A Project Report on

# CHEATER IDENTIFICATION IN SECRET SHARING SCHEME

Submitted to the Department of Information Technology

**For the partial fulfilment of the degree of B.Tech in Information Technology**

by

1. Sounit Ghosh (510816004)
2. Suman Mahato (510816044)
3. Sarvajeet Haldar (510816031)

B.Tech, 4th year

Under the supervision of

Prof. Shyamalendu Kandar

Department of Information Technology
INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR

*JUNE, 2020*

**Department of Information Technology**
**Indian Institute of Engineering Science and Technology, Shibpur**

# CERTIFICATE

This is to certify that the work presented in this report entitled **"Cheater Identification in Secret Sharing Scheme"**, submitted by **Sounit Ghosh, Suman Mahato, Sarvajeet Haldar** having the examination roll number **510816004, 510816044, 510816031** has been carried out under my supervision for the partial fulfilment of the degree of Bachelor of Technology in Information Technology during the session 2018-19 in the Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur.

_____

Shyamalendu Kandar
Assistant/Associate Professor
Department of Information Technology
Indian Institute of Engineering Science and Technology, Shibpur

_____

Hafijur Rahaman
Head of the Department
Department of Information Technology
Indian Institute of Engineering Science and Technology, Shibpur

_____

Dean (Academic)
Indian Institute of Engineering Science and Technology, Shibpur

Date: 27.06.2020

# Acknowledgements

We would like to express our gratitude to our supervisor **Prof. Shyamalendu Kandar**, Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur. For his inspiration, guidance and motivation throughout the project conducted at Indian Institute of Engineering Science and Technology, Shibpur. This project would never been possible without his immense guidance and insight. He was always so helpful and co-operative. He was always there to clear our doubt and discuss our work.

Date: 27.06.2020

Sounit Ghosh, Suman Mahato, Sarvajeet Haldar
Department of Information Technology
Indian Institute of Engineering Science
and Technology, Shibpur

**Abstract**

In a ($t$, $n$) secret sharing scheme, a secret $s$ is divided into $n$ shares and shared among a set of $n$ shareholders by a mutually trusted dealer in such a way that any $t$ or more than $t$ shares will be able to reconstruct this secret;
Fewer than $t$ shares cannot know any information about the secret. When shareholders present their shares in the secret reconstruction phase, dishonest shareholder(s) (i.e. cheater(s)) can always exclusively derive the secret by presenting faked share(s) and thus the other honest shareholders get nothing but a faked secret.
Cheater detection and identification are very important to achieve fair reconstruction of a secret. In this paper, we consider the situation that there are more than $t$ shareholders participated in the secret reconstruction. Since there are more than $t$ shares (i.e. it only requires $t$ shares) for reconstructing the secret, the redundant shares can be used for cheater detection and identification. The scheme we have worked on uses the shares generated by the dealer to reconstruct the secret and, at the same time, to detect and identify cheaters.

# Contents

# 1. INTRODUCTION

Secret sharing schemes were originally introduced by both Blakley and Shamir independently in 1979 as a solution for safeguarding cryptographic keys and have been studied extensively in the literatures.

In a secret sharing scheme, a secrets is divided into n shares and shared among a set of n shareholders by a mutually trusted dealer in such a way that any t or more than t shares will be able to reconstruct this secret; but fewer than t shares cannot know any information about S. Such a scheme is called a (t, n) secret sharing, denoted as (t, n)-SS. Shamir's (t, n)-SS scheme is very simple and efficient to share a secret among n shareholders.

However, when the shareholders present their shares in the secret reconstruction phase, dishonest shareholder(s) (i.e. cheater(s)) can always exclusively derive the secret by presenting faked share(s) and thus the other honest shareholders get nothing but a faked secret.

It is easy to see that the Shamir's original scheme does not prevent any malicious behavior of dishonest shareholders during secret reconstruction. Cheater detection and identification are very important to achieve fair reconstruction of a secret. There are many research papers to investigate the problem of cheater detection and/or identification for secret sharing schemes. Some of them consider that there are exactly t shareholders participated in the secret reconstruction. In order to enable each shareholder the ability of cheater detection and identification, the dealer needs to generate and distribute additional information, such as using check vectors and certificate vectors for each shareholder. Some other papers proposed to design a secret sharing scheme based on an error-correcting code in which faked shares can be treated as error codes to be detected and corrected based on coding technique.

