general, have a similar behavior: after the databases is accessed, a share can be uploaded or downloaded, depending on the case.

- *distribution_firebase_api(key, app_name, share)* handles the uploading of a share in the Google Firebase database using the *key* and *app_name* to retrieve in the *db* variable, the client for accessing the specified database.

- *retrieval_firebase_api(key, app_name, share)* handles the downloading of the one share stored in the cloud database. The access to the database happens in the same way in *distribution_firebase_api*.

- *distribution_clevercloud_api(key, share)* handles the uploading of a share similar to Google's Firebase API, with the exceptions that *db* is a cursor for the database accessed trough the MySQL connection object returned in *db_connection*.

- *retrieval_clevercloud_api(key)* handles the downloading of the one share stored in the cloud database. The access to the database happens in the same way in *distribution_clevercloud_api*.

- *distribution_cosmos_api(key, share)* handles the uploading of a share similar to Google's Firebase API and CleverCloud's API, with the exceptions that *col* is an instance of MongoDB that access a specific table in the online storage.

- *retrieval_cosmos_api(key)* handles the downloading of the one share stored in the cloud database. The access to the database happens in the same way in *distribution_cosmos_api*.

Each of the cloud technology has a dedicated folder in which there is a file for configuring the connection to the cloud database and a file for handling the uploading and downloading of the share, with all of the auxiliary functions. The file with functionality for handling shares has the suffix "SSSS_" and the one for getting hold of an object referring to the database, has the prefix "_config". We will start with the configuration file.

– *initialize_firebase(key, app_name)* is a complex function that handles authentication to a specific Firebase database. The key is a dictionary that contains information for connection to a specific account and its respective database. *credentials.Certificate* has been imported from the *firebase_admin* library that handles authentication as an administrator. The *app_name* will serve as a point of reference between the first App instance and the rest that will be created after that. To note here is that an App instance will be present for the whole duration of the program, and "App instance" is not related to the main program, but is a Firebase's specific object. If no *app_name* is given when we create a second App instance, that is, when we plug in another certification for another account, the program will get confused and will throw an error saying that the first App instance, namely, the default App, is already running. One solution would be to stop the first App instance and start the second one, but that will result in lost connection to the previous database, and by the end of the execution of the whole application, only on the last App instance some changes will be made. As an example, if in this format we would like to use three different Google Firebase accounts in the same instance of the program, the first two will be ignored. The best solution to this problem in this case is to create for each of the account given another App instance that will run concomitantly in the whole program. Another reason for giving the App a name besides its implementation, is to refer to it later to specify to what account we should upload or download a share. With this implementation, the app names will be, in this order of creation: "[DEFAULT]", "app1", "app2", etc.

– *access_firebase(key, app_name)* is the first step in communicating to the Google Firebase database. In order to get the necessary client to access the database, the arguments needed are given and the *db* variable returned will help in simplifying the flow of the whole uploading or downloading process.

– *upload_firebase(db, share)* will take the client *db* as argument returned by the previous function *access_firebase* and the share to be uploaded. Before uploading the share, the cloud database will be erased of any other information. The data of the share, namely the x and y coordinates, has to be formatted as a dictionary and then uploaded to the cloud. This has to do with Google's own API. Also important to note is that the shares coordinates have to be converted to string, since the x and y will, in most of the cases, be extremely big and the Firebase's API has limitation on the max of integers. The data will be stored in a folder on cloud named "Shares" and a file with an automated index will contain the dictionary.

– *cleanup_firebase(db)* will erase the information already on the "Shares" folder in the cloud. First recurrent step in uploading a share in any cloud database is the erasing step. We have to make sure that each cloud database holds only one share, for the fact that this is Shamir's SSS where one participant holds only one share.

– *download_firebase(db)* will interrogate the cloud databases for the files in the "Shares" folder. When we retrieve the share we have to make sure the coordinates are integers. The share is put in a *share_list* but only the one element will be returned. Although, the retrieval process could have been simplified, this makes room for improvements in case the databases are to hold more shares, and the downloading of the share will take place in accordance with the name of the share or the date of when it was uploaded to correspond to a specific distribution.

As with Google Firebase, we will start explaining the authorization to the cloud service Clever Cloud by presenting the configuration file.

– *initialize_clever(key)* takes a key as a list, and each element of the list is a dictionary that contains information pertaining to the database characteristics. The reason why, just some of the fields of the key are accessed is due to two reasons: ease of use and sufficient information. When the participants takes the key from the Clever Cloud web page we do not want the user to manually pick the fields, but rather to just copy paste all of the access data. Other details about the access data are not necessary, therefore not directly picked out.

The following functions are located in the *SSSS_Clever.py*. In Clever Cloud we selected our preferred database as MySQL, therefore we would have to employ the specific SQL syntax.

– *access_clevercloud(key)* is the first function called when we want to use CleverCloud's cloud services. *db* is a cursor of the database accessed through the MySQL connection object. returned in db connection.

– *upload_clevercloud(db, db_connection, share)* works as such: the data on a specific table is destroyed, a new one is created using the MySQL syntax having the columns x and y as strings, the data is inserted into the databases, and finally the changes are committed using the MySQL connection object.

– *cleanup_clever(db)* drops the table with all the rows inside, if it exists.

– *download_clevercloud(db)* selects all the shares from the "shares" table. In order to read it one by one, *fetchall* is employed. After reading the string values of x and y, they are converted to int and appended to a list. Once again, we return only the first and only share.

The last cloud service introduced is Azure Cosmos DB using the MongoDB type of database.

– *initialize_cosmos(key)* takes a key as a dictionary with a single item. After the value of the key *URI* is retrieved, using the *MongoClient* class, we can extract a client that will give us access to all of the databases inside. By using the client we access the database *ShamirSecretSharing* and get the cursor for the table *Share*.

Further we will present the functions inside *SSSS_Cosmos.py*, very similar to Firebase.

– *access_cosmos(key)* simply gets the cursor of the table *Share* by accessing *initialize_cosmos* with the key provided as argument.

– *upload_cosmos(col, share)* clears the databases of any other rows present preparing the new share to be uploaded. The share is transformed into a dictionary, and once again, the values of the coordinates, x and y, are transformed into strings.

– *cleanup_cosmos(col)* takes every row in the table, gets evey *_id* for identifications and deletes it one by one.

– *download_cosmos(col)* takes every row in the table, and gets the value of the last share, although there is supposed to be only one share.

Finally, we will talk about the *GUI.py* file, the most generous part of the application in terms of number of functions. Everything related to user interaction happens here, hence the use of Tkinter library. The code always executed when the file is run will create a new Toplevel widget, basically a main window such with any other Windows application. The logo, size of screen taken and the title will also be set here. Also, an error catcher function will be set to run every time an error is thrown that is not caught by the other predefined functions.

A frame contains everything that happens inside the window. A *Frame* object will organise the widgets in a table like orientation. Each time the application gets from a frame to another supposed frame, the actual frame which is passed as argument is deleted and a new one is created. When we will place a widget inside the cell we will specify the row and column, along with some other information for a better style. At every new frame we will have a title. Most of the time we will have a label under the title that shows instructions.

– *start_gui()* is the first function that handles a frame. The user can choose to follow two paths: distribution or reconstruction of the secret by clicking on one of the two buttons. Each button will call an anonymous function when the button is clicked.

– *distribution_input_gui(frame)* is the frame in which information will be collected from the dealer, necessary for the distribution of the secret. The text boxes in which the user puts the information are *Entry* objects. From this point onward we will pass the secret, threshold and total share number as a list of arguments when calling the next functions. After clicking the button displayed, a function will be called first in order to check if the information received does not make sense in our conjuncture. For example, we check to see if the shares number is bigger than the threshold, using *check_threshold_shares* present in the *API.py* file.

– *choose_databases_gui(frame, list_of_arguments)* is a joint function that will also be called in the reconstruction stage. Here the user can select from the list of checkboxes which databases are needed. After they hit the confirmation button, the button will change its text instruction and the check boxes will be disabled. Now the user has to put in each entry a number of databases that the participants have. When hitting the button now, the function *equal_shares_number* will check to see if the number of shares selected is the same that was entered when the function *distribution_input_gui* was called. If the numbers are equal, *access_databases_gui* is called.

– *access_databases_gui(frame, list_of_arguments, shares_proportions)* is a function that changes the frame to the one manipulating the keys for the cloud database or the local shares. Based on the number of selected databases and their types, *entries_creator* is called to generate a button and label per cloud database. After all the cloud database keys have been introduced the confirmation button will call *distribute_shares*.

– *distribute_shares(main_frame, list_of_arguments, button, cl_number, labels_fb, labels_cc, labels_co)* is an intermediary function that will call *input_api* from *API.py* that will return the share list. In case there are any local shares that have to be distributed, the function will redirect to another function *show_clear_shares* in which the user copies the displayed shares. Otherwise, the function redirects to *sent_confirmation_gui*.

– *sent_confirmation_gui(frame)* functions confirm with the user that the shares have been distributed. The button displayed will redirect to the start frame.

– *reconstruction_input_gui(frame)* is the frame in which the threshold number must be given so the reconstruction process can start. When the confirmation button is clicked, it will call *check_threhsold* for checking if the threshold number is an integer bigger than 1 and if everything is in order, *choose_databse_gui* is called, the same function as in the distribution phase where the databases keys are provided.

– *reconstruct_secret(frame, list_of_arguments, button, cl_number, labels_fb, labels_cc, labels_co)* will take the shares from the cloud databases with the given keys as files and if there are local shares to be given by the user, the function will redirect to *distribute_local_gui*. Otherwise, the function will redirect to the last frame by calling the *secret_reconstruction_gui*.

