CD3

FYP Final Report

A software implementation of Shamir's secret sharing scheme

by

RUAN Keda

CD3

Advised by

Prof. Cunsheng Ding

Submitted in partial fulfillment
of the requirements for COMP 4991
in the
Department of Computer Science
The Hong Kong University of Science and Technology
2014-2015

Date of submission: April 17, 2015

Contents

1 Introduction **5**
1.1 Overview . . . 5
1.2 Objectives . . . 5
1.3 Literature Survey . . . 6

2 Design & Preliminaries **7**
2.1 Design . . . 7
2.2 Fundamentals of Finite Field . . . 8
2.2.1 Finite Field . . . 8
2.2.2 Finite Field Arithmetic . . . 9
2.3 Secret Sharing and (t, n)-Threshold Scheme . . . 9
2.4 Shamir's Secret Sharing Scheme . . . 10
2.4.1 Secret Partition . . . 10
2.4.2 Secret Reconstruction . . . 10
2.5 Proof: Shamir's Secret Sharing Scheme is a (t, n)-Threshold Scheme . . . 11
2.5.1 Recoverability . . . 11
2.5.2 Secrecy . . . 12

3 Implementation **13**
3.1 Construct the Finite Field $\mathbb{F}_{2^m} = GF(2^m), 8 \leq m \leq 64$ . . . 13
3.1.1 Generate Irreducible Polynomial . . . 13
3.1.2 Finite Field Addition and Subtraction . . . 13
3.1.3 Finite Field Multiplication . . . 14
3.1.4 Finite Field Multiplicative Inverse . . . 15
3.2 Implement Shamir's Secret Sharing Scheme over $\mathbb{F}_{2^m} = GF(2^m), 8 \leq m \leq 64$ 17
3.2.1 Secret Partition . . . 17
3.2.2 Secret Reconstruction . . . 18
3.3 Web Application For Image and Text Sharing . . . 19
3.3.1 Image Sharing . . . 20
3.3.2 Image Sharing (Revised) . . . 21
3.3.3 Text Sharing . . . 23
3.4 Technical Challenges . . . 24
3.4.1 Speed Issue For Large Finite Field and Large Input Secret . . . 24

4 Testing **26**

4.1 Test Construction of Finite Field $\mathbb{F}_{2^m} = GF(2^m), 8 \leq m \leq 64$ 26

4.1.1 Test Finite Field Addition and Subtraction 26

4.1.2 Test Finite Field Multiplication 26

4.1.3 Test Finite Field Multiplicative Inverse 26

4.2 Test Implementation of Shamir's Secret Sharing Scheme 27

4.3 Test Web Application 28

5 Evaluation **29**

5.1 Accuracy 29

5.2 Efficiency 29

5.3 User-friendliness 30

6 Conclusion and Further Work **31**

6.1 Conclusion 31

6.2 Further Work 31

7 Project Progress and Planning **32**

7.1 Distribution of Work 32

7.2 GANTT Chart 32

8 References **33**

9 Appendix **34**

9.1 Minutes 34

9.1.1 Minutes for the 1st Project Meeting 34

9.1.2 Minutes for the 2nd Project Meeting 35

9.1.3 Minutes for the 3rd Project Meeting 36

9.1.4 Minutes for the 4th Project Meeting 37

9.2 Table of Irreducible Polynomials 38

1 Introduction

1.1 Overview

Ever since human beings began to communicate, how to protect people's own information is being concerned widely. And such question brings us to a subject called cryptography. In the daily life, there are cases that you need to share some classified information with others. But since you and other people do not trust each other, you may be afraid that the information would leak. Under such circumstance, you need a scheme to ensure that the information is secure and will not leak by any of you.

To visualize this, let us assume a situation: a father named A is making a will for his three sons named B, C, and D. The main part of the will is a map which is believed that can lead people to a mysterious place full of treasure. Because the treasure is so huge that the father cannot simply give to one of his sons. Also he is afraid that his sons cannot trust each other and the one who hold the map might just go to the treasure place and take everything without sharing to others. So the father need a scheme to make sure that the treasure is being found by all of the three sons and share together. A simplest way to solve such question is to split the map into three parts, and give each son with one piece. And this brings us to the concept of secret sharing. But this also bring to a question that how to make sure that all three parts are needed in order to recover the whole map while any two pieces combined cannot lead to the correct direction. Shamir's Secret Sharing Scheme[8] is used to solve such question.

1.2 Objectives

The goal of this project is to implement Shamir's Secret Sharing Scheme over $GF(2^m), 8 \leq m \leq 64$. Specifically, there are several steps to follow which focus on the following objectives:

1. Get familiar with Shamir's Secret Sharing Scheme.

2. Learn the basics of the finite fields.

3. Implement Shamir's Secret Sharing Scheme over $GF(2^m), 8 \leq m \leq 64$.

4. Build application using the implementation.

1.3 Literature Survey

I did online survey and found some real-world applications using Shamir's Secret Scheme and are related to the project:

1. "gfshare" in Ubuntu[3]
   According to the Ubuntu manuals, gfshare is a package that provides a library for implementing the sharing of secrets, and there are two tools for simple use-cases of the algorithm. The two tools are named gfsplit and gfcombine, which can split an arbitrary file into shares and combine a number of shares to form the original file.

2. The "ssss"[2]
   The "ssss" is an implementation of Shamir's secret sharing scheme for UNIX/Linux machines. The "ssss" does both: the generation of shares for a known secret and the reconstruction of a secret using user provided shares.

3. Sharing a secret on the Web[1]
   The "Sharing a secret on the web" is a web application that makes it easy and safe to share secret on the web, it is based on the Secret Sharing Scheme proposed by Shamir in 1979.

2 Design & Preliminaries

2.1 Design

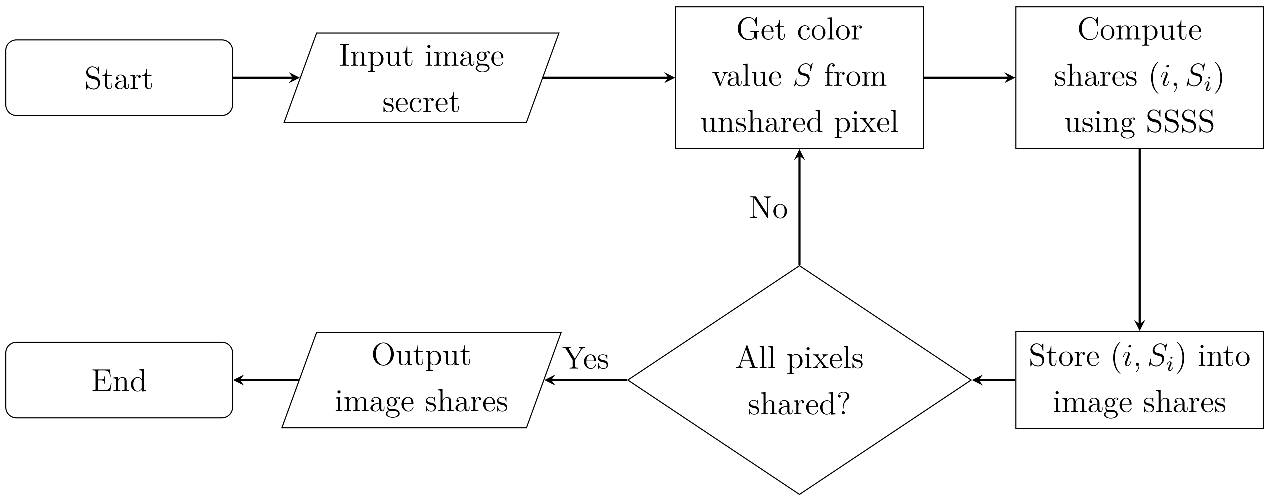
In this project, we build a web application that allows user to partition images and text into several shares, and to reconstruct image and text from shares.

The basic principle for this project is straight forward, and there are 3 steps to take to do the secret partition and reconstruction:

1. Input the secret (or shares).
2. Compute the shares (or secret) by using Shamir's Secret Sharing Scheme.
3. Output the shares (or secret).

To build that application, there are some sub-steps to take. And the following flow chart illustrates the image partition procedure of the application:

Image Patition


Start
Input image secret
Get color value S from unshared pixel
Compute shares (i, S_i) using SSSS
No
Store (i, S_i) into image shares
All pixels shared?
Yes
Output image shares
End

Note: "SSSS" stands for "Shamir's Secret Sharing Scheme".

The image reconstruction procedure is quite similar with the procedure shown above, it is operated with the opposite flow direction.

To start the project, we first need to learn the basics of the finite field and Shamir's Secret Sharing Scheme.

2.2 Fundamentals of Finite Field

2.2.1 Finite Field

A field[6], denoted as $\{\mathbb{F}, +, \times\}$, is a set $\mathbb{F}$ together with two binary operations, "+" and "$\times$", such that:

1. $\mathbb{F}$ is an Abelian group under "+", with identity element 0.

