SHAMIR'S SECRET SHARING SCHEME USING FLOATING POINT ARITHMETIC

by

Timothy Finamore

A Thesis Submitted to

the Faculty of The Charles E. Schmidt College of Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Florida Atlantic University

Boca Raton, Florida

May 2012

SHAMIR'S SECRET SHARING SCHEME USING FLOATING POINT ARITHMETIC

by

Timothy Finamore

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Thomas Eisenbarth, Department of Mathematical Sciences, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the Charles E. Schmidt College of Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

SUPERVISORY COMMITTEE:

Thomas Eisenbarth, Ph.D
Thesis Advisor

William Kalies, Ph.D

Rainer Steinwandt, Ph.D

Lee Klingler, Ph.D.
Chair, Department of Mathematical Science

Gary W. Perry, Ph.D.
Dean, Charles E. Schmidt College of Science

Barry T. Rosson, Ph.D.
Dean, Graduate College

April 12, 2012
Date

ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks and love to his Mother, Father, Brothers and Sister for their continued support through the writing of this manuscript. The author is grateful to colleagues in the Department of Mathematics for their assistance. Specifically, Eric Golinko who was always there to provide laughter, levity as well as motivation and support. Additionally, for his constant willingness to help, Drake Harmon, will forever be remembered as an integral part of the author's studies. Many professors throughout the department will also forever be appreciated. Namely, Dr Eisenbarth for his patience and determination, Dr. Kalies and Dr. Steinwandt for being a part of the supervisory committee, and Dr. Moosai for just being an all around pleasure to be near.

ABSTRACT

Author: Timothy Finamore

Title: Shamir's Secret Sharing Scheme using Floating Point Arithmetic

Institution: Florida Atlantic University

Thesis Advisor: Dr. Thomas Eisenbarth

Degree: Master of Science

Year: 2012

Implementing Shamir's secret sharing scheme using floating point arithmetic would provide a faster and more efficient secret sharing scheme due to the speed in which GPUs perform floating point arithmetic. However, with the loss of a finite field, properties of a *perfect* secret sharing scheme are not immediately attainable. The goal is to analyze the plausibility of Shamir's secret sharing scheme using floating point arithmetic achieving the properties of a *perfect* secret sharing scheme and propose improvements to attain these properties.

Experiments indicate that property 2 of a*perfect* secret sharing scheme, "Any $k-1$ or fewer participants obtain no information regarding the shared secret", is compromised when Shamir's secret sharing scheme is implemented with floating point arithmetic. These experimental results also provide information regarding possible solutions and adjustments. One of which being, selecting randomly generated points from a smaller

interval in one of the proposed schemes of this thesis. Further experimental results indicate improvement using the scheme outlined. Possible attacks are run to test the desirable properties of the different schemes and reinforce the improvements observed in prior experiments.