– *secret_reconstruction_gui(frame, list_of_shares)* will call *reconstruction_api* from *API.py* with the list of shares as argument. The secret will be displayed after it is converted from an integer to the initial input given.

– *delete_frame(frame)* is used every time a frame changes most of its elements. All of the widgets are called one by one and destroyed. After that, the frame in which the widgets resided is also destroyed. In this way, errors are avoided and the memory stack is clean.

– *browseFiles(list_labels, index_label)* will be called when a key for accessing the databases will be introduced. The way this function works is by opening a Windows File Explorer in which the user has to select the predefined .JSON file. The user can change the type of the file it can select, like with any ordinary File Explorer. After the file is "opened", the name of the label on top of the button changes with the path of that file. Proceeding this way, when all of the keys have been introduced, we have a way of memorizing the paths to the keys needed.

– *add_scrollbar(frame)* is a function for adding a scroll bar when needed. This is not a simple task, and one of the most elegant way to achieve this is by creating a frame, in that frame a canvas to which the scroll bar is added in addition to a second frame to which the widgets will attach. We also configure the scroll bar to move along with the view of the canvas by binding a function to the upside-downside command.

4.4 Manual

This manual is a comprehensive guide on how to install and use the application. We will start with a quick note about the recommended system requirements followed by the installation of the program, and finally how to use the full capabilities of the application.

Recommended system requirements

The following specification are just recommendations, the program is likely to run of lower settings: 64-bit Windows 10, Intel® Core™ i5-8250U Processor and 4 GB

RAM available, 80 MB disk space. Due to the fact that the application uses Windows based libraries and tests on other operating systems have not been conducted, the program may or may not work on Linux, Mac OS, etc. One very important note is that Python is not needed.

Installation

After downloading the *Shamir_Secret_Sharing_Application.exe*, the user has to run the file by double clicking on the icon.

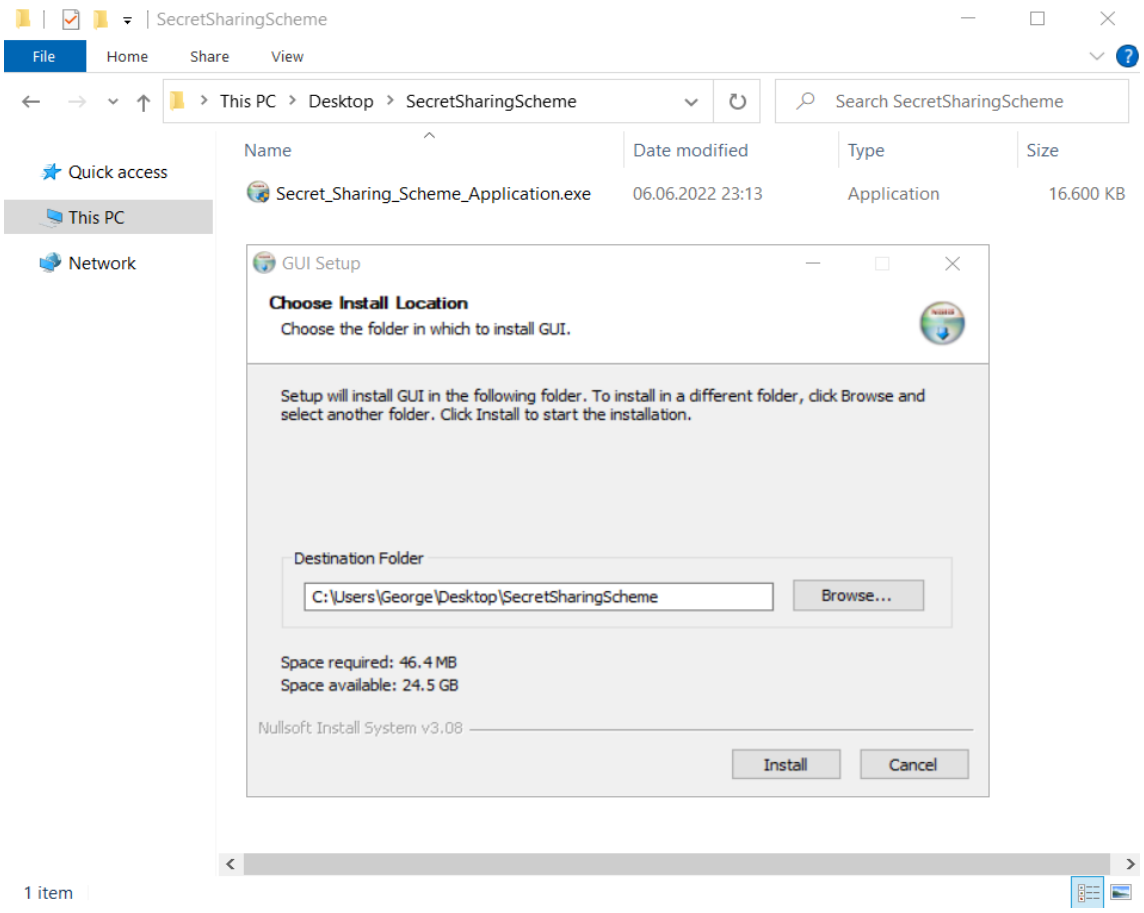

Figure 4.1: Installation step - selecting destination folder

The user has to choose the destination where the folder GUI will be installed. As a default, the path will be automatically set to the folder in which the .exe file is located. After the path has been selected, the user has to click on the "Install" button.

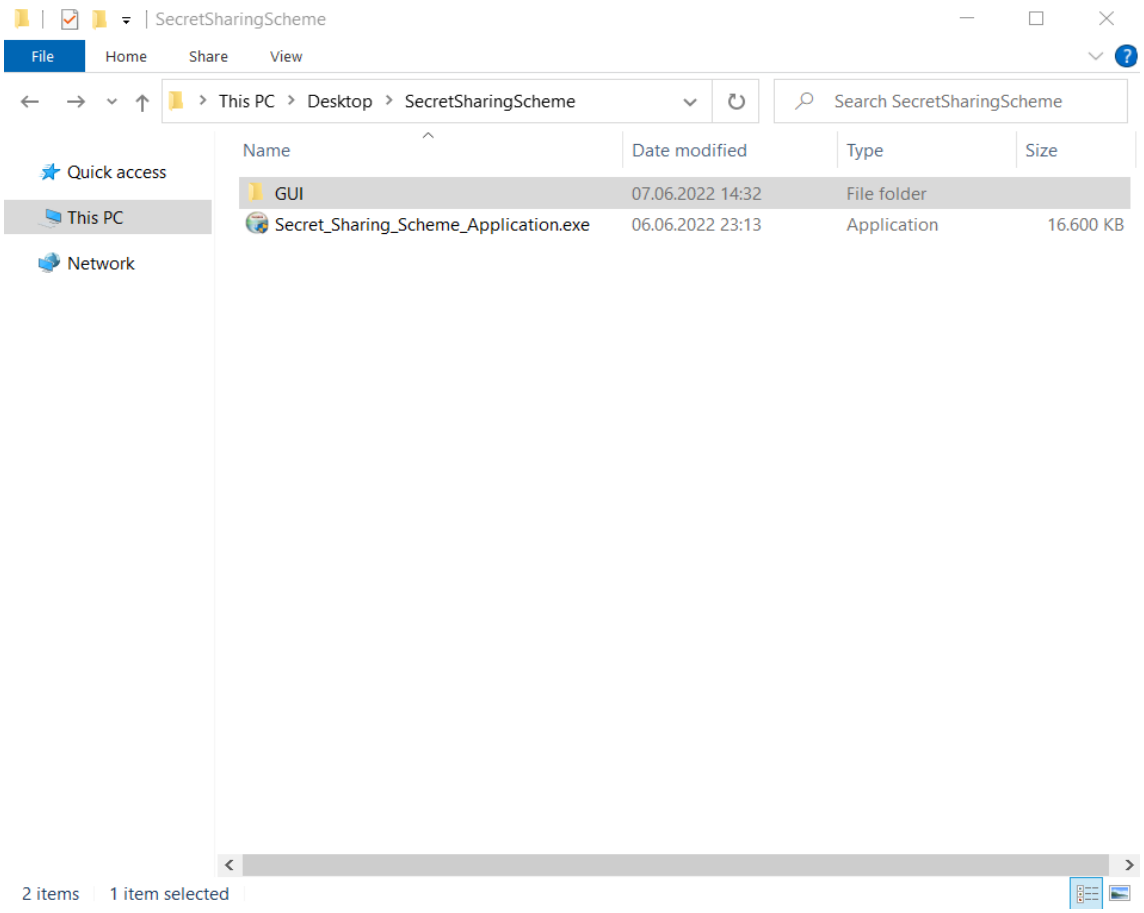

Figure 4.2: After installation

A folder named GUI will appear. In it, there are the requisite files for launching the application.

Usage

In this section we will discuss about how to use the application. The following information is intended to instruct the dealer. Derivations from this guide can lead to error, but all of them will be alerted by o pop-up screen with a short description about what has happened.