2. The nonzero elements of $\mathbb{F}$ form an Abelian group under "$\times$".

3. The distributive law $a \times (b + c) = a \times b + a \times c$ holds.

A finite field or Galois field, denoted as $\mathbb{F}_{p^n} = GF(p^n)$, is defined as

$$\begin{aligned} GF(p^n) = {} & (0, 1, 2, ..., p-1) \cup \\ & (p, p+1, p+2, ..., p^2+p-1) \cup \\ & (p^2, p^2+1, p^2+2, ..., p^2+p-1) \cup ... \cup \\ & (p^{n-1}, p^{n-1}+1, p^{n-1}+2, ..., p^{n-1}+p-1) \end{aligned}$$

where $p \in \mathbb{P}$ and $n \in \mathbb{Z}^+$. The number of elements in a finite field is called the *order* of the field. The order of the finite field is given by p^n where p is called the *characteristic* of the field.

In this report, we restrict discussion to $p = 2$, and the number that belongs to the finite field $\mathbb{F}_{2^m} = GF(2^m)$ over $\mathbb{F}_2 = GF(2)$.

To construct $\mathbb{F}_{2^m}$, we use a polynomial basis representation. Elements $a(x) \in \mathbb{F}_{2^m}$ can be regarded as polynomial

$$a(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + a_0$$

where $a_i \in \mathbb{F}_2$. This element associates with the vector of coefficients$(a_{m-1}, a_{m-2}, ..., a_0)$, and the bit pattern of the element is denoted as $a_{m-1}a_{m-2}...a_0$.

Arithmetic operations are performed followed by modulo an irreducible polynomial $m(x)$ of degree m

$$m(x) = x^m + k_{m-1}x^{m-1} + k_{m-2}x^{m-2+} ... + k_1x + k_0$$

where $k_i \in \mathbb{F}_2$.

2.2.2 Finite Field Arithmetic

Addition and Substation

Suppose $f(x)$ and $g(x)$ are polynomial elements in $\mathbb{F}_{2^m} = GF(2^m)$, and let $m(x)$ be an irreducible polynomial of degree m. If $h(x)$ denotes the sum(or difference) between $f(x)$ and $g(x)$, then

$$h(x) = f(x) \pm g(x) \ (\textbf{mod} \ m(x))$$

Multiplication and Multiplicative Inverse

Suppose $f(x)$ and $g(x)$ are polynomial elements in $\mathbb{F}_{2^m} = GF(2^m)$, and let $m(x)$ be an irreducible polynomial of degree m.
If $h(x)$ denotes the product of $f(x)$ and $g(x)$, then

$$h(x) = f(x) \times g(x) \ (\textbf{mod} \ m(x))$$

If $h(x)$ denotes the multiplicative inverse of $f(x)$, then

$$f(x) \times h(x) \ (\textbf{mod} \ m(x)) = 1.$$

The multiplicative inverse of $f(x)$ is also denoted as $f^{-1}(x)$.

2.3 Secret Sharing and (t, n)-Threshold Scheme

Secret sharing, proposed independently by Shamir[8] and Blakley[4] in 1979, refers to the methods that divide a secret into a number of shares, and distribute the shares to several participants, with each of whom allocate a share of the secret.

Let n denotes the number of participants. A scheme is called (t, n)-threshold scheme, if any group of t or more participants can together recover the secret S, but no group of fewer than t participants can.

A (t, n)-threshold scheme should satisfy the following two properties:

1. Recoverability: Given any t shares, secret S can be completely recovered.
2. Secrecy: Given any $t - 1$ or less shares, secret S cannot be obtained.

2.4 Shamir's Secret Sharing Scheme

Shamir's secret sharing scheme[8] is designed to share the secret S into n shares $(S_1, S_2, ..., S_n)$, such that any k shares can be used to reconstruct S. The numbers used in Shamir's Secret Sharing Scheme are chosen from the finite field $\mathbb{F}_p$, where p is a prime ($p \in \mathbb{P}$).

2.4.1 Secret Partition

To share the secret S, we first choose a prime $p > S$ and get the finite field $\mathbb{F}_p$. Then we randomly choose a_j $(j = 1, 2, ..., k-1)$ from $\mathbb{F}_p$, and there are to be the coefficients of degree $k-1$ polynomial. Let $a_0 = S$, and we form the following polynomial:

$$f(x) = S + a_1x + a_2x^2 + ... + a_{k-1}x^{k-1} \textbf{ mod } p$$

The partition of the secret is done by calculating $S_i = f(i)$ for $i = 1, 2, ..., n$. Together, i and its corresponding share S_i form a point (i, S_i) $(i = 1, 2, ..., n)$.

In this project, we use randomly chosen distinct x_i $(i = 1, 2, ..., n)$ from $\mathbb{F}_p$, and do the partition to get the shares $(x_i, S_i = f(x_i))$ $(i = 1, 2, ..., n)$.

2.4.2 Secret Reconstruction

To reconstruct the secret S, we choose any k points $(x_1, S_1), ..., (x_k, S_k)$, and put them into the polynomial

$$f(x) = a_0 + a_1x + a_2x^2 + ... + a_{k-1}x^{k-1}$$

Then we get the matrix-vector and can solve the coefficients $a_0, a_1, ..., a_{k-1}$ of the polynomial:

$$\begin{bmatrix} 1 & x_1 & \cdots & x_1^{k-1} \\ 1 & x_2 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & \cdots & x_k^{k-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_k \end{bmatrix} \longrightarrow \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^{k-1} \\ 1 & x_2 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & \cdots & x_k^{k-1} \end{bmatrix}^{-1} \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_k \end{bmatrix}$$

However, to calculate the inverse of a square matrix in $\mathbb{F}_p$ is difficult. Here we use Lagrange Interpolation to reconstruct the polynomial $f(x)$:

$$f(x) = \sum_{i=1}^{k} S_i \ell_i(x) \textbf{ mod } p$$

where $\ell_i(x)$ is the Lagrange basis polynomial:

$$\ell_i(x) = \prod_{1 \le j \le k, j \ne i} \frac{(x - x_j)}{(x_i - x_j)}$$

By Gaussian elimination, the secret S is the polynomial $f(x)$ evaluated at $x = 0$:

$$S = \sum_{i=1}^{k} S_i \ell_i(0) \textbf{ mod } p$$

2.5 Proof: Shamir's Secret Sharing Scheme is a (t, n)-Threshold Scheme

To prove Shamir's Secret Sharing Scheme is a (t, n)-Threshold Scheme, recoverability and secrecy should hold. The proof is as follows:

2.5.1 Recoverability

Given any t shares, secret S should be completely recovered.

Proof: Given t shares of (x_i, S_i) for $i \in \mathcal{G}$, we can have t points on the curve of an at-most $t-1$ degree polynomial. By using Lagrange Interpolation, we can get this an at-most $t-1$ degree polynomial:

$$f(x) = \sum_{i \in \mathcal{G}} S_i \prod_{j \in \mathcal{G}, j \neq i} \frac{(x - x_j)}{(x_i - x_j)}$$

To prove the recoverability, we need to prove the existence and uniqueness of the polynomial $f(x)$ hold.

For existence: To prove the existence, we need to obtain a polynomial that interpolates the point (x_i, S_i). Here, we define a sub-polynomial $\ell_i(x)$ that has the property that:

$$\ell_i(x_j) = \begin{cases} 0 & \text{if } j \neq i \\ 1 & \text{if } j = i \end{cases}$$

Observed that $\prod_{j \in \mathcal{G}, j \neq i}(x - x_j)$ is 0 if $x \neq x_i$, and its value at x_i is $\prod_{j \in \mathcal{G}, j \neq i}(x_i - x_j)$. Multiplying this polynomial and the value, we get the $t-1$ degree sub-polynomial

$$\ell_i(x) = \prod_{j \in \mathcal{G}, j \neq i} \frac{(x - x_j)}{(x_i - x_j)}$$

Then by multiplying S_i and $\ell_i(x)$, we get $f_i(x) = S_i \ell_i(x)$ with the property that $f_i(x_j) = 0$ if $j \neq i$ and $f_i(x_i) = S_i$. Therefore, $f_i(x)$ interpolates the point of (x_i, S_i). Then, the summation of $f_i(x)$ gives us the polynomial $f(x)$ that interpolates the points of (x_i, S_i) for $i \in \mathcal{G}$.
Thus, the existence holds.