# 2. PRELIMINARIES AND DEFINITIONS

This section provides some basic mathematical and cryptographic concepts, which are major tools in secret sharing schemes, and reviews related works.
First of all, the definition of finite field is provided together with some properties. Polynomials in a finite field and Lagrange interpolation are also discussed to give understanding on the concepts. After that, we review related works such as linear secret sharing schemes, access structure, and some previous schemes like Shamir's and fiedman's.

## 2.1 Definitions:

2.1.1 Secret:

- A highly sensitive data meant to be kept unknown

  E.g. encryption keys, missile launch codes, numbered bank accounts etc.

2.1.2 Secret sharing:

- Breaking a secret into multiple shares
- Distributing the shares among multiple parties
- A subclass of these parties can reconstruct the secret
- Thus there is no single point of failure that can lead to its loss

2.1.3 Cheater:

- A person who acts dishonestly in order to gain an advantage and the way he/she does this is called cheating.

## 2.2. Basic Mathematical and Cryptographic Concepts:

This is an overview of some mathematical concepts, which are useful in secret sharing like finite field, polynomial and Lagrange interpolation.

### 2.2.1. Finite Field:

**Definition 1.** finite field $\mathbb{F}$ is a finite set on which addition, subtraction, multiplication, and division are defined and the following axioms are satisfied.

*(1)Associative*: for all a, b, and in $\mathbb{F}$, a + (b + c) = (a + b) + c and a · (b · c) = (a · b) · c.

*(2)Commutative*: for all and b∈ $\mathbb{F}$, a + b = b + a and a · b = b · a.

*(3)*Existence *of identity*: there exists elements e and e' ∈ $\mathbb{F}$, such that a + e = e + a = a and a·e' = e' · a = a.
(4)Existence of inverse: for every element a ∈ $\mathbb{F}$, there exists an element -a such that a + (-a) = e. Similarly for every element a ∈, there exists an element $a^{-1}$∈ $\mathbb{F}$ such that a · $a^{-1}$ =e' .
*(5)Distributive*: for all a, b, and in $\mathbb{F}$, a · (b + c) = (a · b) + (a · c).
Note that an element -a is called additive inverse and another element is called multiplicative inverse. An element is an additive identity and an element is multiplicative identity.

## 2.2.2. Polynomials over Finite Field:

Definition 1.(polynomial over $F_p$ ). Let $F_p$ be a field. Any expression
f (x)=$\sum_{i=0}^{t} aixi$ a-i ∈ Fp

where t is an arbitrary positive integer which is called a polynomial over $F_p$ .

Definition 2. (degree of polynomial f(x)). Given a nonzero polynomial
f (x)=$\sum_{i=0}^{t} aixi$ , where $a_i \neq 0$, the number is said to be the degree of f(x) denoted as deg f(x).

Definition 3. (equal polynomials). Let f(x)=$\sum_{i=0}^{t} aixi$ and
f(x)=$\sum_{i=0}^{m} bixi$, where $a_t \neq 0$ and $b_m \neq 0$ are two polynomials of degrees t and m, respectively. The two polynomials are equal and write f(x)=f' (x), if t = m and ai =bi for all i = {0, 1, 2, …,t} .

Definition 4 (roots of a polynomial). An element α∈F is called a root of f(x) if f(α) = 0.
A polynomial f (x)=$\sum_{i=0}^{t} aixi$ , ai∈F of degree t cannot have more than t roots in the field F .

## 2.2.3. Lagrange Interpolation:

This is a method of reconstructing a polynomial from given known points. The polynomial constructed by Lagrange interpolation is called Lagrange interpolation polynomial, which is unique. To reconstruct the polynomial of degree t, t + 1 values are required, i.e., (αi,βi ) for all i = 0, 1, 2, …, t such that f(αi) =βi .