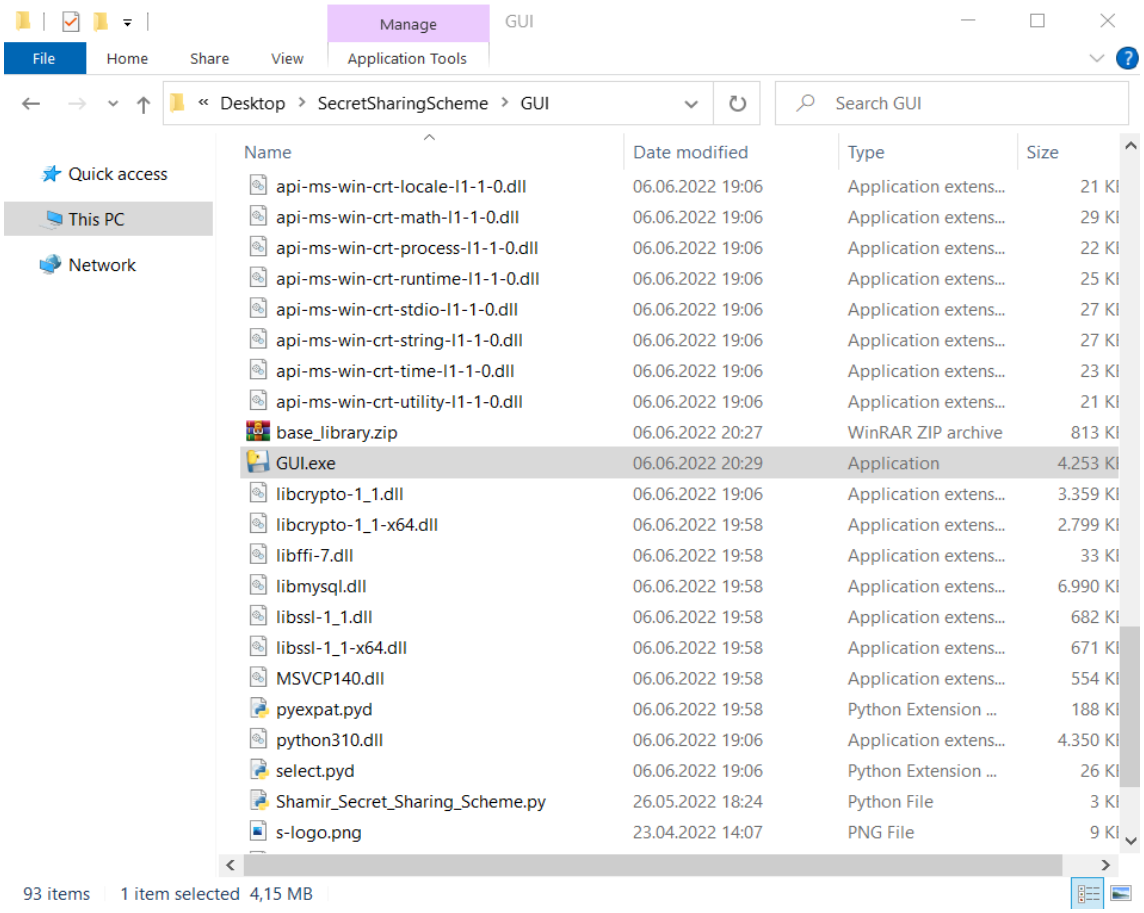

Figure 4.3: Starting the application

In the installation folder *GUI*, there is a file *GUI.exe*. The user has to click on the *GUI.exe* file to start the application.

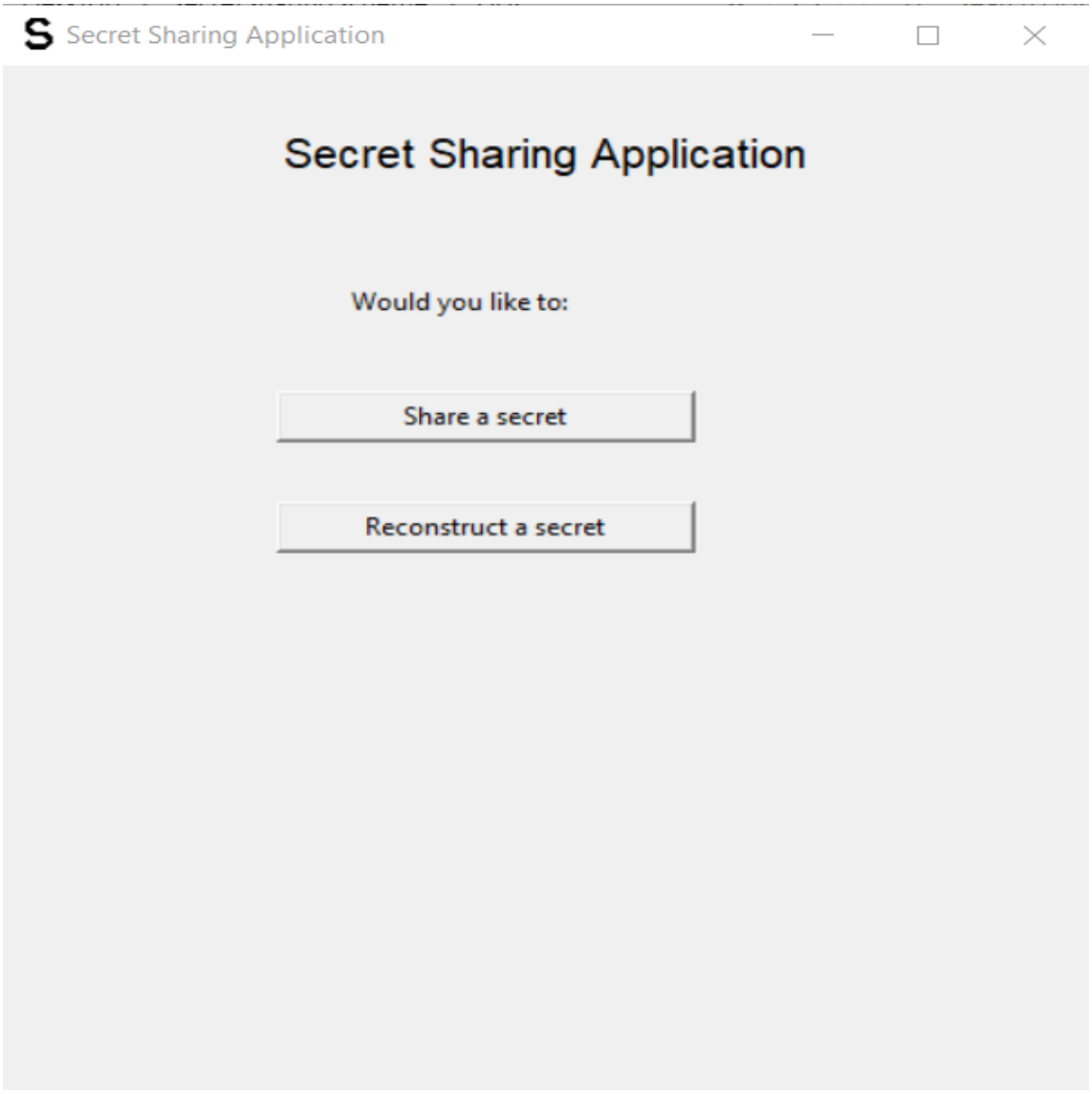

Figure 4.4: Start screen

On every step of the application there will be a title suggesting at what step are we at and/or an instruction on how to proceed further. Here we can see that we can click on the two buttons to either distribute (share the secret) or reconstruct the secret. We will start by clicking on the sharing button. This will sent us to another view of the application.