For uniqueness: Suppose there are two different polynomials of at-most $t-1$ degree, named $p(x)$ and $q(x)$, that interpolate the points of (x_i, S_i) $(i \in \mathcal{G})$, then we can get $r(x) = p(x) - q(x)$ is also an at-most $t-1$ degree polynomial. Because both $p(x)$ and $q(x)$ interpolate t points, $r(i) = p(i) - q(i) = S_i - S_i = 0$ for $i \in \mathcal{G}$. If $r(x)$ is a non-zero polynomial, then $r(x)$ has t roots. But according to fundamental theorem of algebra, a $t-1$ degree polynomial $r(x)$ has only $t-1$ roots, which is contradicted with the statement that $r(x)$ has t roots. Then $r(x)$ can only be a zero polynomial, and $r(x) = 0$.

$$r(x) = 0 = p(x) - q(x) \to p(x) = q(x)$$

Then the uniqueness holds.

Thus, there exists an unique polynomial $f(x)$ based on t shares. And Gaussian elimination helps to find the coefficient at degree 0, which is the secret S.

$$S = f(0) = \sum_{i \in \mathcal{G}} S_i \prod_{j \in \mathcal{G}, j \neq i} \frac{-x_j}{x_i - x_j}$$

Thus, recoverability holds.

2.5.2 Secrecy

Given any $t-1$ or less shares, secret S cannot be obtained.

Proof: Suppose we only have $t-1$ shares (x_i, S_i) for $i \in \mathcal{G}$ to do the reconstruction, by using Lagrange Interpolation, we can only get an unique $t-2$ degree polynomial $p^*(x)$ and $S^* = p^*(0)$. Since the polynomial we used in partition stage is a $t-1$ degree polynomial, any constant ξ can be used to form the new polynomial

$$g(x) = p^*(x) + \xi \times \prod_{i \in \mathcal{G}} (x - x_i)$$

which interpolates $t-1$ points.

Let $\prod_{i \in \mathcal{G}} (-x_i) = c$, then c and S^* is fixed with given (x_i, S_i) $i \in \mathcal{G}$. Therefore, the secret S calculated by $S = g(0) = p*(0) + \xi \times c = S^* + \xi \times c$ can equal-likely be all values with different ξ, and correct secret S cannot be obtained.

Thus, secrecy holds.

3 Implementation

The application for image and text sharing requires byte value sharing to be done by using the implementation of Shamir's Secret Sharing Scheme, and the arithmetic operations in Shamir's Secret Sharing Scheme requires to be done in finite field $\mathbb{F}_{2^m} = GF(2^m)$.

To start with, we implement the finite field $\mathbb{F}_{2^m} = GF(2^m)$. Then we implement the Shamir's Secret Sharing Scheme. Finally, we build the application for image and text sharing.

3.1 Construct the Finite Field $\mathbb{F}_{2^m} = GF(2^m), 8 \leq m \leq 64$

3.1.1 Generate Irreducible Polynomial

Irreducible polynomials play an important role in finite field arithmetic operations. In $\mathbb{F}_{2^m}$, the arithmetic operations between polynomial elements need to modulo an irreducible polynomial $f(x)$ of degree m.

In this report, we use "Table of Low-Weight Binary Irreducible Polynomials" provided by Hewlett-Packard Laboratories[7] to get the irreducible polynomial $f(x)$ of degree m.

The table of irreducible polynomials of degree m ($8 \leq m \leq 64$) is available in the appendix.

3.1.2 Finite Field Addition and Subtraction

Let a_k, b_k, and c_k be the coefficients of x^k of $f(x)$, $g(x)$, and $h(x) = f(x) \pm g(x)$ respectively, where $f(x)$ and $g(x)$ are polynomial elements in $\mathbb{F}_{2^m}$, then

$$c_k = a_k \pm b_k \ (\mathbf{mod}\ 2),\ k = 0, 1, 2, ..., m-1$$

Since a_k and b_k are in $\mathbb{F}_2$, and the addition and subtraction in $\mathbb{F}_2$ are as follows:

$$0 + 0 \ (\mathbf{mod}\ 2) = 0,\ 0 - 0 \ (\mathbf{mod}\ 2) = 0$$

$$0 + 1 \ (\mathbf{mod}\ 2) = 1,\ 0 - 1 \ (\mathbf{mod}\ 2) = 1$$

$$1 + 0 \ (\mathbf{mod}\ 2) = 1,\ 1 - 0 \ (\mathbf{mod}\ 2) = 1$$

$$1 + 1 \ (\mathbf{mod}\ 2) = 0,\ 1 - 1 \ (\mathbf{mod}\ 2) = 0$$

The addition and subtraction can be regarded as exclusion-or (XOR) operation, and they are the same in finite field with the characteristic field is 2.

For addition and subtraction between polynomial elements in $\mathbb{F}_{2^m}$, we can use bit pattern to compute the sum and difference. The algorithm for addition and subtraction is as follows:

```
Algorithm 1 Addition and Subtraction in $\mathbb{F}_{2^m}$
Input: $B_1, B_2$ : bit pattern of $a_1(x)$ and $a_2(x)$ in $\mathbb{F}_{2^m}$
 1: $result \leftarrow B_1\ XOR\ B_2$
Output: $result$ : bit pattern of $a_1(x) \pm a_2(x)$ in $\mathbb{F}_{2^m}$
```

3.1.3 Finite Field Multiplication

Observed that

$$\begin{aligned} h(x) &= f(x) \times g(x) \textbf{ mod } m(x) \\ &= a_{m-1}(x^{m-1} \times g(x)) + a_{m-2}(x^{m-2} \times g(x)) + ... + a_1(x \times g(x)) + a_0 g(x) \textbf{ mod } m(x) \end{aligned}$$

we can iteratively compute $x^j \times g(x)$ **mod** $m(x)$ and add to the result $h(x)$ if $a_j = 1$. For $x^j \times g(x)$ **mod** $m(x)$, we can compute it by shifting bit pattern of $g(x)$ to left by j bits, and modulo by $m(x)$.

Starting from $x \times g(x)$, let $b_{m-1}b_{m-2}...b_1b_0$ be the bit pattern of $g(x)$, then the bit pattern of $x \times g(x)$ becomes $b_{m-1}b_{m-2}...b_1b_0 0$. If $b_{m-1} = 0$, then $x \times g(x)$ is still in $\mathbb{F}_{2^m}$ and no further operation needs to be done. If $b_{m-1} = 1$, then we need to modulo $x \times g(x)$ by $m(x)$. The modulo operation is as follows:

$$\begin{aligned} x \times g(x) \textbf{ mod } m(x) &= (x^m + b_{m-2}x^{m-1} + ... + b_1x^2 + b_0x) \textbf{ mod } m(x) \\ &= (b_{m-2}x^{m-1} + ... + b_1x^2 + b_0x) + (x^m \textbf{ mod } m(x)) \\ &= (b_{m-2}x^{m-1} + ... + b_1x^2 + b_0x) + (m(x) - x^m) \\ &= (b_{m-2}b_{m-3}...b_1b_0)\ XOR \text{ bit pattern of } (m(x) - x^m) \end{aligned}$$

In general, let B_1 and B_2 denote the bit pattern of polynomial elements of $a_1(x)$ and $a_2(x)$ in $\mathbb{F}_{2^m}$, and B_m^* denote the bit pattern of $(m(x) - x^m)$ where $m(x)$ is an irreducible polynomials of degree m.

If B_2 consists a single bit in j^{th} position from the right, then B_1 multiplies by x for j times. If the previous B_1's MSB is 0, shift the B_1 to the left by j bits and insert j bits of 0 from the right. If B_1's MSB is 1, shift the B_1 to the left as above, and then take XOR operation with B_m^* for the result.

If B_2 is an arbitrary bit pattern, we consider the bit pattern to be a sum of bit patterns with each containing only single bit. The algorithm for multiplication is shown as follows:

Algorithm 2 Multiplication in $\mathbb{F}_2^m$

```
Input: $f(x), g(x) \in \mathbb{F}_{2^m}$ with coefficients $a_k$ and $b_k$ $(k = 0, 1, ..., m-1)$ respectively
1: if $a_0 = 1$ then
2:     $h(x) \leftarrow g(x)$
3: else
4:     $h(x) \leftarrow 0$
5: end if
6: for $i$ from 1 to $m-1$ do
7:     $g(x) \leftarrow x \times g(x)$ mod $m(x)$
8:     if $a_i = 0$ then
9:         $h(x) \leftarrow h(x) + g(x)$
10:    end if
11: end for
Output: $h(x) = f(x) \times g(x)$ mod $m(x)$
```

3.1.4 Finite Field Multiplicative Inverse

To compute the multiplication inverse, we use Extended Euclidean Algorithm.