SHAMIR'S SECRET SHARING SCHEME USING FLOATING POINT ARITHMETIC

List of Tables....viii

List of Figures....ix

1. Motivation....1

2. Background Information....3

2.1 Sharing a Secret....3

2.2 Floating Point Number Systems....5

2.3 Potential Problems....7

3. Applying Floating Point Arithmetic to Secret Sharing....10

3.1 Forward error....10

4. Experiments....14

4.1 Defining our Arithmetic....14

4.2 Sage....15

4.3 Experimental Results....16

4.3.1 Interval Arithmetic....17

4.3.2 Bit error....22

4.3 Understanding our Experiments....27

5. Possible Attacks....29

5.1 Shared Secret Distribution....29

5.2 K-1 Points....32

6. Conclusion....34

Bibliography....35

LIST OF TABLES

Table 1: Defining arithmetic.....14

Table 2: Interval arithmetic with 2 points and p=3.....19

Table 3: Interval arithmetic with 2 points and p=4.....20

Table 4: Interval arithmetic with 3 points and p=4.....23

Table 5: Interval arithmetic with 3 points and p=20.....24

Table 6: Success-failure rate using scheme 1.....26

Table 7: Success-failure rate using scheme 2.....27

Table 8: Knowledge of k-1 points attack.....33

Table 9: Knowledge of k-1 points attack scheme 2.....33

LIST OF FIGURES

Figure 1: Small example of floating point system.....6

Figure 2: Forward and backward error.....8

Figure 3: g(x) and f(x).....24

Figure 4: Distribution of shared secrets.....31

1. MOTIVATION

Secret sharing schemes are a vital tool in information security. Secret sharing schemes can be used for any situation in which the access to an important resource has to be distributed over several parties. The case of opening bank vaults or launching a nuclear missile are both situations in which a secret sharing scheme can be utilized [5]. Traditionally, Shamir's secret sharing scheme [6] which does computations over a finite field has been the standard. However, with recently developed GPUs, floating point arithmetic can be done faster than integer arithmetic [2]. Research involving the use of graphics cards and cryptography has provided significant results. For example, the NVIDIA GTX 295 graphics cards are being used to employ the elliptic curve method at record setting rates [2]. Results such as these provide motivation for getting Shamir's secret sharing scheme to work with floating point arithmetic.

The key difference between Shamir's secret sharing scheme and a secret sharing scheme using floating point arithmetic will be the absence of a finite field. A floating point system is a subset of the real numbers, which is a field, but the real numbers are not finite. Thus, a floating point number system can only represent finitely many real numbers. This is dependent upon the precision that is used. Thus, when a real number cannot be represented exactly as a floating point it must be rounded. Our goal is to implement Shamir's secret sharing scheme using floating point arithmetic, while maintaining the attributes that contribute to the security of Shamir's secret sharing scheme. These attributes will be discussed in Chapter 2 along with some relevant background information. Chapter 3 highlights some important information regarding

the numerical stability of certain algorithms and floating point numbers in general. In this chapter we will also decide on an interpolation method for our secret sharing scheme. Chapter 4 then seeks to understand and address some questions regarding key distribution, rounding errors, and unrepresentable numbers through experiments. After getting a glimpse of some possible problems and techniques for a secret sharing scheme using floating point arithmetic, we then show how this information can be exploited and compromise the integrity of a secret sharing scheme using floating point numbers in Chapter 5.

2 BACKGROUND INFORMATION

Before discussing the application of floating point arithmetic in Shamir's secret sharing scheme, it is useful to discuss some preliminary topics. It should be noted that Shamir's secret sharing scheme is only one of many secret sharing schemes. Blakely's secret sharing scheme which is provably secure is an alternative to Shamir's, but requires larger shares distributed to participants. A trivial secret sharing scheme which utilizes XORing, unlike Shamir's, is not a threshold scheme. As mentioned there are other reliable secret sharing schemes available, but the focus of this thesis will be on Shamir's secret sharing scheme. We will only address this specific scheme in this section. Information regarding floating point arithmetic will be provided as well as properties that we seek to retain in a secret sharing scheme that utilizes floating point arithmetic later in the chapter.

2.1 Sharing a Secret

Eleven people are working on a secret project. They wish to lock up their information so that 7 people or more are required to be present to unlock it, and that less than 7 people obtain no information regarding the key. This is exactly the problem Shamir's secret sharing scheme solves. The scenario above is a $(7, 11)$ *threshold scheme*. In general, we will refer to our data as D. Our goal is to divide D into n pieces $D, \ldots D_n$. Such a scheme is called a (k, n) *threshold scheme* [6]. The scenario above is a $(7, 11)$ *threshold scheme*. The idea is to use a polynomial of degree $k - 1$,

$q(x) = a_0 + a_1 x + \ldots a_{k-1} x^{k-1}$ in which $a_0 = D$ and evaluate

$$D_1 = q(1), \ldots D = q(i), \ldots D_n = q(n).$$

We can then find the coefficients of $q(x)$ through interpolation, and then find $D = a_0$. This solves our problem because having $k-1$ points on the polynomial does not suffice to calculate D. Furthemore, having $k-1$ points does not increase the likelihood of finding D.