Proposition 10.

Let for all i = 0, 1, 2, …, t be distinct elements of F and βi for all i = 0, 1, 2, …, t be arbitrary elements of F. There exists no more than one polynomial f(x) of degree at most t such that f(αi) =βi for all i = 0, 1, 2, …, t.

Theorem 11.

Let α0, α1 , …αt, be distinct elements of F and β0, β1..,βt, be arbitrary elements of F . There exists a unique polynomial

$$f(x)=\frac{(x-\alpha 0)\ldots(x-\alpha i-1)(x-\alpha i+1)\ldots(x-\alpha t)}{(\alpha i-\alpha 0)\ldots(\alpha i-\alpha i-1)(\alpha i-\alpha i+1)\ldots(\alpha i-\alpha t)}$$

of degree at most t such that f(αi) = βi for all i = 0, 1, 2, …, t.

Proof. We adopt the proof by Slinko. The polynomial was constructed as follows. First construct polynomials gi(x) of degree t such that gi(αi) = 1 and gi(αj) = 0 for i ≠ j. These polynomials are

$$f(x)=\frac{(x-\alpha 0)\ldots(x-\alpha i-1)(x-\alpha i+1)\ldots(x-\alpha t)}{(\alpha i-\alpha 0)\ldots(\alpha i-\alpha i-1)(\alpha i-\alpha i+1)\ldots(\alpha i-\alpha t)}$$

Thus, the polynomials g(x), g1(x), …, gt(x) are constructed. Furthermore, we multiply by for i = 0, 1, 2, …, t and obtain the polynomials $\beta_0 g_0(x)$, $\beta_1 g_1(x)$, …, $\beta_t g_t(x)$. Summing the polynomials $\beta_i g_i(x)$ the desired polynomial f(x) is constructed as Equation

$$f(x)=\sum_{i=0}^{t} \beta i g i(x)$$

We set f(αi) =βi as required. This polynomial is unique by Proposition 10.

# 3 Related Works :

This section reviews linear secret sharing schemes, access structure of secret sharing schemes, and some previous schemes like Shamir's scheme .

### 3.1.1. Linear Secret Sharing Scheme:

Linear (t, n) secret sharing scheme is a special type of secret sharing scheme where all the shares of the secret satisfy a linear relationship. The Definition 13 gives what linear secret sharing scheme is.

Definition 13 (linear secret sharing scheme). A (t, n) secret sharing scheme is a linear secret sharing scheme when the n shares v1,v2 ,v3 , …vn, can be presented as in Equation

v1,v2,v3…vn=(k1,k2,…,kt)H,

where H is a public t × n matrix whose any t × t submatrix is not singular. The vector (k1,k2 ,k3 …,kt ) is randomly chosen by the dealer.

According to Definition 13, we can see that Shamir's (t, n) secret sharing scheme is a linear scheme. Let
f (x)=a0+a1x+a2x2+…+at-1xt-1

The shares vi = f(i), i =1,2,…, n can be presented as in Equation

(v1,v2,…,vn)=(a0,a1,a2…at-1)H,

where hi,j = $j^{i1}$ (hi,j denotes the entry at ith row and jth column of matrix H).

### 3.1.2. Shamir's (t, n) Threshold Scheme:

Shamir proposed a (t, n) threshold scheme that splits a secret s∈ S into shares, which are distributed to users . Splitting is done by a dealer using an algorithm called share generation algorithm. The algorithm uses a polynomial f(x) of degree t - 1 to generate and distribute shares. The secret is reconstructed based on interpolating a polynomial using Lagrange interpolation, which is reconstructed by users. The users combine their shares to reconstruct a polynomial (x) of degree using reconstruction algorithm. The algorithm inputs the user's identity and their share , which forms a point or an ordered pair (i, ) for all i = 1, 2, …, t and outputs the secret f(0) = s. Shamir's scheme has the following important properties.