The greatest common divisor of any two polynomials ($a(x)$ and $m(x)$) can be expressed as a linear combination of two polynomials ($r(x)$ and $s(x)$),

$$a(x) \times r(x) + m(x) \times s(x) = GCD(a(x), m(x))$$

Extended Euclidean Algorithm can be used to compute $r(x)$ and $s(x)$, and $GCD(a(x), m(x))$. The algorithm is as follows:

Algorithm 3 Extended Euclidean Algorithm

```
Input: $a(x), m(x) : a(x) \in \mathbb{F}_{2^m}, m(x)$ is irreducible polynomial of degree $m$
1: $r(x) \leftarrow 0$;    $old\_r(x) \leftarrow 1$;
2: $s(x) \leftarrow 1$;    $old\_s(x) \leftarrow 0$;
3: $t(x) \leftarrow m(x)$;    $old\_t(x) \leftarrow a(x)$;
4: repeat
5:     $q(x) \leftarrow old\_t(x)$ div $t(x)$
6:     $tmp \leftarrow t(x);\ t(x) \leftarrow old\_t(x) - q(x) \times t(x);\ old\_t(x) \leftarrow tmp$
7:     $tmp \leftarrow r(x);\ r(x) \leftarrow old\_r(x) - q(x) \times r(x);\ old\_r(x) \leftarrow tmp$
8:     $tmp \leftarrow s(x);\ s(x) \leftarrow old\_s(x) - q(x) \times s(x);\ old\_s(x) \leftarrow tmp$
9: until $t(x) = 0$
Output: $old\_r(x), old\_s(x), old\_t(x)$
```

From the output, we get: $old\_t(x) = GCD(a(x), m(x)) = a(x) \times old\_r(x) + m(x) \times old\_s(x)$.

Let $m(x)$ be an irreducible polynomial with degree m, and $a(x)$ be the polynomial element in $\mathbb{F}_{2^m}$, then $GCD(a(x), m(x)) = 1$, and the following holds:

$$a(x) \times old\_r(x) + m(x) \times old\_s(x) \equiv 1 \ (\textbf{mod}\ m(x))$$

This implies

$$a(x) \times old\_r(x) \equiv 1 \ (\textbf{mod}\ m(x))$$

Thus $old\_r(x)$ is the inverse of $a(x)$ $(\textbf{mod}\ m(x))$. We use $a^{-1}(x)$ to denote the multiplicative inverse of $a(x)$, $a^{-1}(x) = old\_r(x)$ $(\textbf{mod}\ m(x))$.

In this report, we use a variant of the Extended Euclidean Algorithm reported by Hankerson et al.[5] to compute the multiplicative inverse of a non-zero element in $\mathbb{F}_{2^m}$.
The algorithm for multiplicative inverse is as follows, where $deg(x)$ is the function to calculate the degree of a polynomial.

Algorithm 4 Finite Field Multiplicative Inverse in $\mathbb{F}_{2^m}$

```
Input: $a(x), m(x) : a(x) \in \mathbb{F}_{2^m}, m(x)$ is irreducible polynomial of degree $m$
1: $r(x) \leftarrow 1; \quad u(x) \leftarrow a(x)$
2: $s(x) \leftarrow 0; \quad v(x) \leftarrow m(x)$
3: while $deg(u(x)) \neq 0$ do
4:     $j \leftarrow deg(u(x)) - deg(v(x))$
5:     if $j < 0$ then
6:         $tmp \leftarrow u(x);\ u(x) \leftarrow v(x);\ v(x) \leftarrow tmp$
7:         $tmp \leftarrow r(x);\ r(x) \leftarrow s(x);\ s(x) \leftarrow tmp$
8:         $j \leftarrow -j$
9:     end if
10:    $s(x) \leftarrow s(x) + r(x) \times x^j$
11:    $u(x) \leftarrow u(x) + v(x) \times x^j$
12: end while
Output: $r(x) : r(x)$ is the multiplicative inverse of $a(x)$
```

From the output, we get $r(x) = a^{-1}(x)$.

3.2 Implement Shamir's Secret Sharing Scheme over $\mathbb{F}_{2^m} = GF(2^m), 8 \leq m \leq 64$

The implementation consists of secret partition and secret reconstruction functions. These functions require computational operations, and all the operations are done in $\mathbb{F}_{2^m} = GF(2^m)$.

3.2.1 Secret Partition

The secret partition is done by computing the following polynomial

$$S_i = f(x_i) = S + a_1 x_i + a_2 x_i^2 + ... + a_{k-1} x_i^{k-1} \textbf{ mod } m(x)$$

where x_i $(i = 1, 2, ..., n)$ are distinct and randomly chosen from $\mathbb{F}_{2^m}$, a_j $(j = 1, 2, ..., k-1)$ are also randomly chosen from $\mathbb{F}_{2^m}$, and $m(x)$ is an irreducible polynomial of degree$m$.

For fast calculation of polynomial $f(x_i)$, we use Horner's method. We first define the sequence of $\{b_n\}$:

$$b_{k-1} = a_{k-1},\ b_j = a_j + b_{j+1} x_i\ (j = 1, 2, ..., k-2)$$

and $f(x_i)$ becomes:

$$\begin{aligned} f(x_i) &= S + x_i(a_1 + x_i(a_2 + x_i(a_3 + ... + x_i(a_{k-2} + a_{k-1}x_i)...))) \\ &= S + x_i(a_1 + x_i(a_2 + x_i(a_3 + ... + x_i(a_{k-2} + b_{k-1}x_i)...))) \\ &= S + x_i(a_1 + x_i(a_2 + x_i(a_3 + ... + x_i(b_{k-2})...))) \\ &= \vdots \\ &= S + x_i(b_1) \end{aligned}$$

The algorithm for secret partition is as follows:

Algorithm 5 Secret Partition in $\mathbb{F}_{2^m}$

```
Input: $S,\ n,\ k$ : $S$ is the value of secret, $n$ and $k$ are parameters
1: randomly choose $a_i$ $(i = 1, 2, ..., k-1)$ from $\mathbb{F}_{2^m}$
2: for $i$ from 1 to $n$ do
3:     randomly choose $x_i$ from $\mathbb{F}_{2^m}$ where $x_i \neq x_p$ $(p = 1, 2, ..., i-1))$
4:     $b_{k-1} \leftarrow a_{k-1}$
5:     for $j$ from $k-2$ to 1 do
6:         $b_j \leftarrow a_j + b_{j+1} \times x_i$
7:     end for
8:     $S_i \leftarrow S + x_i \times b_1$
9: end for
Output: $(x_i, S_i)$ $(i = 1, 2, ..., n)$
```

From the output, we get n shares of (x_i, S_i) $(i = 1, 2, ..., n)$.

3.2.2 Secret Reconstruction

The secret reconstruction is done by solving the polynomial

$$f(x) = a_0 + a_1x + a_2x^2 + ... + a_{k-1}x^{k-1}$$

with the given k points $(x_1, S_1), ..., (x_k, S_k)$. And the result of coefficient a_0 of the polynomial $f(x)$ is the secret S.

In this report, we use Lagrange Interpolation to reconstruct:

$$f(x) = \sum_{i=1}^{k} S_i \prod_{1 \leq j \leq k, j \neq i} \frac{(x - x_j)}{(x_i - x_j)}$$

The secret S is the polynomial $f(x)$ evaluated at $x = 0$:

$$S = f(0) = \sum_{i=1}^{k} S_i \prod_{1 \leq j \leq k, j \neq i} \frac{-x_j}{x_i - x_j}$$

The algorithm for secret reconstruction is as follows:

Algorithm 6 Secret Reconstruction in $\mathbb{F}_{2^m}$

```
Input: $(x_i, S_i)$ $(i = 1, 2, ..., k)$ : $k$ shares of secret
 1: $result \leftarrow 0$
 2: for $i$ from 1 to $k$ do
 3:     $p \leftarrow 1$;  $q \leftarrow 1$
 4:     for $j$ from 1 to $k$ do
 5:         if i=j then
 6:             pass
 7:         else
 8:             $p \leftarrow p \times (-x_j)$
 9:             $q \leftarrow q \times (x_i - x_j)$
10:         end if
11:     end for
12:     $result \leftarrow result + S_i \times (p \times q^{-1})$
13: end for
Output: $result$ : value of secret $S$
```

From the output, we get the initial value of secret S.

3.3 Web Application For Image and Text Sharing

In this project, we have implemented the finite field $\mathbb{F}_{2^m} = GF(2^m), 8 \leq m \leq 64$, and the Shamir's Secret Sharing Scheme over that finite field. Then we use this scheme implementation to build a web application for image sharing and text sharing.

The web application has the following functions:

1. upload image/text secret and download shares with corresponding n and k
2. upload shares and reconstruct the original image/text secret if there is enough shares

3.3.1 Image Sharing

The image file format is very orderly, the file contains image header and image data. The image data is a pixel array containing the pixel data for each pixel. Different image file type has different pixel format, and in this report, we use grayscale bitmap as example.

