Secret Sharing

> CSYE7374 Cryptography project

Background

Secret sharing is a technique to distribute a secret among a group. Each participant will get a "share", but no individual has any information about the secret. When sufficient number of "shares" are combined, we can recover the original secret. Secret sharing provides a secure way to let a group of people hold a secret, which can help avoid the lost of secret and spread the risk of being hacked.

One of the most wildly used secret sharing scheme is "threshold secret sharing scheme". It has two arguments - n and t. n is the number of participants and t is the number of participants required to reconstruct the secret. This scheme can be also called (n, t)-threshold scheme.

This project is to understand and implement different secret sharing schemes, discover implementation details and find out each implementation's advantages and disadvantages.

Schemes

There are three secret sharing schemes that are most widely used, they are all based on threshold scheme: Shamir's scheme, Blakley's scheme and scheme using Chinese remainder theorem. In this report, we denote the secret to be encrypted as S, and the number of participants as n, the threshold as t. The secret S here is just an integer number, in the following "application" section we will discuss how to convert it into encryption key which can be used by other encryption algorithms like AES.

Shamir's scheme

The core idea of Shamir's scheme is to construct a polynomial curve and give out each participant a "share" which is a point on the curve. We can then reconstruct this curve later using these points. The curve has to cross the point $(0, S)$ so that when we can reconstruct this curve we are able to get back the secret S.

To ensure it requires at least t shares to recover the secret, we will make this curve's degree to be $(t-1)$. We need at least t points to determine a curve with $(t-1)$ degree, so we have a threshold mechanism here.

Steps to encrypt:

- Randomly pick a secret S
- Construct a random curve with degree $(t-1)$ which closes $(0, S)$
  - A common used construction: $f(x) = S + a_1x + a_2x^2 + \ldots + a_{t-1}x^{t-1}$ where $a_1 \ldots a_{t-1}$ are random natural numbers

- Pick n points and give their coordinates to participants
  - we can let x to be $1, 2, \ldots, n$ and calculate $f(x)$ to get a point on the curve

Steps to decrypt:

- Get at least t points' coordinate
- Create a system of equations using these points
- Solve it and reconstruct the curve's expression
- Get S by substitute $x = 0$

Blakley's scheme

Blakley's scheme is somewhat similar to Shamir's scheme. Instead of picking n points on a curve, Blakley's scheme picks n non-parallel $(t-1)$-dimensional hyper-planes in t-dimensional space, so they will intersect at a specific point. Like Shamir's scheme, we can make this specific point to contain our secret S (e.g. make the first coordinate to be S).

Each share will be the coefficients of one hyper-plane. When we have t hyper-planes, we can calculate the intersection point and get the secret S back.

Steps to encrypt:

- Randomly pick a secret S
- Randomly pick a point p in t-dimensional space, its coordinate should be $(S, c1, c2, \ldots, c_{t-1})$
- Construct n non-parallel hyper-planes, they should all cross point p. To generate such hyper-plane:
  - The hyper-plane's equation should be: $A_1x_1 + A_2x_2 + \ldots + A_tx_t = c$
  - We can randomly generate $A_1, A_2, \ldots, A_t$, then substitute $x_1, x_2, \ldots, x_t$ using p's coordinate to get c
  - Ensure the new generated hyper-plane is not parallel to any hyper-plane generated before
- Give each participant a hyper-plane's coefficients $A_1, A_2, \ldots, A_t$ and c as a share

Steps to decrypt:

- Get at least t hyper-planes' coefficients
- We can construct a matrix equation $Ax = C$, where A is the coefficients of hyper-planes, x is the coordinate of their intersection point p, and C is the result that we got when generating the hyper-plane by substituting p's coordinates for $x_1, x_2, \ldots, x_t$.
- By solving this matrix equation we can get p's coordinate x, and then we get the secret S

Scheme using Chinese remainder theorem

This scheme is different to the other two schemes and it utilizes modular arithmetic and Chinese remainder theorem.

Chinese remainder theorem (CRT)

Say we have a system of congruences:

$$\begin{cases} x & \equiv & b_1 & mod & m_1 \\ & & \vdots & & \\ x & \equiv & b_k & mod & m_k \end{cases}$$

If m_i and m_j are relatively prime for all (i, j), then Chinese remainder theorem is able to calculate a unique solution in range $(0, M)$ where $M = LCM(m_1, \ldots, m_k)$. Here $LCM()$ is least common multiple.