(i)Share size is exactly equal to secret size.
(ii)If a new player joins or leaves, it is easy to add or delete shares without affecting the other shares.(iii)It is easy to change the shares of the same secret just by changing the polynomial without breaching any security.
(iv)t - 1 users do not reveal any information about the secret.
However, Tompa and Woll discovered that the scheme cannot withstand cheating if there is an untrusted user during secret reconstruction . As a result, Shamir's scheme faces the following challenges during secret reconstruction.
(i)Any malicious user can present a forged share without being noticed.
(ii)It is difficult to detect if the reconstructed secret is invalid.
(iii)A malicious user, once is successful in cheating other users, will be able to reconstruct the valid secret.

### 3.1.3. Cheating Prevention:

Cheating prevention in secret sharing became a great concern after Tompa and Woll introduced cheating concept. As a result, many schemes with cheating prevention are proposed where some detect cheating, others identify cheaters, and so on. Some of the categories of cheating prevention are as follows.

(i)Cheating detection: schemes provide the method to detect any forged share submitted for secret reconstruction by malicious user . The assumption is that the dealer is trusted.
(ii)Cheater identification: schemes provide the method to detect and identify any forged share presented for secret reconstruction by a malicious user. The assumption is that the dealer is trusted.
(iii)Robust secret sharing: schemes assume the dealer is trusted. Schemes can reconstruct a correct secret even if there are a number of forged shares presented by untrusted user .
(iv)Verifiable secret sharing: schemes assume that the dealer is not trusted. Each user verifies the shares if valid using verification algorithm before reconstruction is done .

# 4. PROBLEM DEFINITION

---

Cheater Identification in a Secret Sharing Scheme by Software realization of Shamir and Feldman's SSS.

---

# 5. Shamir's Secret Sharing:

- An old cryptography algorithm invented by the Israeli cryptographer Adi Shamir(1979).
- Split a secret S in n parts
- Any k-out-of-n pieces can reconstruct the original secret S
- But with any k-1 pieces no information is exposed about S
- This is conventionally called a (n, k) threshold scheme.

## 5.1. Creating the share:

- To split a secret S into n shares, such that

-any combination of <= L shares can't learn S
-any combination of >=L shares learns S

- We construct a degree-L polynomial f such that f(0) = S
- And compute,

$share_1 = f(1)$
share2 = f(2)
……..
……..
$share_n = f(n)$

## 5.2. Reconstructing the Secret:

- First we need to recreate the polynomial using the Lagrange Polynomial Interpolation(1795)

Ex:

- Let the shares be f(5)=3, f(7)=2, f(12)=6, f(30)=15
- Then $\partial_i(x) = \prod \frac{x-j}{i-j}$ for j ∈ C, j!=i where C ={5,7,12,30}

- Now we can reconstruct the polynomial by computing,
  $f(x) = 3\,\partial_5(x) + 2\partial_7(x) + 6\partial_{12}(x) + 15\partial_{30}(x)$
- And to get the secret we just need to compute f(0)