The pixel format of grayscale bitmap is 8-bit per pixel, and each pixel has color value of a 8-bit binary number. Thus, the color value for each pixel ranges from 0 to 255, and this characteristic enables us to use $\mathbb{F}_{2^8} = GF(2^8)$ for image sharing.

Image Partition

By reading the color value for each pixel in the bitmap, a secret S ($S \in \mathbb{F}_{2^8} = GF(2^8)$) is obtained. The set of shares $\{(x_i, S_i)|i = 1, 2, ..., n\}$ for this secret S can be obtained by using the implementation of Secret Partition. Then (x_i, S_i) is delivered to i-th share.

The following figure illustrates this procedure:

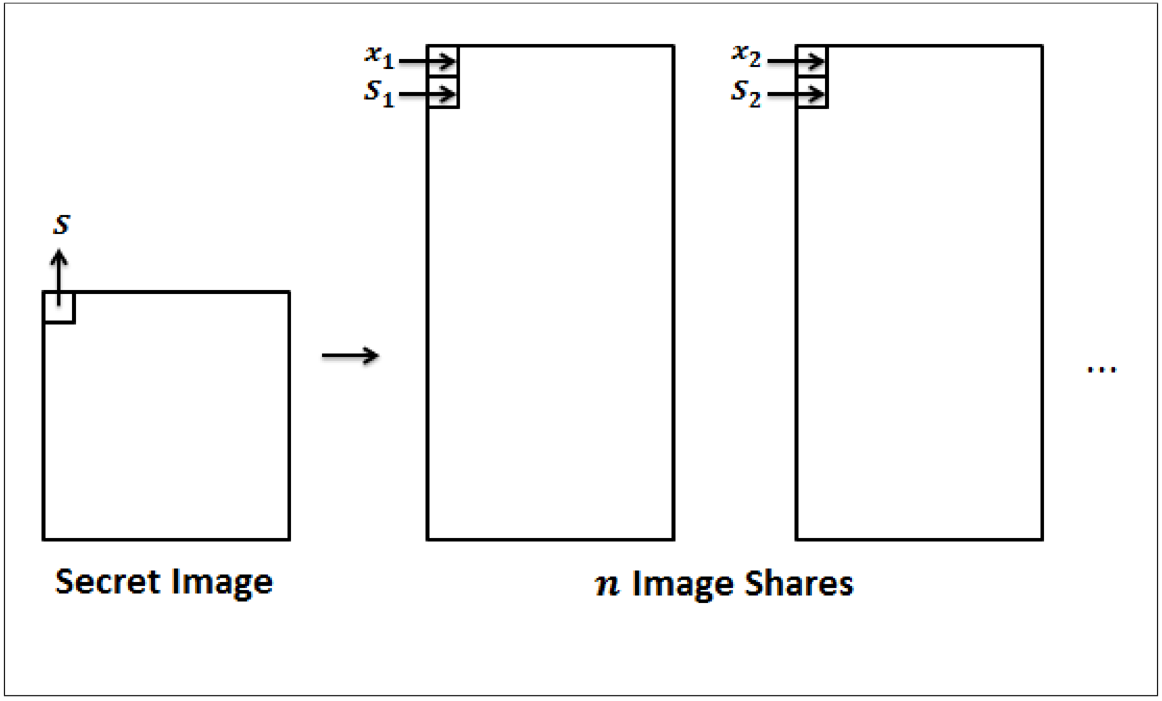


Figure 1: Image Sharing Procedure

From the output, we can get n image shares with each twice size as the secret image.

Image Reconstruction

The point of shares $(x_i, S_i)(i = 1, 2, ..., n)$ are saved consecutively in the image shares, with the first pixel saving x_i and second pixel saving S_i. By reading the color values of every two consecutive pixel, the point of shares $(x_i, S_i)(i = 1, 2, ..., n)$ can be obtained.

Thus, the initial color value S of the pixel can be obtained by using the implementation of Secret Reconstruction.

Example

Here, we use 512×512 grayscale Lena as the secret image, and we choose $n = 3, k = 2$. The following figure shows the secret image (Lena), the shares, and the reconstructed image.

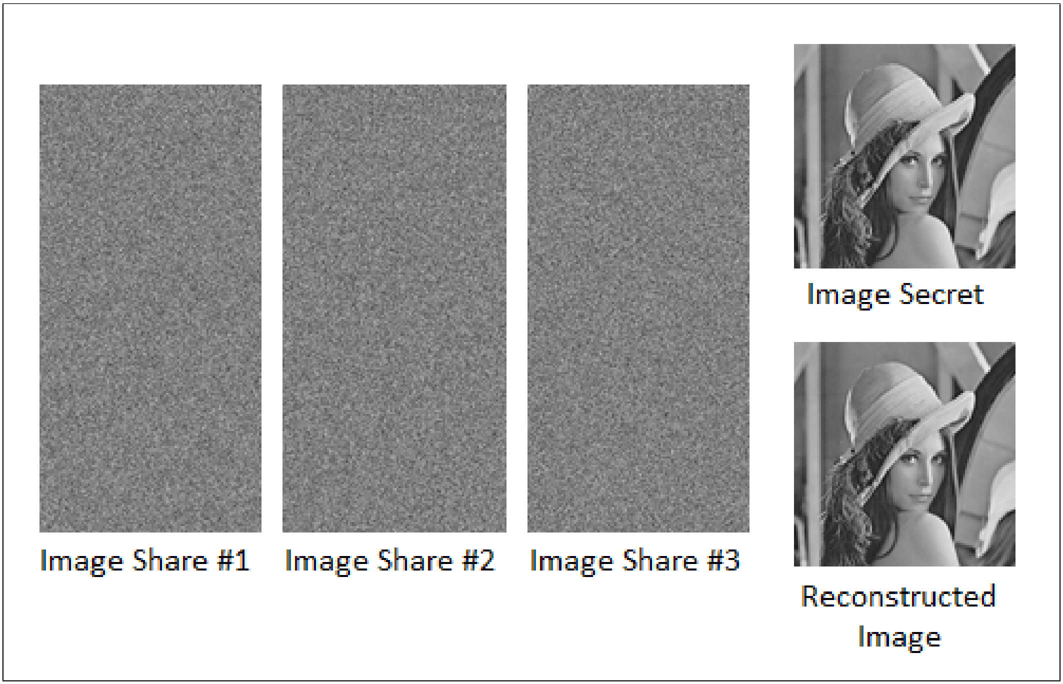


Figure 2: Image Secret, Image Shares, and Reconstructed Image

3.3.2 Image Sharing (Revised)

After testing, we find that it is unnecessary to use random x_i to generate shares (x_i, S_i). Using i for $i = 1, 2, ..., n$ to get the shares (i, S_i) is still secure because Shamir's Secret Sharing Scheme guarantee the secrecy. Therefore, we use $i = 1, 2, ..., n$ as the index in the revised version application for image sharing. Then the secret S is calculated by:

$$S = f(0) = \sum_{i\in\mathcal{G}} S_i \prod_{j\in\mathcal{G}, j\neq i} \frac{-j}{i-j}$$

Meanwhile, since there is no need to store x_i inside the image shares, this revised version saves space for image shares. Thus, each image shares are equally same size as the initial image secret, storing S_i inside image and i in the file name.

The following figure illustrates this procedure:

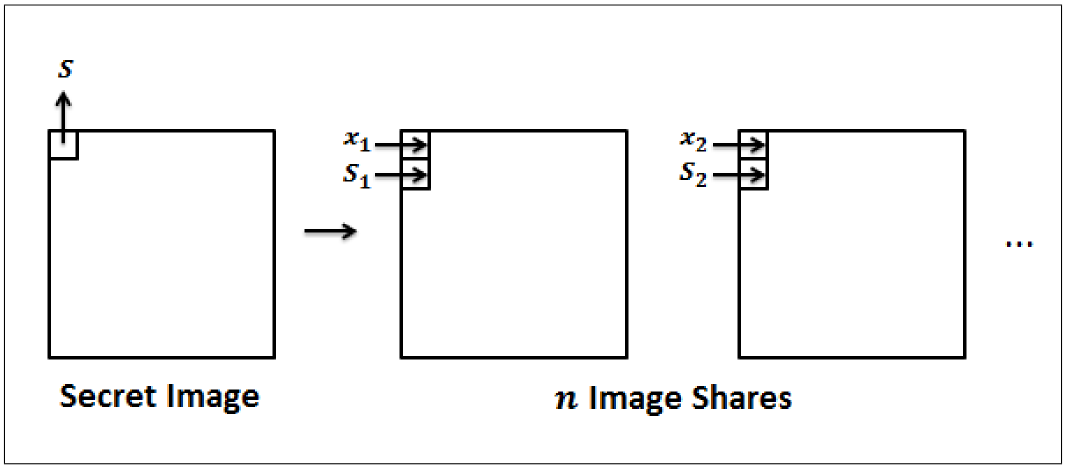


Figure 3: Image Secret, Image Shares, and Reconstructed Image

Example

Here, we use 512×512 grayscale Lena as the secret image, and we choose $n = 3, k = 2$. The following figure shows the secret image (Lena), the shares, and the reconstructed image.