This is all done over a finite field. A set of consecutive integers from 1 to $p-1$ modulo a prime p forms a field. In a field every non-zero element has an inverse, which makes interpolation possible. Given D, we pick a prime p that is larger than both D and n. The coefficients for q are chosen uniformly at random from $[0, p)$, and the values $D_1, ..., D_n$ are computed modulo p. For more information please see [6].

Above we have noted some desirable properties for our secret sharing scheme. We will go into a little more detail, so that we are clear on what exactly we will be looking for with our secret sharing scheme. In order to do this we will introduce some notation and define a *perfect* secret sharing scheme. A secret sharing scheme is said to be *perfect* if (1.) knowledge of any k or more pieces makes the shared secret reconstructible; (2.) Any $k-1$ or fewer participants obtain no information regarding the shared secret. To make this more precise, we use the entropy function H. K will denote our secret key, and Γ as the collection of subsets of participants that can reconstruct the secret key. [7] defines a perfect secret sharing scheme by:

(1.) Any qualified subset X can reconstruct the shared secret:

$\forall_{X \in \Gamma} H(K|X) = 0$ and:

(2.) Any unqualified subset has no information regarding the shared secret:

$\forall_{X \notin \Gamma} H(K|X) = H(K)$

The focus now shifts to some preliminary information regarding floating point numbers.

2.2 Floating Point Number Systems

A general understanding of floating point representation is necessary if we are to apply to Shamir's secret sharing scheme. A floating point number system $F \subset \mathbb{R}$ is a subset of the real numbers whose elements have the form [3]:

$y = \pm m \cdot \beta^{e-t}$

The system F is characterized by four integer parameters:

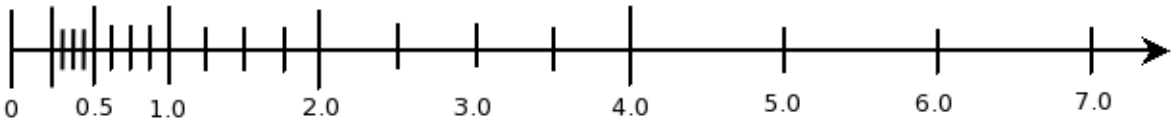
- the *base* β (sometimes called the radix)
- the *precision* t, and
- the *exponent range* $e_{min} \leq e \leq e_{max}$.

The *significand* m is an integer satisfying $0 \leq m \leq \beta^t - 1$. To ensure a unique representation for each nonzero $y \in F$ it is assumed that $m \geq \beta^{t-1}$ if $y \neq 0$, so that the system is *normalized*. It is important to realize floating point numbers are not equally spaced. If $\beta = 2$, $t = 3$, $e_{min} =$ -1 and $e_{max} = 3$ then the nonnegative floating point numbers are:

$$0, 0.25, 0.3125, 0.3750, 0.4375, 0.5, \ldots, 4.0, 5.0, 6.0, 7.0 \tag{1}$$

They can be represented pictorially as follows:

Supplementary Figure 1: Small example of floating point system



Notice that the spacing of the floating point numbers jumps by a factor of 2 at each power of 2. The spacing can be characterized in terms of *machine epsilon*, which is the distance ϵ_M from 1.0 to the next larger floating point number.

LEMMA: The spacing between a normalized floating point number x and an adjacent normalized floating point number is at least $\beta^{-1}\epsilon_M|x|$ and at most $\epsilon_M|x|$ [3] .

Proof. see [3]

In general the IEEE standard uses normalized numbers, but when $e = e_{min}$ - t it will allow for $m \leqslant \beta^{t-1}$. A floating point number of this form is called *denormalized*. This allows us to represent numbers less than $\beta^{e_{min}-1}$. This gives us the luxury of gradual underflow. Without denormalized numbers, anything less than $\beta^{e_{min}-1}$ will be rounded to 0. However, when denormalized numbers are included, we can reduce this error by rounding to the nearest denormalized floating point. We return to example (1.). Notice that we can only represent numbers as small as 0.25. Anything less than 0.25 will be rounded to 0 or 0.25. When we include *subnormal*(denormalized) numbers we now have the numbers:

