Shamir's Secret Sharing

Mohammed Mostafa
362118614

April 16, 2020

Contents

Overview **3**
Algorithm 3
Reconstruction 4

Implementation **4**
Code 5

Overview

Secret Sharing refers to methods for distributing a secret amongst a group of participants, each of whom is allocated a share of the secret. The secret can be reconstructed only when a sufficient number of shares are combined together.

There can be different thresholds for how many shares are needed to reconstruct the secret:

$\mathbf{t = 1}$ Secret sharing is trivial. The share is the secret.

$\mathbf{t = n}$ All shares are necessary to recover the secret.

$\mathbf{1 < t < n}$ The shares needed are greater than 1 and less than the total number of shares created. This is called a shreshold secret sharing scheme.

Shamir's Secret Sharing is a type of threshold secret sharing algorithm which will be implemented and analyzed on files in this paper.

Algorithm

Suppose that we have the secret is 1234 ($S = 1234$).

We wish to divide the secret into 6 parts ($n = 6$), where any 3 parts ($k = 3$) is sufficient to reconstruct the secret.

At random we obtain $k - 1$ numbers: 166 and 94.

$(a_0 = 1234; a_1 = 166; a_2 = 94)$, where a_0 is the secret.

Our polynomial to produce secret shares (points) is therefore:

$f(x) = 1234 + 166x + 94x^2$

We construct six points $D_{x-1} = (x, f(x))$ from the polynomial for each share:

$D_0 = (1, 1494); D_1 = (2, 1942); D_2 = (3, 2578)$

$D_3 = (4, 3402); D_4 = (5, 4414); D_5 = (6, 4514)$

We give each person a different point. The points start from $(1, f(1))$ because $f(0)$ is the secret.

Reconstruction

In order to reconstruct the secret any 3 points will be enough. By using interpolation, we can produce the secret a_0 from the points D_1, D_3, D_4.

Using Lagrange basis polynomials:

$$\ell_0 = \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_2} = \frac{x-4}{2-4} \cdot \frac{x-5}{2-5} = \frac{1}{6}x^2 - \frac{3}{2}x + \frac{10}{3}$$

$$\ell_1 = \frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} = \frac{x-2}{4-2} \cdot \frac{x-5}{4-5} = -\frac{1}{2}x^2 + \frac{7}{2}x - 5$$

$$\ell_2 = \frac{x - x_0}{x_2 - x_0} \cdot \frac{x - x_1}{x_2 - x_1} = \frac{x-2}{5-2} \cdot \frac{x-4}{5-4} = \frac{1}{3}x^2 - 2x + \frac{8}{3}$$

Therfore:

$$\begin{aligned} f(x) &= \sum_{j=0}^{2} y_j \cdot \ell_j(x) \\ &= y_0\ell_0 + y_1\ell_1 + y_2\ell \\ &= 1942\left(\frac{1}{6}x^2 - \frac{3}{2}x + \frac{10}{3}\right) + 3402\left(-\frac{1}{2}x^2 + \frac{7}{2}x - 5\right) + 4414\left(\frac{1}{3}x^2 - 2x + \frac{8}{3}\right) \\ &= 1234 + 166x + 94x^2 \end{aligned}$$

Since we know the secret is the free coeffecient, this means $S = 1234$.

Implementation

This is a python implementation of Shamir's Secret Sharing algorithm where the secret is a file. We use two functions, one for splitting "enc" the secret into shares and another for merging them "dec".

Since Shamir's algorithm uses finite field arithmetic for security, the size of the file we can use as the secret is limited by the prime number.
For this implementation the maximum prime number we have is $2^{20996011} - 1$, this allows us to use files as large as 20996011 bits or 2.5 MB.