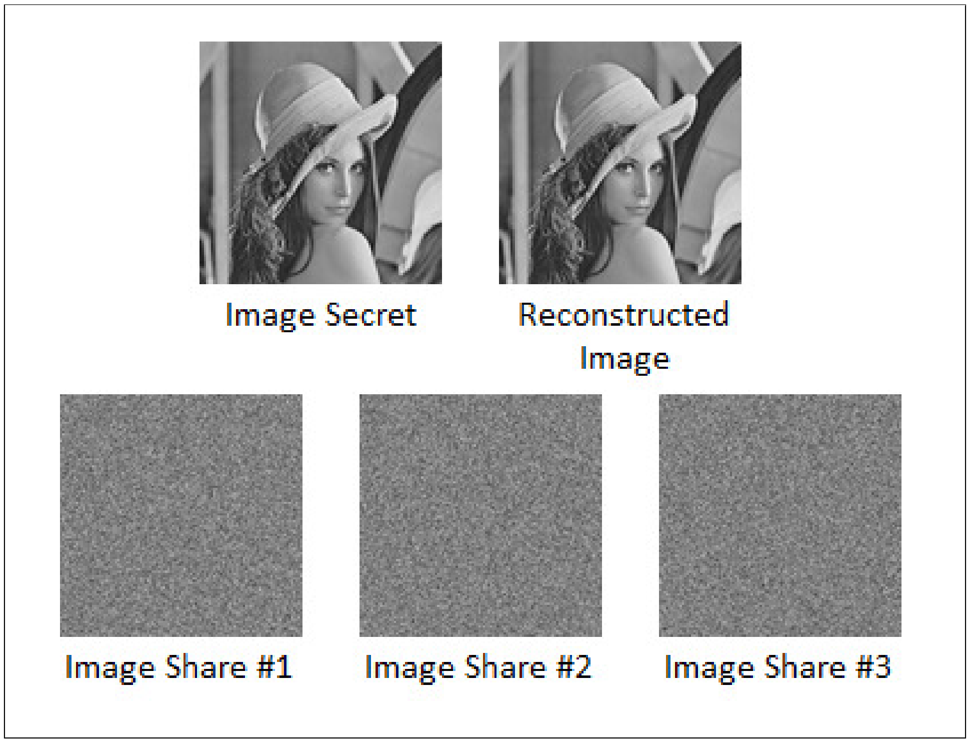


Figure 4: Image Secret, Image Shares, and Reconstructed Image with revised procedure

3.3.3 Text Sharing

Text sharing function is done by reading the character from the text, and use its ASCII code value as the secret. The shares and the initial secret can be obtained by using the implementation of Secret Partition and Secret Reconstruction.

In this report, we implement the Shamir's Secret Sharing Scheme over $\mathbb{F}_{2^m} = GF(2^m), 8 \leq m \leq 64$, and different m leads to different shares of (x_i, S_i). To reduce the space of the shares, we save the shares in hexadecimal form.

Example

Here, we choose $n = 3, k = 2, m = 20$. The following are the text secret, text shares, and reconstructed text.

Listing 1: Text Secret

```
Hello.
This is a text file for demo.
These text are the secret needed to share.
```

Listing 2: Test Shares #1

```
7d6a99a55c04c13eb9b301470cbffcb7a6310b05171aee43364547636523ebea89c2c42
e9d0fc0f47c6df6a5ea343a430335de9071809495bd5f0c9dd3ae03926ceae4f045d799
85d2ce076761d530fab72669a375eff8c8350eb82489dbb92e3da63db36e7654ee3fa86
4c295a8d716e0efac4bcb41722f5f6e8a00855de15ceefc9393ae9b632b6d4b895deaff
59661908f6b0de8e120f03222280bd40784aba66a2e9ce1c20f763473cf4abdaeccb343
310cc340f66d10e4bb5a0d78b6756787f838c32f
```

Listing 3: Test Shares #2

```
fad8a34a1e09892d73db0285197f836f4d1216f62e3e4c86de8a8796ca26d7de2385145
d3c0f8142f8dded4b48687e7606e2bd29b301482b709e1988a757d72477d5cf708b0433
00259c98cec5aa61586e46a34654dffa006a67704873b7875c70cc7bc9dce5f9dcd150c
f152bccae27a1dfd0979f42e43ebed7340195bbc859dd91272e1d367756d07712d3d5f7
62cc9411e731bd8a2414f644d8017c10f02674c725d333384b2ec62179e387b5b196613
621296818cda2899760c1afb5cea580ffa018625
```

Listing 4: Test Shares #3

```
87b6baef270d4ed3ca0403c4e5c051d8eb831da7392222c581cfc7c5af253c323a47a37
3a30041d784b21beed65c426505af63be8281fcbecba1157c74ff14b67e3f2980ceb5aa
830752eda9a67f51c6d9669ae54c3004385f47c86c5a6c6a724bea461fb2949d328bf8a
9d7be107937113004dc53439631e1bfae0162e62f55334db4baf3ad7c7dbb6c9bae3f0f
0baae81917b16376361da5668e81c3508802cea7d73a98246d99a50345116c6f7d5d524
5318a5c15ab73f4dcd3e1785fa9f4d8804694524
```

Listing 5: Reconstructed Text

```
Hello and welcome.
This is a text file for demo.
These text are the secret needed to share.
```

3.4 Technical Challenges

There are three key challenges in the implementation phase, and they are being specified as follows:

3.4.1 Speed Issue For Large Finite Field and Large Input Secret

As what shown previously about the algorithms for finite field arithmetic operations, the finite field addition and subtraction take only $O(1)$ time, which is independent of degree m. But the finite field multiplication and multiplicative inverse operations take $O(m)$ and $O(m)$ time. And the secret partition and reconstruction take $O(nk)$ and $O(k^2)$ finite field arithmetic operations respectively. This leads to a heavy computational cost for partition and reconstruction.

Thus, the scheme implemented works slowly when:

- The finite field used is large.
  The finite field multiplication and multiplicative inverse calculation take loops, which slows down the speed.

- The input secret is large.
  For example, the standard test image of 512×512 8-bit grayscale bitmap, Lena512.bmp, takes 263kB space, and there are 262,144 pixels in it. And for each pixel it takes $O(nk) \times O(m)$ time to generate shares. This works slowly (test data shown in test phase).

Solution

In this report, we take the following methods to improve the speed:

- For large finite field issue:
  To increase the speed of finite field arithmetic operation, we use Lookup Table method. Since the results are stored in advance, it only take $O(1)$ time to obtain the result for multiplication and multiplicative inverse.
  For example, if $m = 8$, there are $2^8 = 256$ elements in $\mathbb{F}_{2^8}$, and the table size for multiplication is 256×256, and table size for multiplicative inverse is 256.
  However, if the m become large ($m > 10$), the space for table become very large ($>$ 10MB). The size for table increases exponentially with m increases, then it become inappropriate to use Lookup Table method. This brings us to use the following method that use small finite field for large input secret.

- For large input secret:
  Observed that in this report, the image we used is a 8-bit per pixel (8bpp) format bitmap, and each pixel has color value of a 8-bit binary number, ranging from 0 to 255. Then this characteristic enables us to use $\mathbb{F}_{2^8} = GF(2^8)$ for image sharing, which also makes it convenient to store shares S_i into image shares.
  Together with Lookup Table method, the efficiency for image sharing improves dramatically.

4 Testing

4.1 Test Construction of Finite Field $\mathbb{F}_{2^m} = GF(2^m), 8 \leq m \leq 64$

To test the construction of $\mathbb{F}_{2^m}$, we need to test the implementation of finite field arithmetic operations: addition, subtraction, multiplication, and multiplicative inverse.

4.1.1 Test Finite Field Addition and Subtraction

The addition and subtraction between polynomial elements in $\mathbb{F}_{2^m}$ are the exclusion-or (XOR) operation between their bit pattern, and there is no need to test this bitwise operation.

4.1.2 Test Finite Field Multiplication

Suppose $f(x)$ and $g(x)$ are polynomial elements in $\mathbb{F}_{2^m}$, and let $m(x)$ be a irreducible polynomial of degree m. And a_i, b_i $(i = 0, 1, ..., m-1)$ are coefficients of $f(x)$ and $g(x)$. Then the product of $f(x)$ and $g(x)$, denoted by $h(x)$ is computed by

$$\begin{aligned} h(x) &= f(x) \times g(x) \ \mathbf{mod} \ m(x) \\ &= (a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1 x + a_0) \times g(x) \ \mathbf{mod} \ m(x) \\ &= \sum_{j=0}^{m-1} (a_j x^j \times g(x) \ \mathbf{mod} \ m(x)) \end{aligned}$$

Since $x^j \times g(x) \ mod \ m(x)$ can be iteratively computed by $x \times g(x) \ \mathbf{mod} \ m(x)$, and from part 3.1.3, we know that

$$x \times g(x) \ \mathbf{mod} \ m(x) = (b_{m-2}b_{m-3}...b_1 b_0) \ XOR \text{ bit pattern of } (m(x) - x^m)$$

which is also an bitwise operation of XOR, there is also no need to test.