## 5.3. Software Realization of Shamir Secret Sharing Scheme:

```
# -*- coding: utf-8 -*-
"""
Created on Fri May  18 17:04:21 2020

@author: Sarva
"""
#from random import randint
#from lagrange import interpolate

from scipy.interpolate import lagrange
#from numpy.polynomial.polynomial import Polynomial

import random
import numpy as np

def generate_shares(n,k,p):

s = float(input("Enter the secret(s) to be divide into shares such that s<p :"))

#p=p_main
#k=k_main
#n=n_main

"""Generating k-1 random numbers from a finite field of size p  """
random_numbers = random.sample(range(1, p), k-1)#Using random.sample()
#print(random_numbers)


'''
We are generating a polynomial of coeffients(a_i) for the polynomial such
that a_i<p, p>s, p>n & a_0 = S
'''random_numbers.append(s)
```

```
poly = np.poly1d(random_numbers)
print("Evaluating the polynomial at 0:", poly(0))

'''
We construct the n shares that are distributed to the participants. Each
share is simply a point on the polynomial just defined.
Each point D can be calculated in an iterative manner:
Dx-1 =(x, f(x) mod p) where x=[1,2,....n]
(Here we generate a list consisting of Dx-1 for all x in [1,2,...n])
'''
D=[]
for i in range(1,n+1):
Dy = poly(i) % p
Dx = i
Dxy = (Dx,Dy)
D.append(Dxy)

return D


def reconstruct_secret(X,Y,p):

#In order to reconstruct the original secret from any k-out-of-n parts, we
#need to recreate the polynomial that we defined in the beginning. This can
#be achieved with the Lagrange Polynomial Interpolation.

#X=X_main
#Y=Y_main
#p=p_main

x = np.array(X)
y = np.array(Y)
poly = lagrange(x, y)

#list_of_coefficients = Polynomial(poly).coef
#poly = np.poly1d(list_of_coefficients)
#s = Polynomial(poly).polyval(0, list_of_coefficients)
s = poly(0) % p
return s


#def reconstruct_secret(shares,p):
# return interpolate(shares, p)


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

print("<<< Shamir's Secret Sharing Scheme >>>")
print("Enter 1 to generate shares from a secret")
print("Enter 2 to find the secret from keys")

inp = int(input("Enter your value: "))

if(inp == 1):
```

```
'''Select a very large prime number(p) as the finite fields to which
our algorthm will be restricted
 '''
p_main = 15485867
print("Default Prime no.(p) for restricting finite fields: ", p_main )

p_change = int(input("To change p enter a large prime else enter 0 :"))
if(p_change!=0):
p_main = p_change
print("Prime no.(p) selected for restricting finite fields: ", p_main )


#print("<<Enter integers only>>", end=" ")
n_main = int(input("Enter the no. of shares(n) to be generated such that n<p :"))

k_main = int(input("Enter the minimum no. of shares(k) required to reconstruct the secret:"))
print("")
print("<<< Publicly known values are >>>")
print("No. of shares :",n_main)
print("Threshold no. of shares :",k_main)
print("Prime p:", p_main)

D_main = generate_shares(n_main,k_main,p_main)
print("The list of shares are:", D_main)


elif(inp == 2):

'''Select a very large prime number(p) as the finite fields to which
our algorthm will be restricted
 '''

p_main = int(15485867)
print("Default Prime no.(p) for restricting finite fields: ", p_main )

p_change = int(input("To change p enter a large prime else enter 0 :"))
if(p_change!=0):
p_main = p_change
print("Prime no.(p) selected for restricting finite fields: ", p_main )

no_of_keys = int(input("Enter the no. of keys(max 20) :"))

#this is because pol1d can't compute shares>20
if(no_of_keys>20):
print("Enter shares<=20")

else:
X_main=[]
Y_main=[]
#shares_main=[]
for i in range(no_of_keys):
print("Key",i+1, end=">>")

X_main.append(float(input("Enter the x value of the key:")))
```

```
Y_main.append(float(input("Enter the y value of the key:")))
print("The secret is:", reconstruct_secret(X_main,Y_main, p_main))

'''
x = float(input("Enter the x value of the key:"))
y = float(input("Enter the y value of the key:"))
xy= (x,y)
shares_main.append(xy)

print("The secret is:", reconstruct_secret(shares_main, p_main))
'''
if __name__ == '__main__':
   main()
```

## Output:

<<< Shamir's Secret Sharing Scheme >>>

- Enter 1 to generate shares from a secret
- Enter 2 to find the secret from keys
- Enter your value: 2
- Default Prime no.(p) for restricting finite fields:  15485867
- To change p enter a large prime else enter 0 :0
- Enter the no. of keys(max 20) :3
- Key 1>>
- Enter the x value of the key:1
- Enter the y value of the key:7280850
- Key 2>>
- Enter the x value of the key:2
- Enter the y value of the key:6466022
- Key 3>>
- Enter the x value of the key:3
- Enter the y value of the key:13042407
-

The secret is: 1024.0

## 5.4. SHORTCOMINGS OF SHAMIR SECRET SHARING SCHEME –

- If a key is reconstructed from shares it is difficult to say that which shares are used for reconstruction.
- If a share belongs to some split secret, there is no way to verify that the data has not been corrupted. So during the reconstruction process wheather it succeeds or fails there is no way to verify the correctness of the retrived secret. It happens when someone submits fake secret.
- It is not specified by Shamir's that if the secret is to be updated then how to update the shares and is this process is totally non-vulnarable.
- Shamir did not mention wheather the shares should be divided and reconstructed using the same system and software.