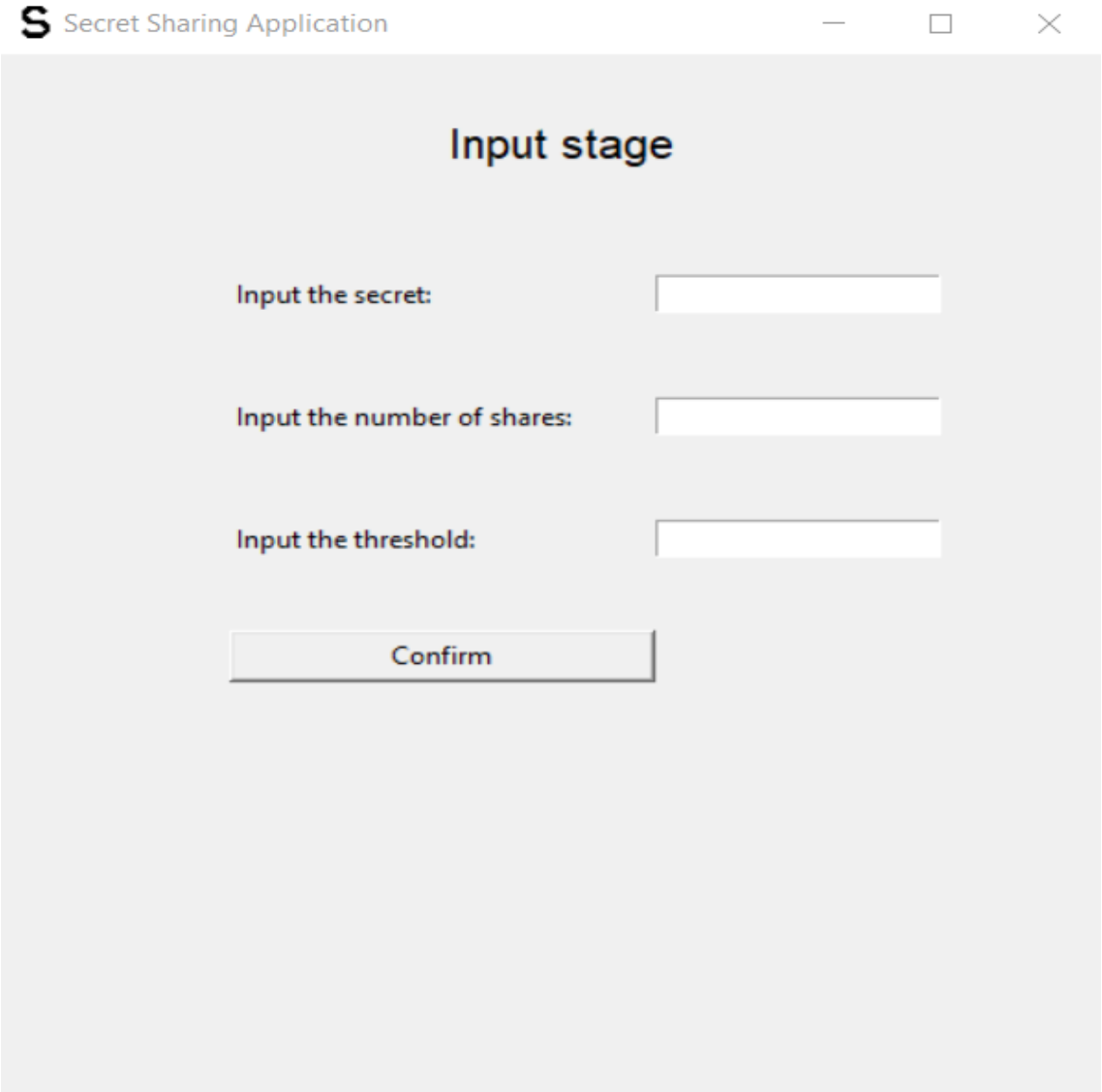

Figure 4.5: Input phase of distribution

We are now in the input phase of distribution. We will put the secret in ASCII format in the white text area right next to the label "Input the secret". One important note here is that the secret does not have to be, necessarily, a number. We will plug in the number of shares, that is, the number of participants in the second text area. The maximum number of participants is 64. The threshold number has to be lower or equal to the number of shares in the second field. The most common errors such as this will be displayed with a warning pop-up. If peculiar input is plugged in, such as a string in the number of shares text area, an error pop-up will appear.

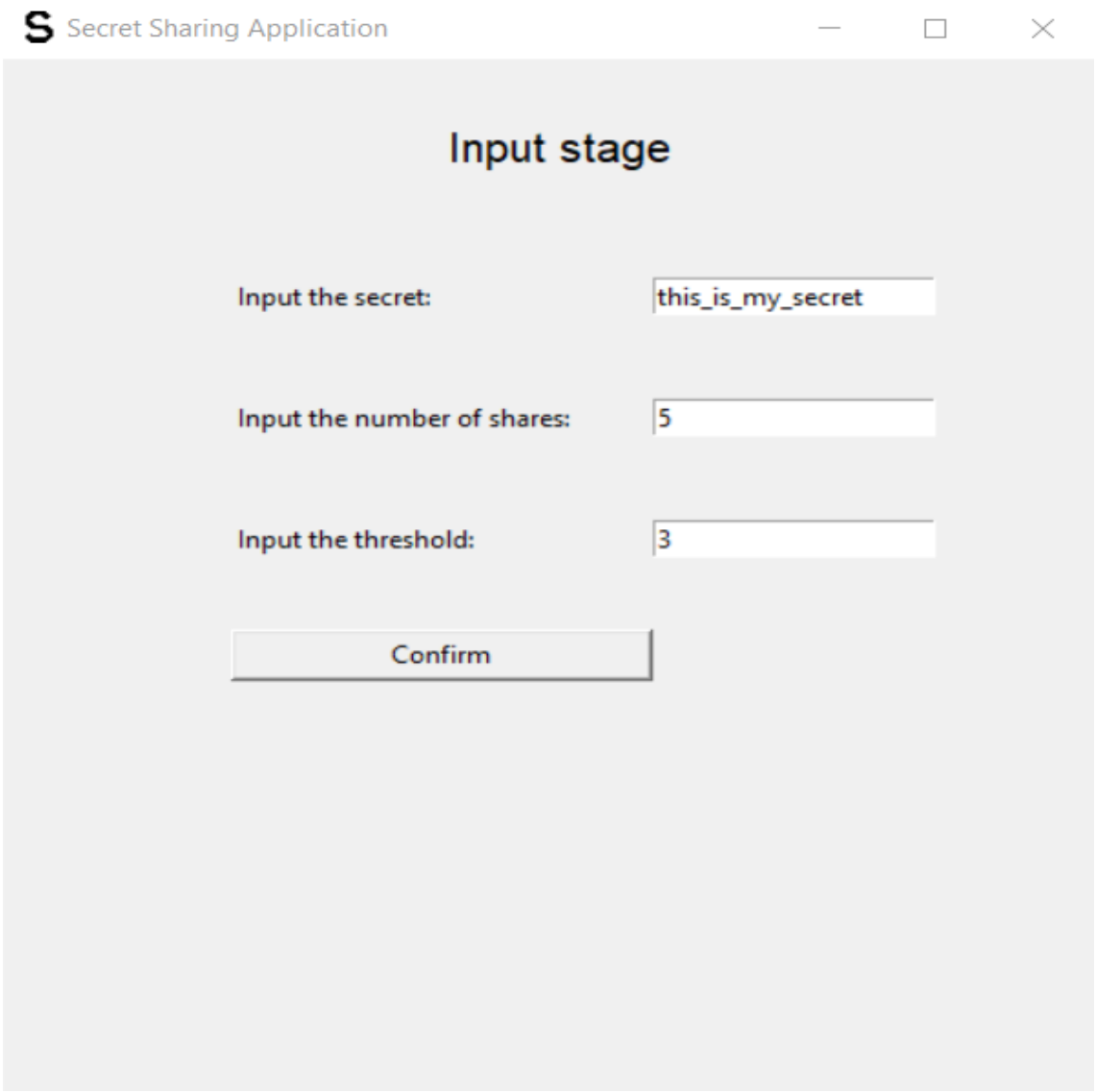

Figure 4.6: Input phase of distribution

For the purposes of comprehension, we will proceed explaining by choosing this example with the secret as "this_is_my_secret", number of total shares is 5 and threshold 3. After clicking on the confirmation button, we will have to select the databases.

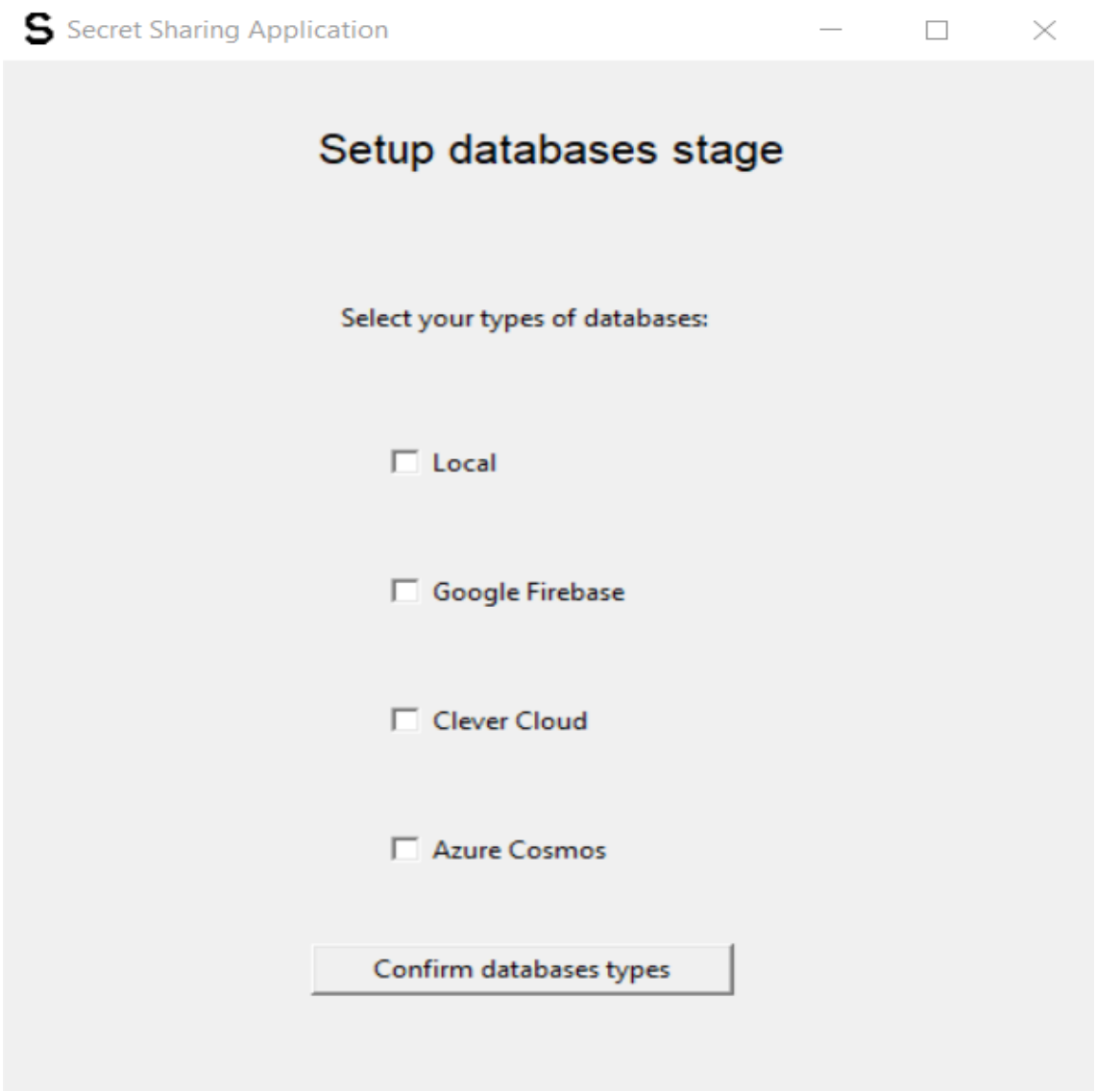

Figure 4.7: Setup stage of the databases

At this stage, we will select our desired databases types where the shares will be stored. The local database option does not require an internet connection. For the purpose of exploring all the features, we will select all of the four types. After clicking on the confirmation button, the frame will slightly change.