4.1.3 Test Finite Field Multiplicative Inverse

To test multiplicative inverse, we use Unit Testing to test the function of computation.
First choose a random non-zero polynomials $a(x)$ in $\mathbb{F}_{2^m}$, and compute its multiplicative inverse $a^{-1}(x)$ by the implementation of multiplicative inverse algorithm in part 3.1.4. Then check whether $a(x) \times a^{-1}(x) == 1$. The test algorithm is as follows:

Algorithm 7 Test Multiplicative Inverse in $\mathbb{F}_{2^m}$

```
1: randomly pick $a(x) \in \mathbb{F}_{2^m}$
2: $a_{inv}(x) \leftarrow a^{-1}(x)$ computed by multiplicative inverse function in part 3.1.4
3: $prod \leftarrow a_{inv}(x) \times a^{-1}(x)$
4: if $prod == 1$ then
5:     assert $True$
6: else
7:     assert $False$
8: end if
```

The result turns out the implementation is accurate.

4.2 Test Implementation of Shamir's Secret Sharing Scheme

To test the implementation, we use Black-box Testing to test the functionality.

We input the secret S, and get the output S_0, then check whether $S == S_0$ The test algorithm is as follows:

Algorithm 8 Test Implementation of Shamir's Secret Sharing Scheme

```
1: randomly pick $S \in \mathbb{F}_{2^m}$
2: randomly pick $n$ and $k$ where $k \leq n$, and $k, n \in \mathbb{Z}^+$
3: randomly pick $n$ number $x_i$ $(i = 1, 2, ..., n) \in \mathbb{F}_{2^m}$, and $x_i \neq x_j$ for $i \neq j$
4: randomly pick $k - 1$ number $a_i$ $(i = 1, 2, ..., k - 1) \in \mathbb{F}_{2^m}$
5: for $i$ from 1 to $n$ do
6:     $S_i \leftarrow f(x_i)$ where $f(x_i) = S + a_1 x_i + a_2 x_i^2 + ... + a_{k-1} x_i^{k-1}$ $mod\ m(x)$
7: end for
8: randomly pick $k$ points of $(x_i, S_i)$ $(1 \leq i \leq n)$ from $\{(x_1, S_1), (x_2, S_2), ..., (x_n, S_n)\}$
9: $S_0 \leftarrow$ output of Secret Reconstruction($k$ points of $(x_i, S_i)$)
10: if $S_0 == S$ then
11:     assert $True$
12: else
13:     assert $False$
14: end if
```

The result turns out the implementation is accurate.

4.3 Test Web Application

To test the application, we need to test both image sharing function and text sharing function.

For text sharing, we read the text byte by byte and do the partition and reconstruction for each byte, and each byte is an ASCII code value between 0 and 255. Since the finite field we used is $\mathbb{F}_{2^m}, 8 \leq m \leq 64$, each byte value is an element in $\in \mathbb{F}_{2^m}$, and we have tested that in the previous phase.

For image sharing, we use five well-known grayscale bitmap of the size 512×512 to test the image sharing function.

Figure 5: Test Images

The result turns out the reconstructed image is same as the image secret, and the implementation is correct.

5 Evaluation

The evaluation of the implementation is carried out according to the following 3 aspects:

5.1 Accuracy

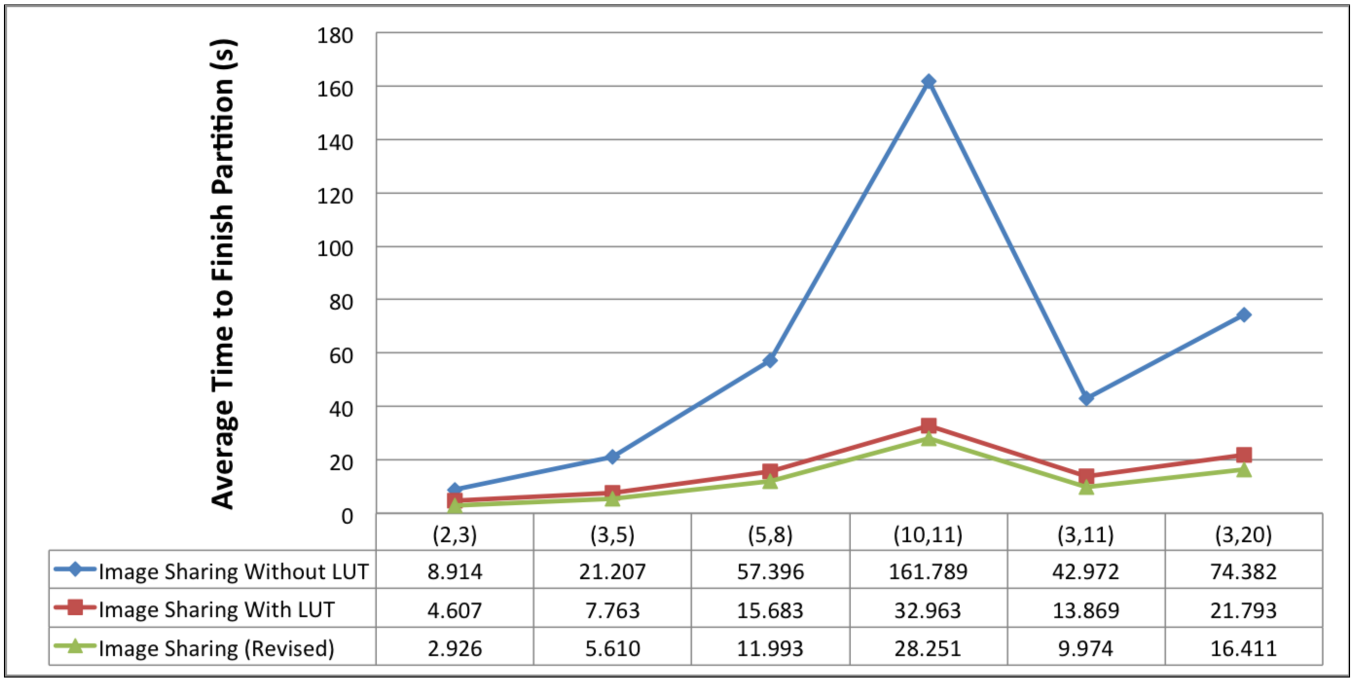
We use Unit Testing to test the construction of the Finite Field $\mathbb{F}_{2^m} = GF(2^m),\ 8 \leq m \leq 64$, and the result turns out that the construction is accurate.
We use Black-box Testing to test the implementation of Shamir's Secret Sharing Scheme, and the result turns out that the implementation is accurate.
We use Black-box Testing and other testing techniques to test the application after finishing the application implementation.
The result turns out all the implementation is accurate.

5.2 Efficiency

The efficiency is evaluated by how fast the Secret Partition and the Secret Reconstruction to be finished. The shorter time needed to be finished, the better efficiency. However, due to the heavy computational cost for division and reconstruction processes, the scheme works slowly.

In Technical Challenges phase, we discuss about the speed issue, and provide the solution to that issue. In Test phase, we use different methods to test the efficiency, and the test result will be evaluated as follows.

Here, we choose $(k, n) = (2, 3), (3, 5), (5, 8), (10, 11), (3, 11)$ and $(3, 20)$ to share the 512×512 grayscale standard test image (lena). The following figure shows the result of speed for image partition:



	(2,3)	(3,5)	(5,8)	(10,11)	(3,11)	(3,20)
◆ Image Sharing Without LUT	8.914	21.207	57.396	161.789	42.972	74.382
■ Image Sharing With LUT	4.607	7.763	15.683	32.963	13.869	21.793
▲ Image Sharing (Revised)	2.926	5.610	11.993	28.251	9.974	16.411

Figure 6: Image Partition Time

From the result, we can find that the Lookup Table method increases the speed for image sharing dramatically because of its $O(1)$ execution time. And the revised version of image sharing application also increase the speed, because it reduce the time for generate random index $x_i \in \mathbb{F}_{2^m}$. Since the revised version application also saves the storage space for image shares, it is the better solution for efficiency.