## 5.5. HOW TO OVERCOME AS STATED BY TOMPA AND WOLL -

Tompa and Woll stated two advantages over Shamir's i.e

1.All currently knowned schemes(including signature schemes) depended upon such unproven hypotheses as the intractability of integer factorization, whereas their secret sharing scheme,

like Shamir's, does not. In fact, their scheme is secure even if the conspirators

have unlimited computational resources.

2.Their scheme is exactly as easy to implement as Shamir's, thus avoiding the complications of implementing an additional scheme.

Now we will see how to modify Shamir's scheme so that the probability of

undetected cheating is less than e, for any e > 0.

1.Choose any prime p > max((s-1)(k-1)/e + k, n).

2.Choose a1, a2,..…...ak-1€ Zp randomly, uniformly, and independently.

3.Let q(x) = D + a1x + a2x^2+.....+ ak-1x^k-1.

4.Choose (x1,x2,.....xn) uniformly and randomly from among all permutations of n distinct elements from {1, 2 . . . . ., p-1}. Let Di = (xi, di), where di = q(xi).

The main difference between this and Shamir's scheme occurs in step 4.

The proofs of properties given by Shamir are stated here -

(a) Any k participants can determine the secret uniquely by interpolation, since

the points x1, x2 ,...…..xn are distinct.

(b) Suppose participants i1 ,i2 ,.....ik-1 conspire to determine the secret without consulting participant ik. When the values of D and xi1 ,xi2 ,.....xik-1 are fixed, q(xi1), q(xi2),.....q(xik-1) are functions of the random variables a1, a2 ,.....ak-1.

Using the interpolation theorem and the mutual independence of a1, a2,.....ak-1 it can be shown that those k-1 values q(xi1), q(xi2) ,.....q(xik-1) are uniformly distributed and mutually independent. Hence, the secret shares

Di1, Di2, .....Dik-1 provide no more information about the value of D than do random numbers. (This proof is somewhat more general than Shamir's, since this assumes that D is chosen by some random process, or at least viewed that way by the conspirators.)

(c) It remains to explore the probability of deceiving another participant. It will be shown that property (c) holds even if the k-1 cheaters know q(x), and

hence know the secret. Suppose participants i1 , i2 ,.....ik-1 fabricate values (xi1', di1'), (xi2', di2') ,.....(xik-1', dik-1') to send to participant ik. Each possible secret D' € {0, 1. . . . . s - 1} defines a distinct polynomial qD(x) of degree at most k-1 passing through the point (0, D') and the fabricated points above. If D' not euals to D, such a polynomial qD(X) can intersect q(x) in at most k-1 points. Participant ik will reconstruct the incorrect secret D' only if qD(xik) = q(xik) and D' not equals to D. Recall that xik is a random element of {1, 2 . . . . . p - 1 } - {xi1, xi2 ,..... xik-1}. Thus for each polynomial qD(x) with D' not equals to D the probability that qD(xik) = q(xik) is at most (k-1)/(p-k). There are s-i legal but incorrect secrets, so the fabricated values yield s -1 corresponding polynomials. Any one of these polynomials would deceive participant ik with probability at most (k -1)/(p-k). Thus the probability of deceiving participant ik is at most (s-1)(k-1)/(p-k) < e.

It will now be shown that this scheme runs in expected time polynomial in k, n, log s, and log(1/e). It suffices to demonstrate that the expected time is polynomial in k, n, and log p, since p may always be chosen so that log p is linear in log k, log n, log s, and log(1/e). A certified prime p of this magnitude can be found in expected time polynomial in log p(1). The random choice of a1, a2 ,.....ak-1 and

(x1, x2 ,.....xn) can be done in expected time polynomial in k, n, and log p, as can the evaluation of q(x) at n points over Zp. Finally, interpolating k points over Zp can be done in time polynomial in k and log p [2], [8].