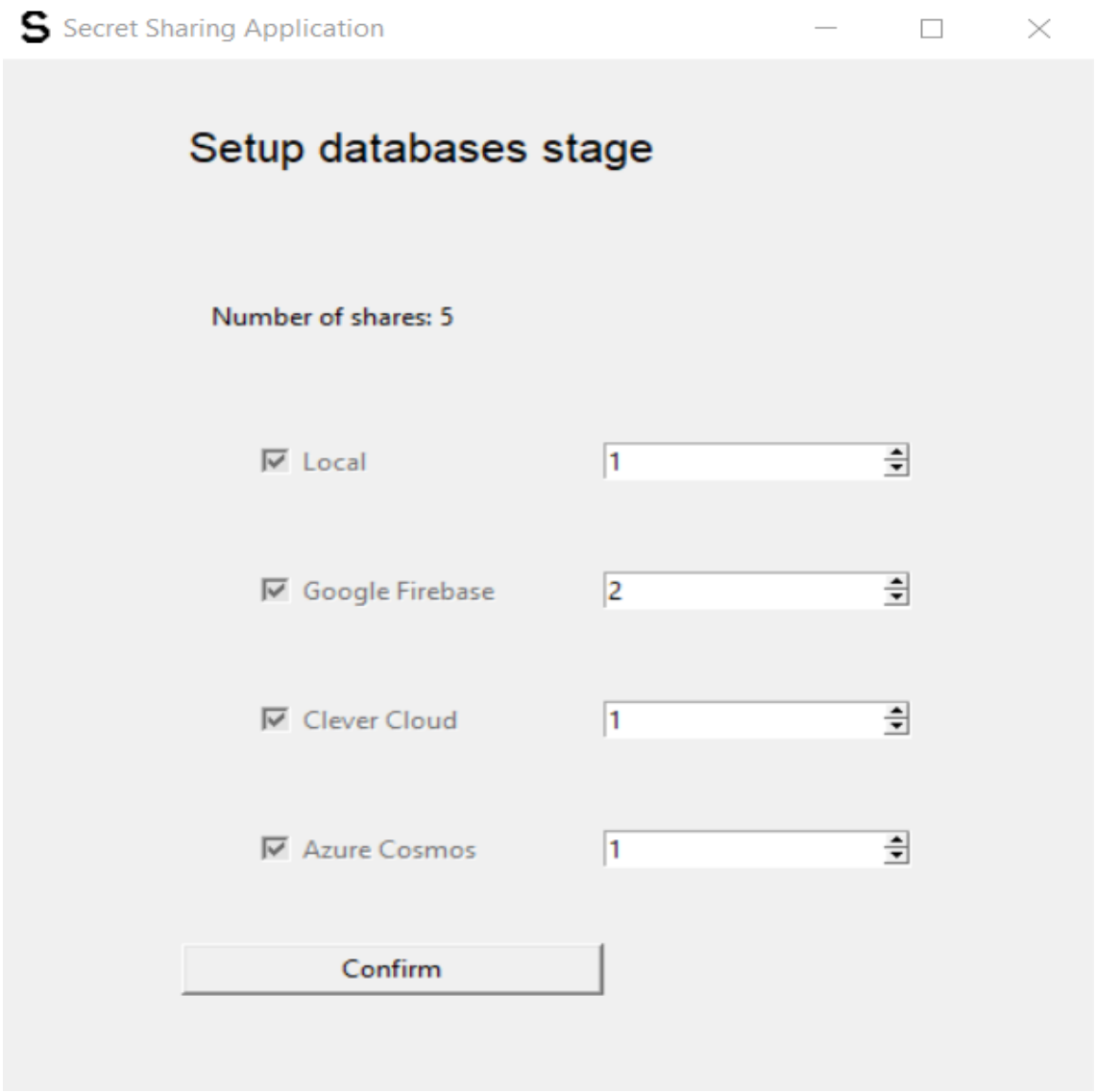

Figure 4.8: Selecting 5 databases

After selection, if a checkbox was filled, a spin box on the right part of frame will be generated. The default value is 1 and the number can be changed by clicking on the upper arrow/lower arrow of each spin box or by simply typing the number. Since a usual mistake is clicking the confirm button before selecting an exact number of shares with the one displayed, it will result in a new label under the "Number of shares". The label will display "Your number of shares is: " and the sum of shares selected, hinting that some other combination of numbers should be used. We will select two Google Firebases databases, and each one of the others.

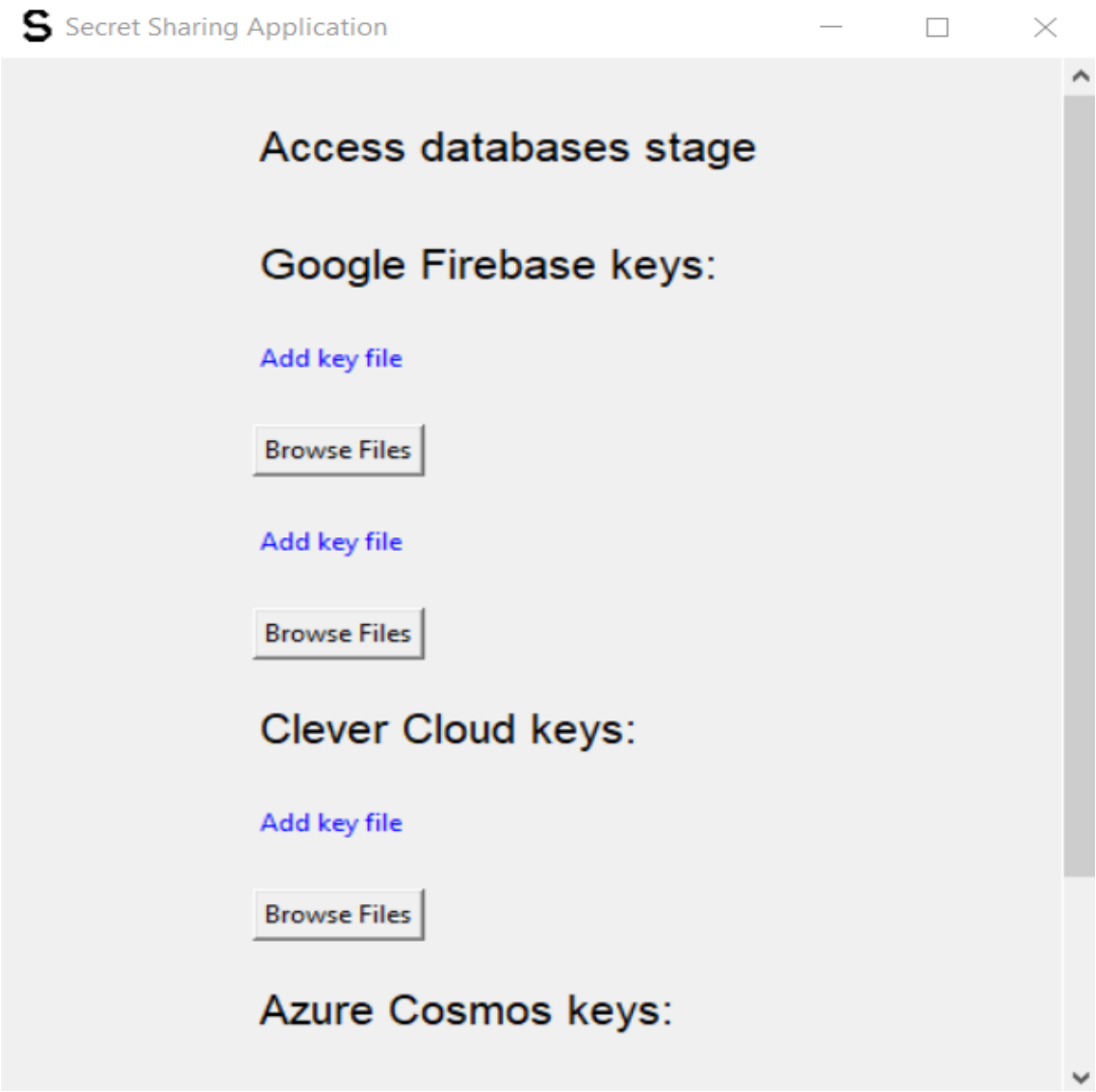

Figure 4.9: Empty selection slots

We now have to click on each of the buttons to browse for files and select each of the cloud database keys.