5.3 User-friendliness

6 Conclusion and Further Work

6.1 Conclusion

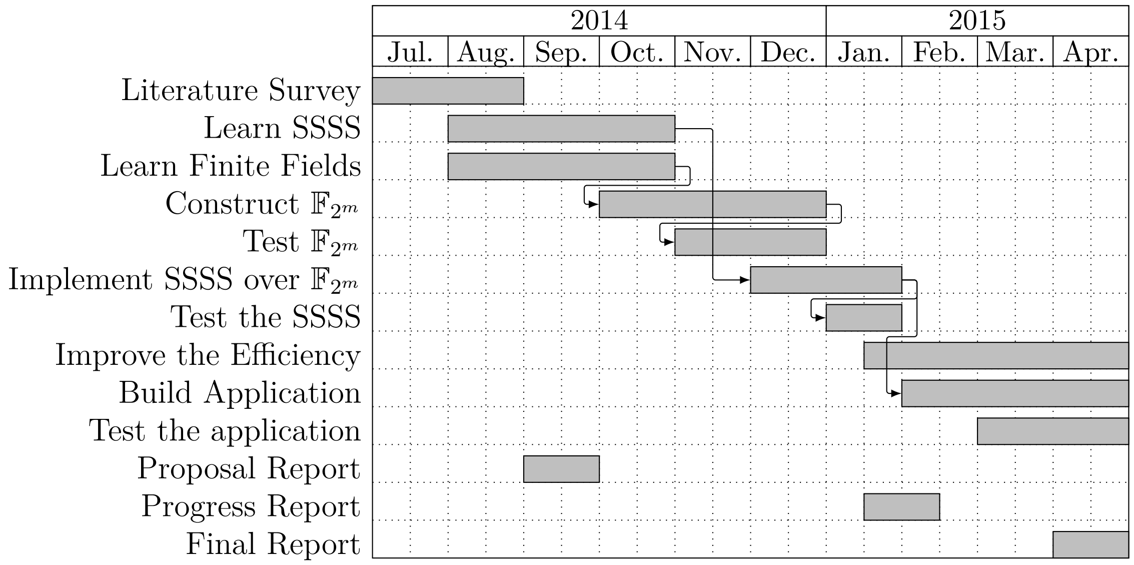
6.2 Further Work

7 Project Progress and Planning

7.1 Distribution of Work

Since this project is one-person project, only myself will do all the tasks.

7.2 GANTT Chart



Note:
"SSSS" stands for "Shamir's Secret Sharing Scheme"

8 References

[1] Sharing a secret on the web. http://www.dis.uniroma1.it/˜damore/sss/, Sep. 2014.

[2] ssss: Shamir's secret sharing scheme. http://point-at-infinity.org/ssss/, Sep. 2014.

[3] Ubuntu manpage: gfshare - explanation of shamir's secret sharing in gf(2**8). http://manpages.ubuntu.com/manpages/hardy/man7/gfshare.7.html, Sep. 2014.

[4] George R. Blakley. Safeguarding Cryptographic Keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, volume 48, pages 313–317, June 1979.

[5] Darrel Hankerson, Julio López Hernandez, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. In *Cryptographic Hardware and Embedded Systems—CHES 2000*, pages 1–24. Springer, 2000.

[6] Robert J McEliece. *Finite fields for computer scientists and engineers*, volume 23. Kluwer Academic Publishers Boston, 1987.

[7] Gadiel Seroussi. *Table of low-weight binary irreducible polynomials.* Hewlett-Packard Laboratories, 1998.

[8] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

9 Appendix

9.1 Minutes

9.1.1 Minutes for the 1$^{\mathbf{st}}$ Project Meeting

Date:	May 30, 2014
Time:	16:00
Place:	LG7 Milano Restaurant
Present:	RUAN Keda, Prof. Cunsheng Ding
Recorder:	RUAN Keda

1. Approval of the minutes
This was the first meeting with my advisor, and there were no minutes to approve.

2. Report on progress
This was the first meeting with my advisor, and there were no progress to report.

3. Discussion items
3.1 Prof. Cunsheng Ding gave me the very basic instructions to follow in order to finish the project.
3.2 Prof. Cunsheng Ding asked me to read the book of "Finite Fields for Computer Scientists and Engineers" written by Robert J. McEliece in order to get familiar with the basis of finite fields.
3.3 Prof. Cunsheng Ding explain the basic scenario of the application of Shamir's Secret Sharing Scheme.

4. Goals for the coming time
Start the project by following the steps given by Prof. Cunsheng Ding: get familiar with Shamir's Secret Sharing Scheme, learn basis of finite fields, and implement the algorithm.

9.1.2 Minutes for the 2nd Project Meeting

Date:	November 16, 2014
Time:	15:00
Place:	Lo Ka Chung University Center
Present:	RUAN Keda, Prof. Cunsheng Ding
Recorder:	RUAN Keda

1. Approval of the minutes
2. Report on progress
3. Discussion items
4. Goals for the coming time

9.1.3 Minutes for the 3rd Project Meeting

Date:	December 18, 2014
Time:	10:45
Place:	LG1 Restaurant
Present:	RUAN Keda, Prof. Cunsheng Ding
Recorder:	RUAN Keda

1. Approval of the minutes
2. Report on progress
3. Discussion items
4. Goals for the coming time

9.1.4 Minutes for the 4th Project Meeting

Date:	March 29, 2015
Time:	10:30
Place:	Lo Ka Chung University Center
Present:	RUAN Keda, Prof. Cunsheng Ding
Recorder:	RUAN Keda

1. Approval of the minutes
2. Report on progress
3. Discussion items
4. Goals for the coming time

9.2 Table of Irreducible Polynomials

Table 1: Irreducible Polynomials for $\mathbb{F}_2^m = GF(2^m), 8 \leq m \leq 64$

8	**9**	**10**	**11**
$x^8 + x^4 + x^3 + x^1 + 1$	$x^9 + x^1 + 1$	$x^{10} + x^3 + 1$	$x^{11} + x^2 + 1$
12	**13**	**14**	**15**
$x^{12} + x^3 + 1$	$x^{13} + x^4 + x^3 + x^1 + 1$	$x^{14} + x^5 + 1$	$x^{15} + x^1 + 1$
16	**17**	**18**	**19**
$x^{16} + x^5 + x^3 + x^1 + 1$	$x^{17} + x^3 + 1$	$x^{18} + x^3 + 1$	$x^{19} + x^5 + x^2 + x^1 + 1$
20	**21**	**22**	**23**
$x^{20} + x^3 + 1$	$x^{21} + x^2 + 1$	$x^{22} + x^1 + 1$	$x^{23} + x^5 + 1$
24	**25**	**26**	**27**
$x^{24} + x^4 + x^3 + x^1 + 1$	$x^{25} + x^3 + 1$	$x^{26} + x^4 + x^3 + x^1 + 1$	$x^{27} + x^5 + x^2 + x^1 + 1$
28	**29**	**30**	**31**
$x^{28} + x^1 + 1$	$x^{29} + x^2 + 1$	$x^{30} + x^1 + 1$	$x^{31} + x^3 + 1$
32	**33**	**34**	**35**
$x^{32} + x^7 + x^3 + x^2 + 1$	$x^{33} + x^{10} + 1$	$x^{34} + x^7 + 1$	$x^{35} + x^2 + 1$
36	**37**	**38**	**39**
$x^{36} + x^9 + 1$	$x^{37} + x^6 + x^4 + x^1 + 1$	$x^{38} + x^6 + x^5 + x^1 + 1$	$x^{39} + x^4 + 1$
40	**41**	**42**	**43**
$x^{40} + x^5 + x^4 + x^3 + 1$	$x^{41} + x^3 + 1$	$x^{42} + x^7 + 1$	$x^{43} + x^6 + x^4 + x^3 + 1$
44	**45**	**46**	**47**
$x^{44} + x^5 + 1$	$x^{45} + x^4 + x^3 + x^1 + 1$	$x^{46} + x^1 + 1$	$x^{47} + x^5 + 1$
48	**49**	**50**	**51**
$x^{48} + x^5 + x^3 + x^2 + 1$	$x^{49} + x^9 + 1$	$x^{50} + x^4 + x^3 + x^2 + 1$	$x^{51} + x^6 + x^3 + x^1 + 1$
52	**53**	**54**	**55**
$x^{52} + x^3 + 1$	$x^{53} + x^6 + x^2 + x^1 + 1$	$x^{54} + x^9 + 1$	$x^{55} + x^7 + 1$
56	**57**	**58**	**59**
$x^{56} + x^7 + x^4 + x^2 + 1$	$x^{57} + x^4 + 1$	$x^{58} + x^{19} + 1$	$x^{59} + x^7 + x^4 + x^2 + 1$
60	**61**	**62**	**63**
$x^{60} + x^1 + 1$	$x^{61} + x^5 + x^2 + x^1 + 1$	$x^{62} + x^{29} + 1$	$x^{63} + x^1 + 1$
64			
$x^{64} + x^4 + x^3 + x + 1$			