## 6. VERIFIABLE SECRET SHARING -

Oded Goldreich defines VSS as a secure multi-party protocol for computing the randomized functionality corresponding to some (non-verifiable) secret sharing scheme.

This scheme adds a verifiable feature to the shares that can be verified at the time of secret reconstruction.

A VSS scheme is required to withstand the following two types of active attacks:

1.A dealer sending inconsistent or incorrect shares to some of the participants during the distribution protocol, and

2.Participants submitting incorrect shares during the reconstruction protocol.

## 7. HOMOMORPHIC ENCRYPTION -

Homomorphic encryption is a form of encryption with an additional evaluation capability for computing over encrypted data without access to the secret key.

A homomorphic cryptosystem is like other forms of public encryption in that it uses a public key to encrypt data and allows only the individual with the matching private key to access its unencrypted data (though there are also examples of symmetric key homomorphic encryption as well). However, what sets it apart from other forms of encryption is that it uses an algebraic system to allow you or others to perform a variety of computations (or operations) on the encrypted data.

## 8. FELDSMAN'S SCHEME - AN ECAMPLE OF VSS -

It is based on Shamir's secret sharing scheme combined with any homomorphic encryption scheme.This scheme is, at best, secure for computationally bounded adversaries only.

At first, a cyclic group Zp* of prime order p, along with a generator g of Zp* or prime order q/p-1, is chosen publicly as a system parameter. The group Zp* must be chosen such that computing discrete logarithms is hard in this group.

Shamir's secret scheme is used to compute the shares of the secret x, using polynomial

f(x) = a0+ a1x+ ..... +at-1x^t-1(mod p)

To make these shares verifiable, the dealer distributes commitments to the coefficients - g^a0, g^a1, ..... g^at-1 and to the secret shares - zi = g^yi(mod p)

Once these are given, party i can verify their share:

g^a0 * (g^a1)^i * (g^a2)^i2 * ..... * (g^at-1)^it-1 (mod p) = g summation of (j = 0 to t)aji^j = g^yi.

# 8.1. Software realization of Feldman’s VSS:

```
# -*- coding: utf-8 -*-
"""
Created on Wed May  27 23:29:42 2020

@author: Sarvajeet
"""

import random
import math


# Helper function
def isprime(n):
    if n == 2:
        return True
```

```
    if n == 1 or n % 2 == 0:
        return False
    i = 3
    while i <= math.sqrt(n):
        if n % i == 0:
            return False
        i = i + 2
    return True


print("<<< Feldman's Verifiable Secret Sharing Scheme >>>")


# Pick q, p primes such that q | p - 1, that is equvalent to
# say that p = r*q + 1 for some r

# Choose q
while True:
    q = 127 #int(input("Insert a prime q: "))
    if isprime(q):
        break

print("\n\n<<< Choosing primes >>>")
print("\nq is the prime, ", end='')
print("q = " + str(q))
---------------------------------------------------------------

# Find p and r
r = 1
while True:
    p = r*q + 1
    if isprime(p):
        print("\nr = " + str(r))
        print("p is prime, ", end="")
        print("p = " + str(p))
        break
    r = r +1
---------------------------------------------------------------
# Compute elements of Z_p*
Z_p_star = []
for i in range(0, p):
    if(math.gcd(i,p) == 1):
        Z_p_star.append(i)

print("\n\n<<< Generating cyclic group >>>")
print("\nZ_p* = ")
print(Z_p_star)
---------------------------------------------------------------
# Compute elements of G = {h^r mod p | h in Z_p*}
G = []
for i in Z_p_star:
    G.append(i**r % p)
```

```
G = list(set(G))
G.sort()

print("\nG = ")
print(G)
print("\nOrder of G is " + str(len(G)) + ". This must be equal to q.")
-----------------------------------------------------------------

# Since the order of G is prime, any element of G except 1 is a generator
g = random.choice(list(filter(lambda g: g != 1, G)))
print("g = " + str(g))
print("G is the cyclic group which is generated by g")
# Secret taken from the group Z_q*
while True:
    a0 = 84 #int(input("Inser a secret in Z_q*: "))
    if a0 >= 1 or a0 <= q:
        break
# Secret polynomial coefficients taken from the group Z_q*
a1 = random.randint(1, q)
a2 = random.randint(1, q)

a = [a0, a1, a2]

print("\nLet the secret polynomial be: " + str(a0) + " + " +  str(a1) + "x + " + str(a2) +
"x^2")
print("(Secret polynomial coefficients taken from the group Z_q)")

-----------------------------------------------------------------
# The function f is a polynomial from the group Z_q* (for simplicity 2nd degree is
considered)
def f(x):
    return ((a0 + a1*x + a2*x**2) % q)

# List of shares
s = []
print("\n\n<<< Computing shares and verifying them >>>")

# Compute shares and verify
# Note that P_i receives as a share s = f(i)
# In general, each party P_1,...,P_6 could receive a different arbitrary share
for i in range(1,7):
    print("\ni = " + str(i))
    s.append(f(i))
    print("Share: f(" + str(i) + ") = " + str(f(i)))
    print("Commitment: g^f(" + str(i) + ") = " + str(g**f(i) % p))
    print("Verification: (g^a0)*((g^a1)^i)*((g^a2)^(i^2)) = " +
str((g**a0)*((g**a1)**i)*((g**a2)**(i**2)) % p) + "\n")


# Print shares
print("The list of shares are:", end=" ")
```

```
print(s)

# Parties cooperating to reconstruct the secret
B = [1,2,3]

def delta(i):
    d = 1
    print("i= "+str(i))
    #print(i)
    for j in B:
        if j != i:
            print("j= "+str(j))
            #print(j)
            d *= -j/(i - j)
    print("delta = " + str(d % q))
    return d

print("\n\n<<< Reconstructing the secret >>>")

a0_reconstructed = 0
for i in B:
    print("\nShare of P_" + str(i) + " is " + str(s[i - 1]))
    a0_reconstructed += delta(i)*s[i - 1]
print("\nThe secret is: " + str(a0_reconstructed % q))
```