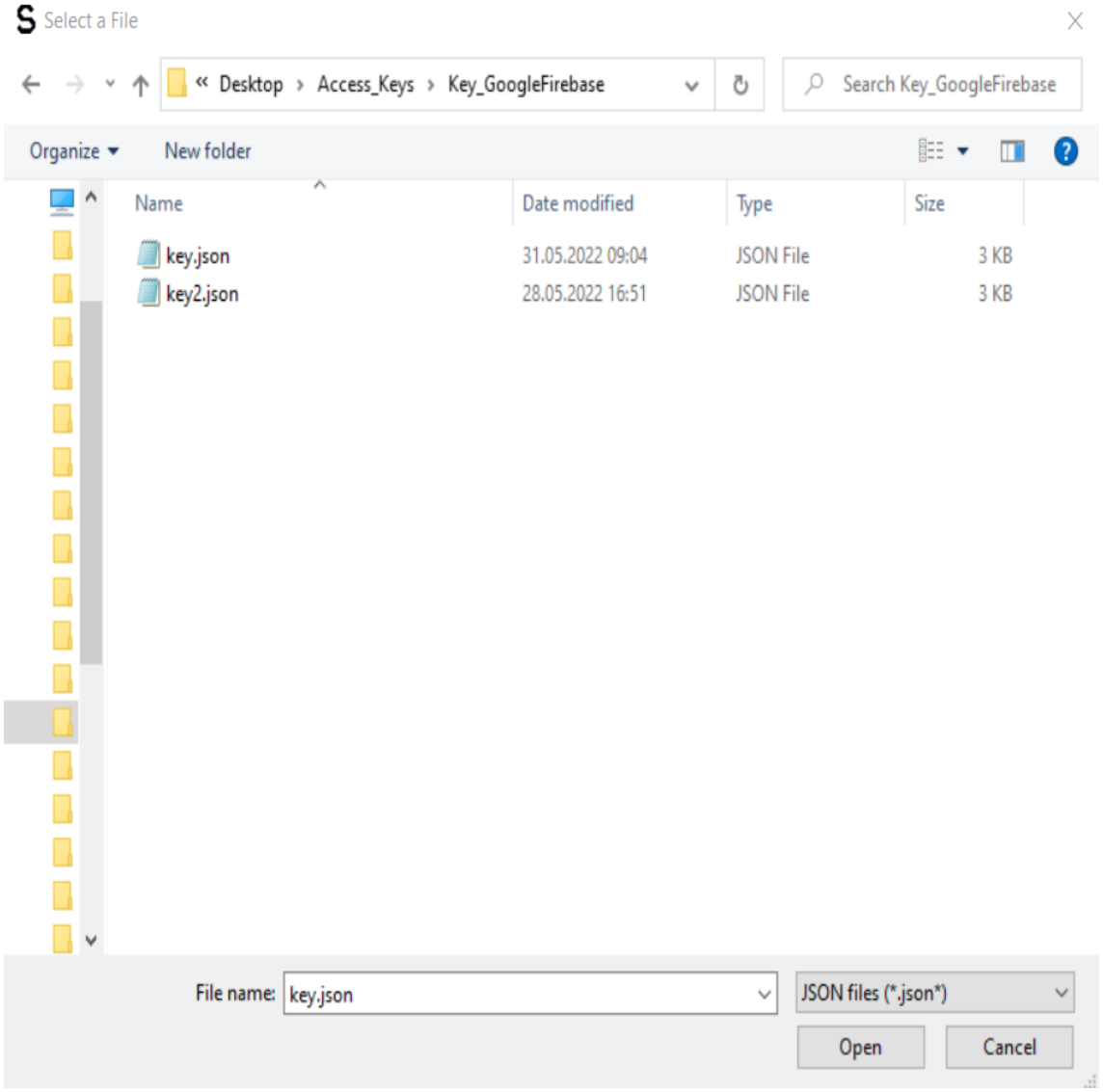

Figure 4.10: Selecting the key file

After selecting our file we click on the "Open" button.

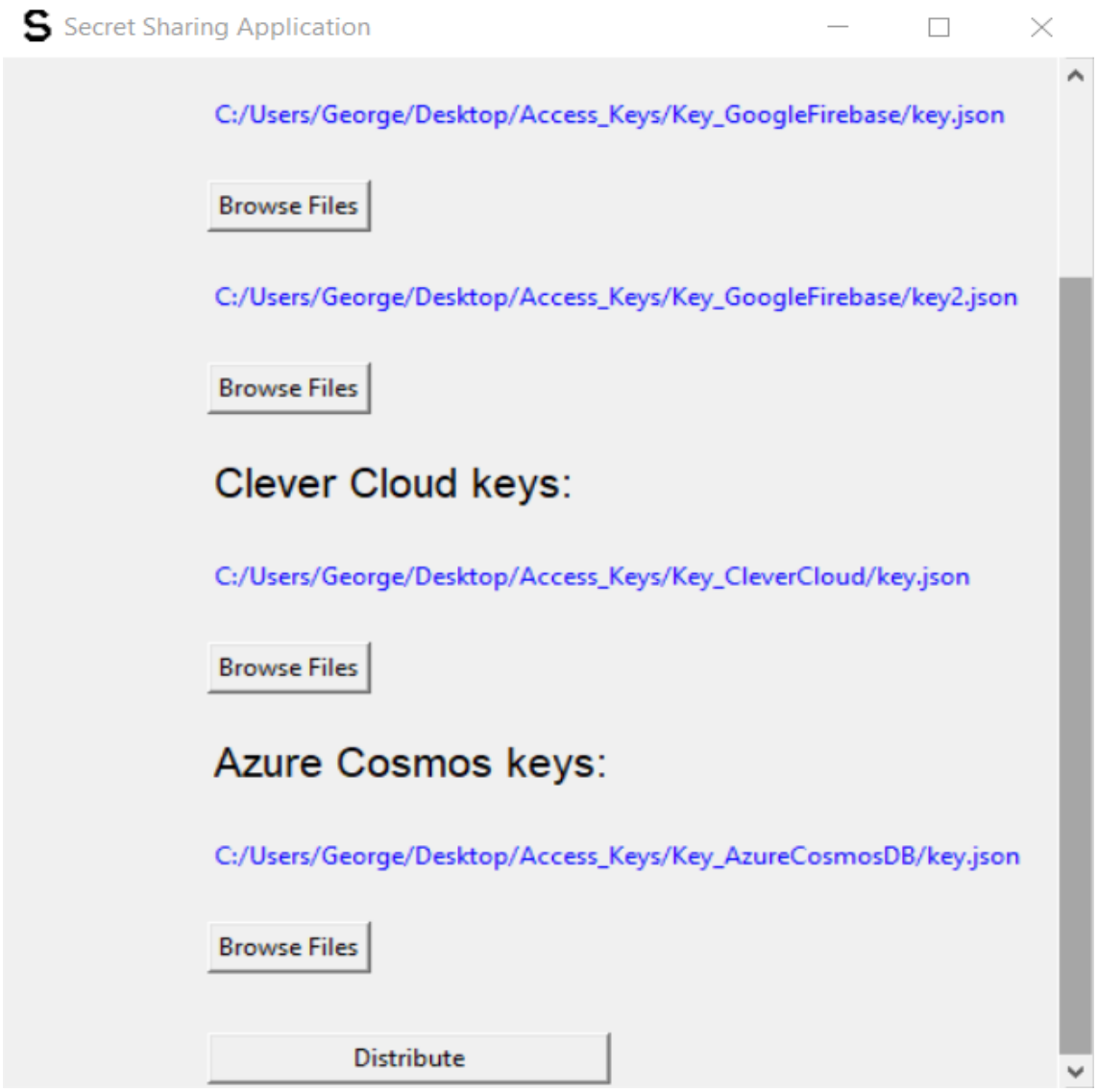

Figure 4.11: Selected cloud databases keys

Each file opened will appear as a path selected to that file, on top of its respective button. To note here is that each key should be unique, otherwise the reconstruction of the secret might not work due to having just one key instead of more. Clicking on the distribution button will forward to the next frame, where we will get the one of the shares.

