Shamir’s secret sharing

Cryptocurrency And Blockchains: Mathematics And Technologies

Sushmanth Kakulla, Rex Liu, and Paul Wawszczyk

Problem

Eleven scientists are working on a secret project. They wish to lock up the documents in a cabinet so that the cabinet can be opened if and only if six or more of the scientists are present. What is the smallest number of locks needed? What is the smallest number of keys to the locks each scientist must carry?

Liu, C.L. Introduction to Combinatorial Mathematics. McGraw- Hill, New York, 1968.

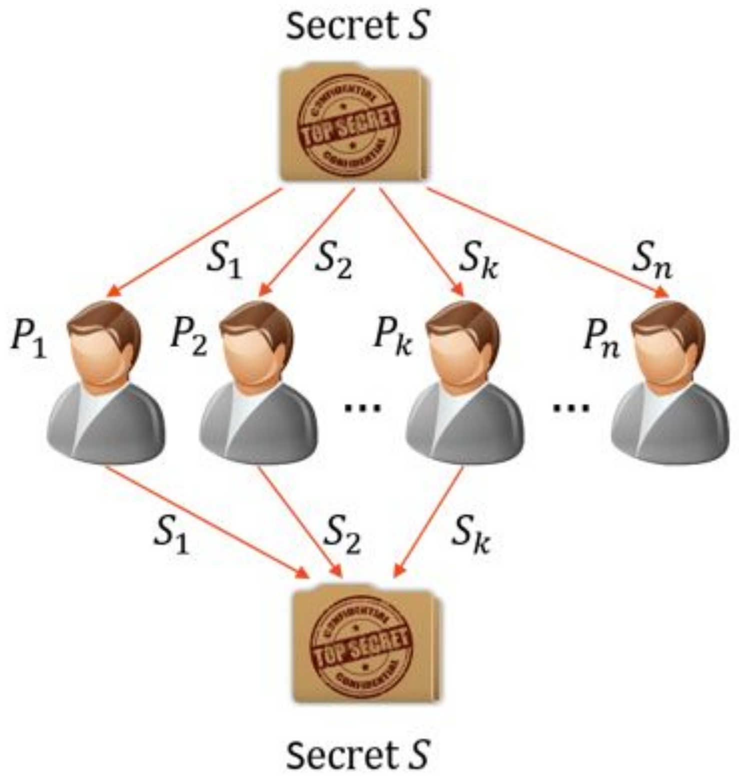
Introduction

First formulated in 1979 by Adi Shamir:

Shamir, A. (1979). How to share a secret. Communications of the ACM, 22(11), 612-613

SSS is used to secure a secret in a distributed form, most often to secure encryption keys. The secret is split into multiple shares, which individually do not give any information about the secret

To reconstruct a secret secured by SSS, a number of shares is needed, called the *threshold*. No information about the secret can be gained from any number of shares below than the threshold (perfect secrecy)



Tso, Raylin & Liu, Zi-Yuan & Hsiao, Jen-Ho. (2019). Distributed E-Voting and E-Bidding Systems Based on Smart Contract. Electronics. 8. 422. 10.3390/electronics8040422.

Properties

Secure: Information-theoretic security

Minimal: The size of each share does not exceed the size of the original data

Extensible: Shares can be dynamically added or deleted without affecting existing shares for a given threshold

Dynamic: New shares can be generated and distributed without changing the secret, but by changing the polynomial occasionally (keeping the same constant term)

Flexible: Keys can be distributed in any combination of shares based on the requirement. For instance, with a threshold of 3, the president could unlock the safe alone if given three shares, while three secretaries with one share each must combine their shares to unlock the safe

Algorithm

k points on the polynomial uniquely determines a polynomial of degree less than or equal to k-1.

The aim is to divide a secret S into n pieces of data $S_1,\ldots,S_n$ (known as shares) in such a way that:

- The entire secret S can be reconstructed from any combination of k or more shares.
- Knowledge of any k-1 or fewer shares S_i leaves S completely undetermined. (no partial information)

The process (1)

Let s be the secret, s=420. Let split it into n=8 shares, where a fraction of the shares, k=4, is sufficient to retrieve the secret.

We take k-1=3 numbers at random, let’s say 500, 4, and 45, and we construct the polynomial:

$$f(x) = 420 + 500x + 4x^2 + 45x^3$$

We then construct the 8 keys as (1, f(1)), (2, f(2)), etc.

The process (2)

Then if we take any subset of at least k=4 shares we can reconstruct the polynomial using Lagrange’s theorem which states that any polynomial of degree d is uniquely defined by d+1 points with the following formula:

$$L(X) = \sum_{j=0}^{n} y_j \left(\prod_{i=0, i \neq j}^{n} \frac{X - x_i}{x_j - x_i} \right)$$

So we can reconstruct the polynomial using the subset.

In practice this is not efficient because we are computing all coefficients here when only the first one is interesting (so another similar formula is used).

The process (3)

All these operations are supposed to be done modulo p, where $p > \max(s,n)$

This does not change anything to the procedure but allows for more security.

Indeed, over $\mathbb{N}$, if you have a fraction of the shares (less than k), then you obtain a system which only have a limited number of solutions in $\mathbb{N}$ (contradict perfect secrecy).

Over a finite field, you get no information from the system (because mod 3 for example, 1=4=7 etc.)

Drawbacks

No verifiable secret sharing: no way to know to that the shares provided are true (and if they are not, you will not find the secret)

Single point of failure: to encrypt the secret you need at some point to have all information centralized (when building the polynomial and computing the shares)

Lack of standardization: there is no real unified software so coding/decoding might not always be using the same procedure

Applications

SSS can be used for various purposes, some examples:

Vault (by Hashicorp) is a general encryption solution for companies that distributes the encryption key amongst multiple computers of the company for increased security using SSS

SLIP-0039 (by Satoshi Labs): preserving a cryptocurrency wallet as an alternative to redundancy, instead of making backups just distribute shares. This protects against the owner of the backup stealing the assets.

Possible improvements (1)

It is possible to improve SSS in many ways, one way is to add verifiable secret sharing using **Feldman’s scheme**.

Assume that SSS has been done with shares distributed (k,f(k)), then we choose a cyclic field and a generator g and we compute the **commitments** c:

c_0 = g^a0, c_1=g^a1, etc.

then anyone can verify their share by computing g^f(k):

$$g^{f(k)} = g^{\sum a_i k^i} = \prod g^{a_i k^i} = \prod c^{k^i}$$

which can be computed using the c

Possible improvements (2)

(the previous algorithm was simplified)

Another possible improvement is **Benaloh's scheme**

It guarantees that each holder is able to verify that all shares are collectively t-consistent (any subset t of n shares will yield the same, correct, polynomial without exposing the secret)

References

https://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf

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

https://developer.hashicorp.com/vault/docs/concepts/seal

https://github.com/satoshilabs/slips/blob/master/slip-0039.md

https://en.wikipedia.org/wiki/Verifiable_secret_sharing