Secret sharing scheme

We can make x to be our secret, and the modulo m and corresponding remainder b as our shares. So we will need n number of modulus $m_1, m_2, \ldots, m_n$. To ensure they are co-prime, we can just use primes as our modulus.

Another property is also important. Because we want to get a unique result using any t shares, and CRT says the solution will only be unique in range $(0, M)$ where $M = LCM(m_1, \ldots, m_k)$, we need to make sure that the secret x or S is smaller than any t modulus' LCM (equivalent to smaller than the smallest t modulus' LCM)

Similarly, we don't want it to be able to recover the secret using any $(t - 1)$ shares. So we also need to ensure the secret x or S is greater than any $t - 1$ modulus' LCM (equivalent to greater than the largest $t - 1$ modulus' LCM)

Use math expression, if:

$$m_1 < m_2 < \ldots < m_n$$

We will need to ensure:

$$LCM(m_{n-t-2}, \ldots, m_n) < S < LCM(m_1, \ldots, m_t)$$

Steps to encrypt

- Random generate n unique primes $m_1, m_2, \ldots, m_n$, where $m_1 < m_2 < \ldots < m_n$
  - Because of the requirements we discussed above, it's very difficult to first generate a random secret then find valid primes. So we first generate primes then pick a random secret later
- Calculate the range $(LCM(m_{n-t-2}, \ldots, m_n), LCM(m_1, \ldots, m_t))$, if it's not valid, re-generate another list of primes
- Randomly pick a number from range above as our secret S
- Perform modulo operation and get all remainders $b_1, b_2, \ldots, b_n$
- Distribute modulus and its corresponding remainders to participants

Steps to decrypt

- Get t modulus and remainders
- Construct a system of congruences
- Solve this system using CRT and we get the secret S back

Implementation

Shamir's scheme

In practice, the method described above might be hacked. Here is an attack example from wikipedia:

> Let $n = 6$, $t = 3$, $f(x) = S + a_1 x + a_2 x^2$
>
> Suppose an attacker is able to find two points on the curve $p_1 = (1, 1494)$, $p_2 = (2, 1942)$, since it only has two shares, in theory he should not get any information about S
>
> But he could perform the following algebra:
>
> - Subsitute x using p_1
>
> $$f(x) : 1494 = S + a_1 * 1 + a_2 * 1^2 \Rightarrow 1494 = S + a_1 + a_2 \quad (1)$$
>
> - Subsitute x using p_2
>
> $$f(x) : 1942 = S + a_1 * 2 + a_2 * 2^2 \Rightarrow 1942 = S + 2a_1 + 4a_2 \quad (2)$$
>
> - Subtract $(2) - (1)$
>
> $$(1942 - 1494) = (S - S) + (2a_1 - a_1) + (4a_2 - a_2) \Rightarrow 448 = a_1 + 3a_2 \quad (3)$$
>
> - Because we know a_1 and a_2 are natural numbers, we can get
>
> $$0 \leq 3a_2 \leq 448 \Rightarrow a_2 \in [0, 1, \ldots, 149] \quad (4)$$
>
> - Replacing a_1 by $448 - 3a_2$ in (2), and replacing a_2 by values in (4), we have:
>
> $$S \in [1046, 1048, \ldots, 1344] \quad (5)$$
>
> - So the attacker only need to take limited number of guesses and he can find the secret S

We will need to use **finite field arithmetic** to solve this problem. This means we will simply need to add a modulo operation to our function expression:

$$f(x) = (S + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}) \ mod \ p$$

Here p is public prime shared with all participants. In the above example, if we use finite field arithmetic, we can no longer get the range of a_2 in (4), because $3a_2$ can be any large and after modulo operation it may still get 448 as result. So this avoids possible attack.

code implementation

We will need to set a public prime p first before generating shares. The prime should be large enough as it's the strength of the system ($S \ mod \ p$ has p possible results). p also need to be larger than every a_i. In my code I used $2^{127} - 1$ as the default prime.

Then we need to generate the coefficients $a_1, a_2, \ldots, a_{t-1}$ for the function. Ideally their range should be in $(0, p)$, but because there is no handy library or functions that can be used to generate uniform random BigInt numbers, I used BigInt's `probablePrime()` for simplicity. (This reduces the security of the system!) The bits number used for `probablePrime()` is 50. The secret S is generated using this function and bits as well.