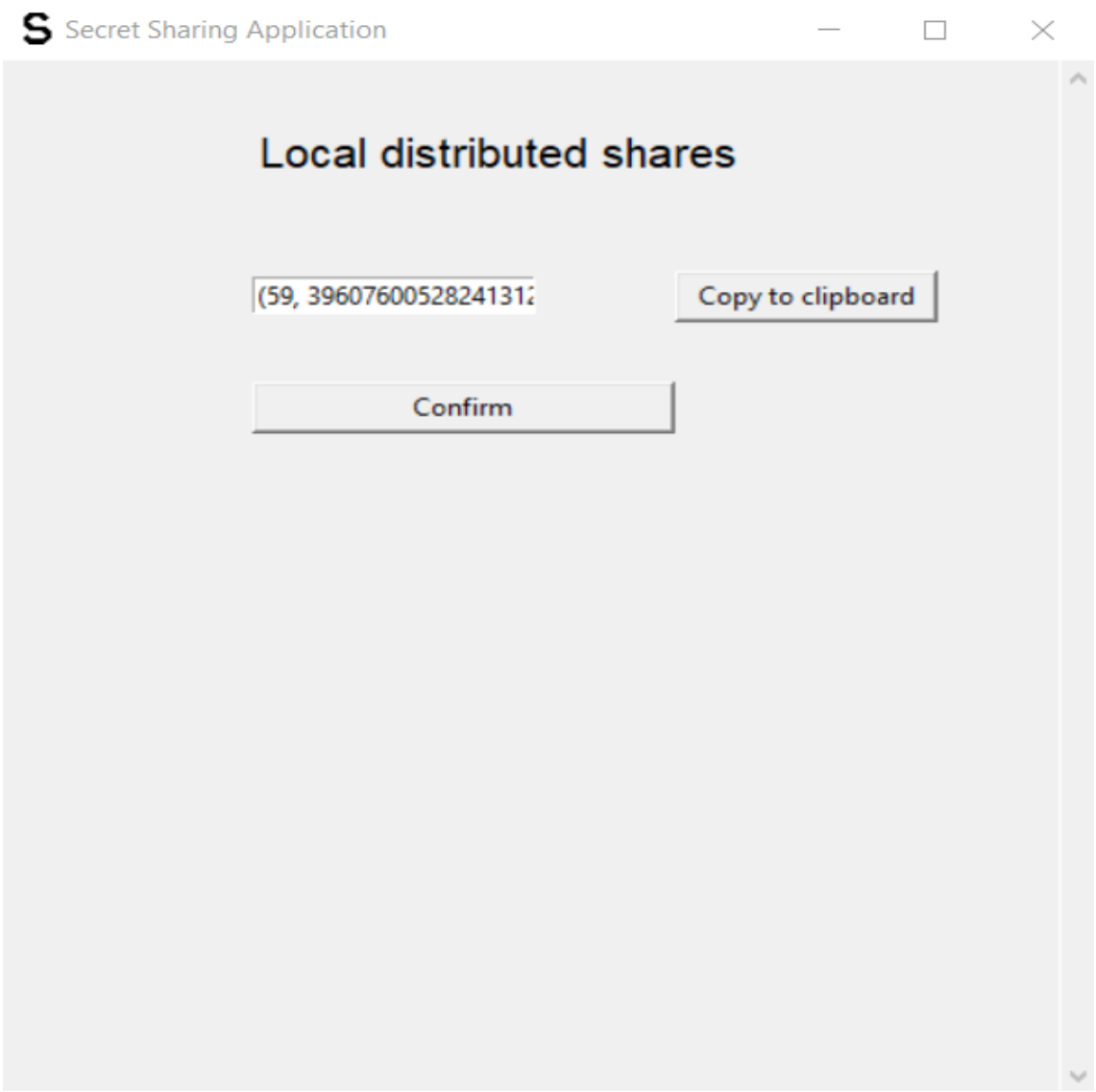

Figure 4.12: Local share selection

Now the dealer will have to copy the share and distribute it to its respective participant and it is up to them to decide which method of sending this information is best. The dealer can use the button for copying the share to the clipboard, overwriting the existing content.

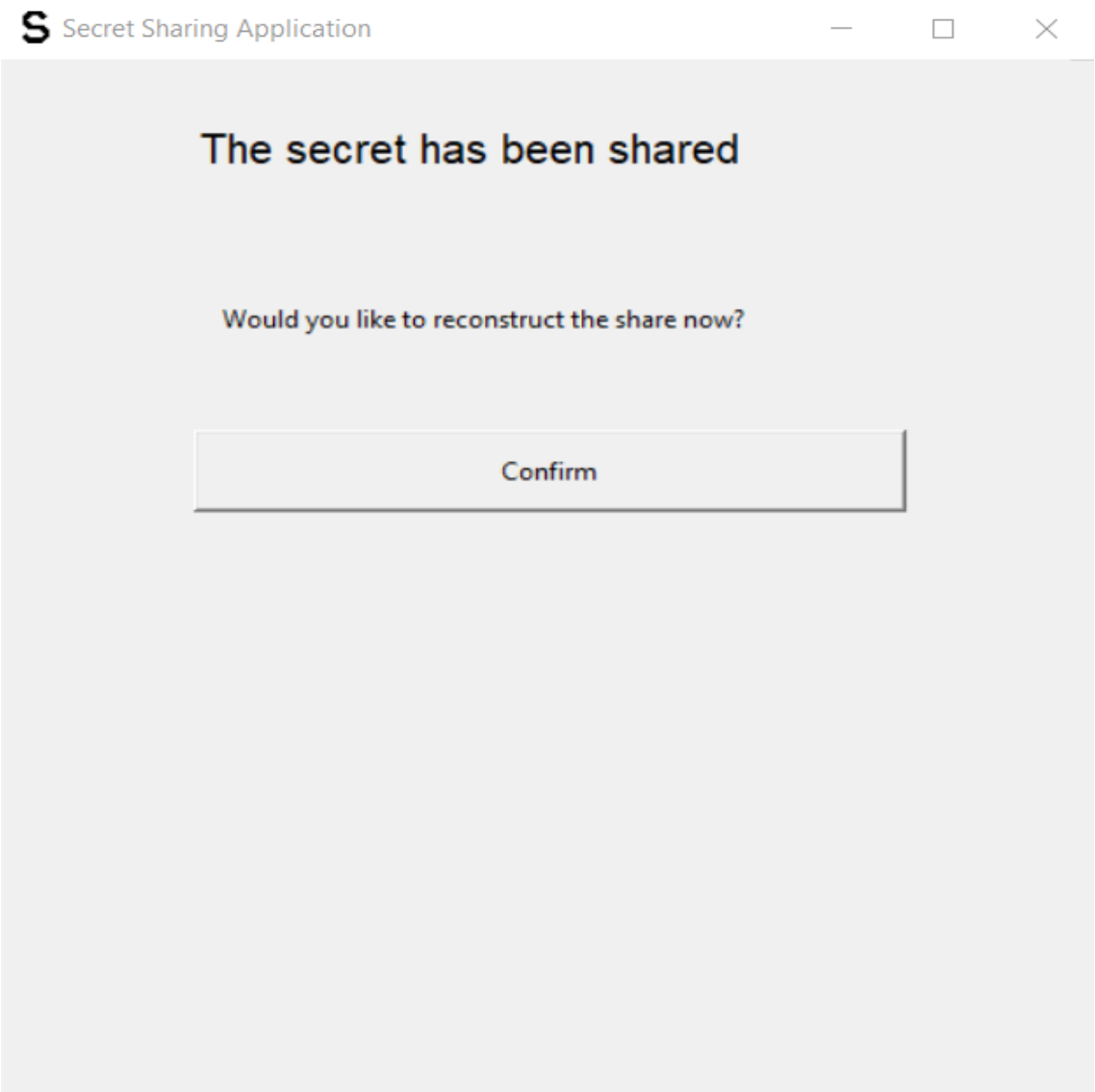

Figure 4.13: Finished distribution frame

After the shares has been distributed, a button that will lead to the start of the application will appear. This has been created for ease of access since the application does not have a back button, so there is no need to close the application in order to restart the distribution process or to reconstruct the secret now.
At the start of the application we will select the button "Reconstruct the secret".

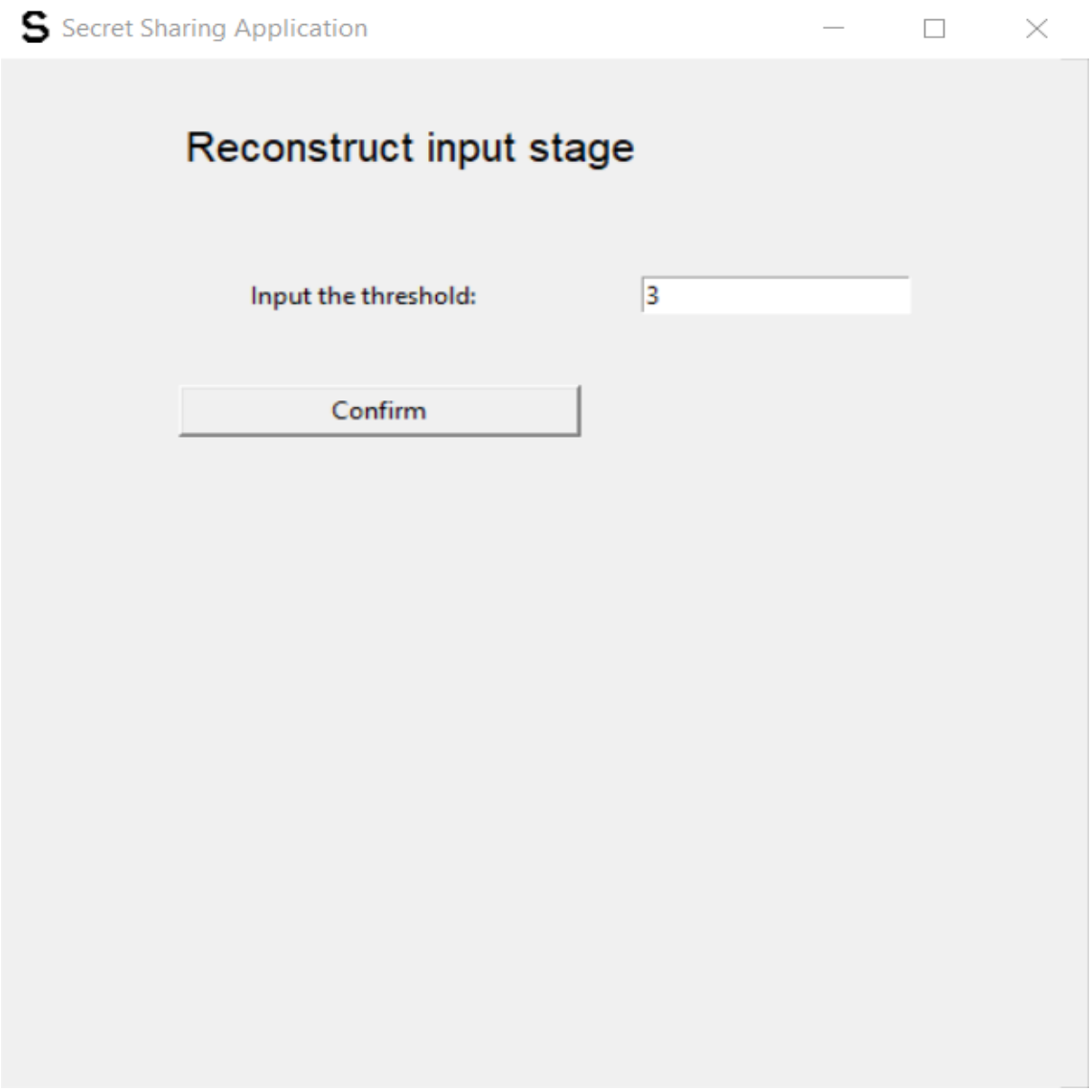

Figure 4.14: Input phase of reconstruction

Since for reconstruction we only need to know what is the minimal number of shares for making the recovering of the secret possible, we will start by plugging in the threshold. From this point forward we will follow the same steps starting with the setup stage of the databases.