## Output:

```
<<< Computing shares and verifying them >>>
i = 1
Share: f(1) = 82
Commitment: g^f(1) = 500
Verification: (g^a0)*((g^a1)^i)*((g^a2)^(i^2)) = 500
i = 2
Share: f(2) = 53
Commitment: g^f(2) = 409
Verification: (g^a0)*((g^a1)^i)*((g^a2)^(i^2)) = 409
i = 3
Share: f(3) = 124
Commitment: g^f(3) = 55
Verification: (g^a0)*((g^a1)^i)*((g^a2)^(i^2)) = 55
i = 4
Share: f(4) = 41
Commitment: g^f(4) = 394
Verification: (g^a0)*((g^a1)^i)*((g^a2)^(i^2)) = 394
i = 5
Share: f(5) = 58
Commitment: g^f(5) = 25
Verification: (g^a0)*((g^a1)^i)*((g^a2)^(i^2)) = 25
i = 6
Share: f(6) = 48
Commitment: g^f(6) = 400
```

Verification: (g^a0)*((g^a1)^i)*((g^a2)^(i^2)) = 400
The list of shares are: [82, 53, 124, 41, 58, 48]

# 9. References


9.1 Wikipedia.

9.2 Araki T.: Efficient (*k*, *n*) threshold secret sharing schemes secure against cheating from *n* – 1 cheaters. In: Proceedings of ACISP'07, LNCS, vol. 4586, pp. 13–142. Springer-Verlag (2007).

9.3 Bhndo C., De Santis A., Gargano L., Vaccaro U.: Secret sharing schemes with veto capabilities. In: Proceedings of the First French-Israeli Workshop on Algebraic Coding, LNCS, vol. 781, pp. 82–89. Springer-Verlag (1993).

9.4 Blakley G.R.: Safeguarding cryptographic keys. In: Proceedings of AFIPS'79, vol. 48, pp. 313–317 (1979).

9.5 How to share a secret with Cheaters – Martin Tompa and Heather Woll

9.6 A Practical Scheme for Non-interactive Verifiable Secret Sharing – Paul Feldman