Finally, when reconstructing the secret, we will need to solve the system of equation. If we are not using finite field arithmetic, we can use matrix equation to get the solution. But with finite field arithmetic, we will have to use Lagrange interpolation to reconstruct the curve.

Blakley's scheme

We first generate the point p's coordinates. I used the same `probablePrime()` function to generate random number. The bits number set for p's coordinate is 20.

Then we generate coefficients for each hyper-plane. In total we need to get $n * t$ coefficients. The bits number set for coefficient is 50.

After we have all coefficients we can calculate the c values, construct shares and distribute them to each participant.

For decrypting/reconstruction, we solve the matrix equation $Ax = C$ as discussed in previous section. A big problem here is the library(Breeze) I used for matrix operation doesn't support `BigInt` matrix solution. So we have to convert them to `Double` then solve the equation, and in the end, we need to convert the solution from `Double` back to `BigInt`. We have some precision lost during conversion and rounding, which may result in wrong secret value when reconstructing.

Another problem is this implementation which only uses integer arithmetic might also be hacked by similar attack to Shamir's scheme demostrated above. For instance, an attacker got $(t - 1)$ shares (hyper-planes). These planes will intersect at a specific line. The attacker can then check for points whose coordinates are all natural numbers and limit the possible values for secret S. We can also use finite field arithmetic to avoid this problem. (Not implemented in the code as Shamir's scheme is overall better)

Scheme using Chinese remainder theorem

We need n unique primes, so it's ideal to use `probablePrime()` function. Bits number is set to 50 by default.

Then we sort these primes and calculate the range $(LCM(m_{n-t-2}, \ldots, m_n), LCM(m_1, \ldots, m_t))$. If the range is not valid, we need to re-generate all primes until it meets the requirement.

For decrypting, we can use this algorithm to solve the system of congruences.

Benchmarking

The operations that takes computing resources for each schemes:

	Shamir's scheme	**Blakley's scheme**	**Scheme using CRT**
encrypt	- Generate secret S - Generate $t-1$ unique coefficients - Calculate n function outputs	- Generate secret S - Generate p's coordinates - Generate n hyper-planes, each of them has t coefficients - Calculate value c for each hyper-plane	- Generate n unique primes - Calculate valid range - Randomly generate a secret S in the range - Calculate the remainder for each modulus
decrypt	- Lagrange interpolation of system of equation	- Construct matrices and solve the equation	- Solve the system of congruences

Blakley's scheme is very similar to Shamir's scheme, but it has more coefficients and uses more spaces.

Here are the results of benchmarking. We set n as 5 and t as 3. Ran 100000 times.

average time(ns)	**Shamir's scheme**	**Blakley's scheme**	**Scheme using CRT**
encrypt	212,968	1,189,995	542,123
Decrypt	5,965	4,969	11,180

We can see that Shamir's scheme is much faster than the other two schemes when encrypting, and its decrypt speed is only 1ms slower than Blakley's scheme. Also Shamir's scheme uses much less spaces than Blakley's scheme. Both Shamir's scheme and CRT scheme has two BigInt number per share.

One limitation of CRT scheme is it cannot use customize secret (which is actually supported by Shamir's scheme and Blakley's scheme). Because we cannot easily find proper primes that meets CRT's requirement, we have to generate primes first then pick the secret S.

Overall, Shamir's scheme seems to be the secret sharing scheme that has best performance. And indeed it is the most widely used scheme in practice.

Application

Now, the secret S we discussed is only an integer. We can of course use these secret sharing schemes to encrypt a number. But if we want to encrypt something more complicated (like a database), we will have to convert the secret S to something that we can use in common encryption methods.

PBE (password based encryption) is one method to convert any input to an encryption key. It has its own hashing function which can take any input and generate an encryption key.

Here are steps to use secret sharing schemes to actually encrypt something (e.g. database)

encrypt

- Use secret sharing schemes to generate random secret S and all the shares
- Convert S to a encryption key K using PBE
- Use K to as encryption key to encrypt data

decrypt

- Reconstruct the secret S using at least t shares
- Convert S again to get the key K using PBE
- Use K to decrypt ciphertext

Summary

In this project, we learned and implemented three popular secret sharing schemes. We also tested the performance of each scheme, and we found out that Shamir's scheme overall has the best performance.

One interesting thought while working on this project is that, all these three schemes assumes that all participants have the same weights or rights, so they all have same kinds of shares. What if we want a more complicated system that we could customize different participants' weights and give them different shares? It could be something worthy and interesting to research on.