Code

```
from __future__ import division
from __future__ import print_function
from tkinter.filedialog import askopenfilename
import random, functools, os, re

# Mersenne Prime
p = [2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281,
3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497, 86243, 110503,
132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593,
13466917, 20996011 ]

# Mersenne Prime
_PRIME = 0
_RINT = functools.partial(random.SystemRandom().randint, 0)


def _eval_at(poly, x, prime):
    accum = 0
    for coeff in reversed(poly):
        accum *= x
        accum += coeff
        accum %= prime
    return accum


def make_random_shares(minimum, shares, secret):
    global _PRIME
    prime = _PRIME
    if minimum > shares:
        raise ValueError("Pool secret would be irrecoverable.")
    poly = [_RINT(prime) for i in range(minimum - 1)]
    poly.insert(0, secret)
    points = [(i, _eval_at(poly, i, prime)) for i in range(1, shares + 1)]
    return points


def _extended_gcd(a, b):
    x = 0
    last_x = 1
    y = 1
    last_y = 0
    while b != 0:
        quot = a // b
```

```
        a, b = b, a % b
        x, last_x = last_x - quot * x, x
        y, last_y = last_y - quot * y, y
    return last_x, last_y


def _divmod(num, den, p):
    inv, _ = _extended_gcd(den, p)
    return num * inv


def _lagrange_interpolate(x, x_s, y_s, p):
    k = len(x_s)
    assert k == len(set(x_s)), "points must be distinct"

    def PI(vals):  # upper-case PI -- product of inputs
        accum = 1
        for v in vals:
            accum *= v
        return accum

    nums = []  # avoid inexact division
    dens = []
    for i in range(k):
        others = list(x_s)
        cur = others.pop(i)
        nums.append(PI(x - o for o in others))
        dens.append(PI(cur - o for o in others))
    den = PI(dens)
    num = sum([_divmod(nums[i] * den * y_s[i] % p, dens[i], p) for i in range(k)])
    return (_divmod(num, den, p) + p) % p


def recover_secret(shares):
    global _PRIME
    prime = _PRIME
    if len(shares) < 2:
        raise ValueError("need at least two shares")
    x_s, y_s = zip(*shares)
    return _lagrange_interpolate(0, x_s, y_s, prime)


def enc():
    global _PRIME
    filename = askopenfilename()
```

```
    fi = open(filename, "rb")
    data = fi.read()
    secret = int.from_bytes(data, "big")
    file_size = os.stat(filename).st_size
    if file_size * 8 > p[-1]:
        raise ValueError("file too big")
    if file_size * 8 in p:
        pass
    else:
        for i in p:
            if i > file_size * 8:
                file_size = i
                break
    _PRIME = 2 ** file_size - 1

    _shares = 25
    _thresh_hold = 9
    shares = make_random_shares(_thresh_hold, _shares, secret)
    try:
        path = os.path.abspath(os.getcwd())
        os.mkdir(path + "/shares")
    except OSError:
        print("Creation of the directory %s failed" % path)
    else:
        print("Successfully created the directory %s " % path)

    for i in range(len(shares)):
        fo = open(f"shares/share_{i+1}", "wb")
        fo.write(int.to_bytes(shares[i][1], file_size, "big"))

    f2 = open(f"shares/size", "w")
    f2.write(f"{file_size}\n{os.path.basename(filename)}")


def dec():
    _thresh_hold = 9
    filecontent = []
    file = open("shares/size", "r")
    for line in file:
        filecontent.append(line)

    print(filecontent[1])
    file_size = int(filecontent[0])

    filename = filecontent[1]
```

```
    n_shares = _thresh_hold  # Number of files to be read
    sharese = []

    for i in range(n_shares):
        with open(f"shares/share_{i+1}", "rb") as file:
            file_data = file.read()
            sharese.append((i + 1, int.from_bytes(file_data, "big")))

    print("Recovering secret...")
    secret = recover_secret(sharese[:_thresh_hold])
    with open(os.path.basename(filename), "wb") as file:
        file.write(int.to_bytes(secret, file_size, "big").lstrip(b"\x00"))
    print("DONE!!")


def main():
    """Main function"""
    enc()
    dec()


main()
```