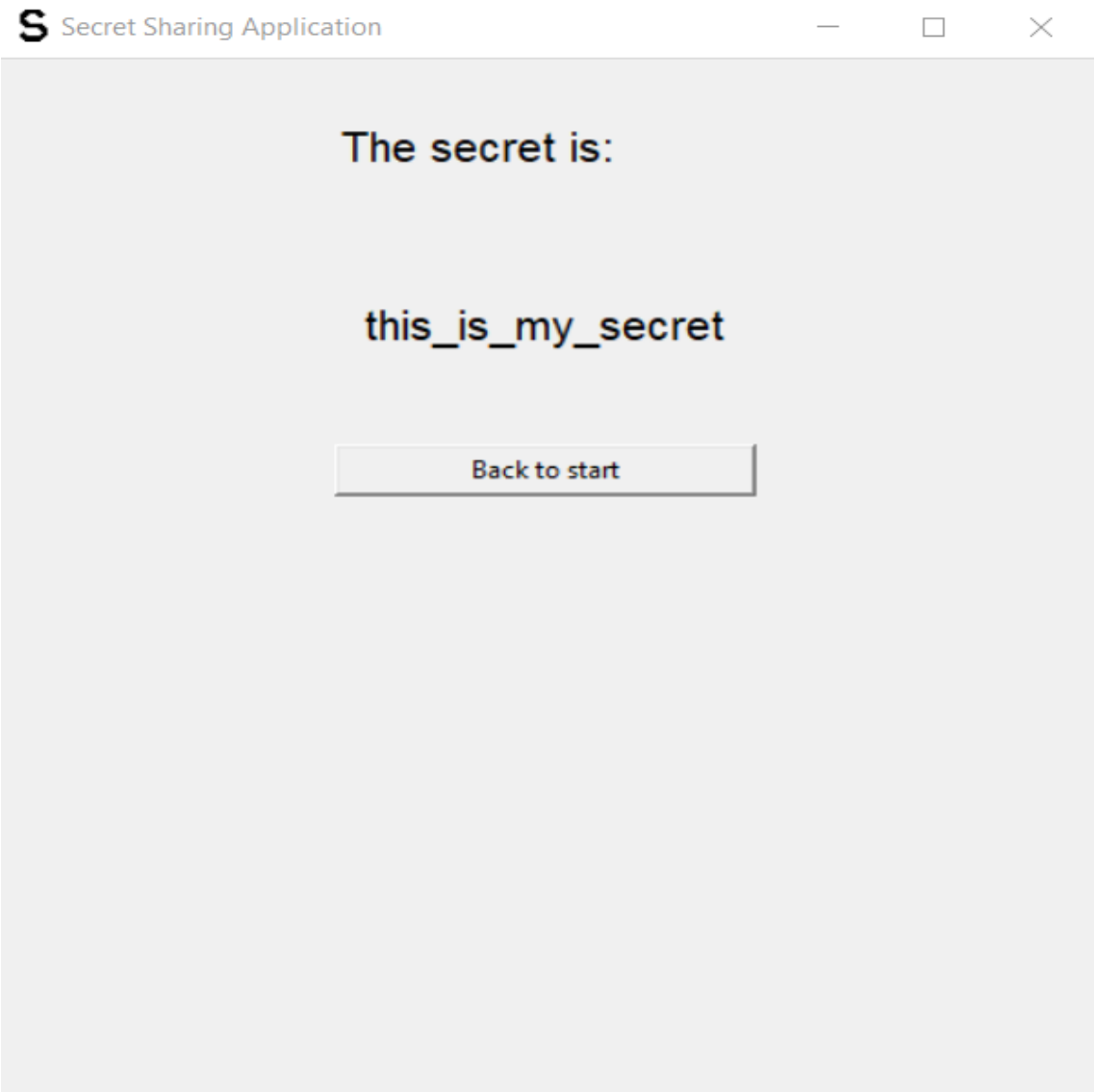

Figure 4.15: Secret revealed

After selecting our types and number of databases, the key files for the cloud stores and plugging in the last local share, the secret will be reconstructed. The button displayed will get us back to start.

Accounts and keys

In order to use the cloud databases in the application, accounts must be created, the databases must be set up, and also keys must be placed in the form of files .JSON or .TXT.

For Google Firebase, the following steps are to be done:

1. Access the Google Firebase. Authentication with a Gmail account is required.
2. Access the console directly.
3. Add a new project by providing the name and other information required.
4. Navigate to the created project, click on "Firebase Database", create a database in production test mode. Select the location where the data will be stored.
5. In that database, create a new collection by clicking "Start collection". Collection ID must be set to the name "Shares" (without quotation marks). As a first document, choose "Auto-ID" option. Click on "Save". As a side note, the user can update the rules of access to the database by selecting "Rules", in case the access is cut off after some time.
6. While in the project folder, click on the gear icon right next to the "Project Overview" button. Select "Project settings".
7. Click on "Service accounts". Generate a private new key. This key will be distributed to the dealer.

For Clever Cloud, the instructions are:

1. Access Clever Cloud. Authentication is required.
2. Click on "Create". Select the add-on option. The MySQL option is needed.
3. Select the "DEV" plan. Give a name and a location of the database.
4. Click on the database name on the left side and click "Information". At "Environment variables" click on "JSON". Copy the contents in a .JSON file. That will be the key to be distributed to the dealer.

Lastly, the instructions for Azure Cosmos DB are:

1. Access Azure Portal. Authentication is required. Search for Azure Cosmos DB.
2. Click on "Create database" and select "Azure Cosmos DB API for MongoDB".
3. Type all the required information, such as a name for the resource group and the name of database.
4. Inside the database, select "Quick start" and copy the content from the "Python" section, namely the "PRIMARY CONNECTION STRING"
5. The owner must include an open curly bracket right at the start of the key file and close it right at the end. In those brackets, the owner of the key has to write the following: "URI": "[KEY]", and replace [KEY] with the connection string copied. The file will be sent to the dealer.

Chapter 5

Conclusion

In this paper we have gained some insight on some of the most known SSSs and few connections between them, in order for a better understanding of what is the underlying theoretical security behind some real world problems, research and inspection of SSSs and cryptography overall. In order to improve this survey, more details and observations can be introduced to better explain the current SSSs. Also, more SSSs can be added to give a better overview on the family tree of SSSs.

Acknowledgement

This work could not have been finished without the thorough assistance and guidance of Associate Professor Ruxandra F. Olimid and Assistant Professor Adela Georgescu.

Bibliography

[1] Amos Beimel. Secret-sharing schemes: A survey. In *International conference on coding and cryptology*, pages 11–46. Springer, 2011.

[2] Amos Beimel, Aner Ben-Efraim, Carles Padró, and Ilya Tyomkin. Multi-linear secret-sharing schemes. In Yehuda Lindell, editor, *Theory of Cryptography*, pages 394–418, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[3] G. R. Blakley. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*, page 313, Los Alamitos, CA, USA, jun 1979. IEEE Computer Society.

[4] G R Blakley and Catherine Meadows. Security of ramp schemes. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, page 242–268, Berlin, Heidelberg, 1985. Springer-Verlag.

[5] Carlo Blundo, Alfredo De Santis, and Ugo Vaccaro. Efficient sharing of many secrets. In *STACS*, 1993.

[6] Ernest F. Brickell. Some ideal secret sharing schemes. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 468–475, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[7] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395, 1985.

[8] László Csirmaz. Secret sharing schemes on graphs. *IACR Cryptology ePrint Archive*, 2005:59, 01 2005.

[9] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 307–315, New York, NY, 1990. Springer New York.

[10] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[11] Oriol Farràs, Jaume Martí-Farré, and Carles Padró. Ideal multipartite secret sharing schemes. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, pages 448–465, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[12] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, pages 548–566, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[13] Yotam Harchol, Ittai Abraham, and Benny Pinkas. Distributed ssh key management with proactive rsa threshold signatures. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security*, pages 22–43, Cham, 2018. Springer International Publishing.

[14] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *annual international cryptology conference*, pages 339–352. Springer, 1995.

[15] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan Part Iii-fundamental Electronic Science*, 72:56–64, 1989.

[16] Mahabir Prasad Jhanwar. A practical (non-interactive) publicly verifiable secret sharing scheme. In Feng Bao and Jian Weng, editors, *Information Security Practice and Experience*, pages 273–287, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[17] R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24(9):583–584, sep 1981.

[18] Tal Rabin. Robust sharing of secrets when the dealer is honest or cheating. *J. ACM*, 41:1089–1109, 1994.

[19] Adi Shamir. How to share a secret. *Commun. ACM*, 22:612–613, 11 1979.

[20] Victor Shoup. Practical threshold signatures. *EUROCRYPT 2000. LNCS*, page 14, 11 2000.

[21] D. R. Stinson and S. A. Vanstone. A combinatorial approach to threshold schemes. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 330–339, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[22] D.R. Stinson and R. Wei. An application of ramp schemes to broadcast encryption. *Information Processing Letters*, 69(3):131–135, 1999.

[23] Tamir Tassa. Hierarchical threshold secret sharing. *Journal of Cryptology*, 20:237–264, 01 2007.

[24] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction for algebraic block codes, U.S. Patent US4633470A, Dec. 30, 1986.