$$0.0625, 0.125, 0.1875$$

and the complete number system can be represented as:

$$0, 0.0625, 0.125, 0.1875, 0.25, 0.3125, 0.3750, 0.4375, 0.5, \ldots$$

In this thesis we will concentrate only on normalized floating point numbers. From this point forward it will be implicit that when we refer to floating point numbers, we are referring to normalized floating point numbers. Moving along, in looking at distances between normalized floating points we find ourselves naturally propelled into discovering a bound on our shared secrets after interpolation and just how many normalized floating point numbers fall within that bound. We now go through some more preliminaries to give us some more insight to this question.

We let y' be the approximation of $y = f(x)$. Using example 1 and $f(x) = 3x$ we seek to represent the number $y = 0.5625$. In this case $x = .1875$ the best we can do is represent it with $y' = 0.625$. We want to know for what Δx do we have $y' = f(x+\Delta x)$? In general, there may be many such Δx, so we should ask for the smallest one. In our example, as $f(.1875+0.020833) = f(0.208333) = 0.625$. The value of $|\Delta x|$, possibly divided by $|x|$, is called the *backward error*. The $absolute(|y-y'|)$ and $relative(|y-y'|/|y|)$ errors of y' are called *forward errors*, to distinguish them from the backward error. Figure 2 should serve as a useful example.

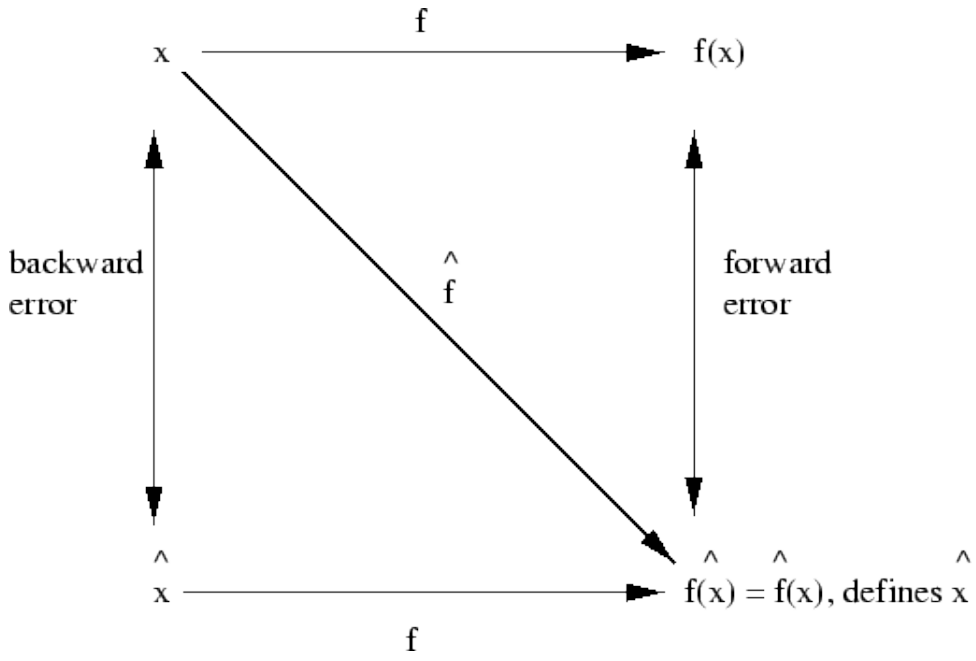
Again using our example, we get a backward error of $(0.0208333/0.1875) = .11111109$ and a relative forward error of $(|0.5625 - 0.625|/0.5625) = 0.111111$.

2.3 Potential Problems

Before we can identify what problems we may encounter in our secret sharing scheme we will point out the properties we seek. As discussed in section 2.1 some of these properties will parallel a *perfect* secret sharing scheme. The following summarizes properties we wish to attain:

1. knowledge of any k or more pieces makes the shared secret reconstructible

Supplementary Figure 2: Forward and backward error



2. Any $k-1$ or fewer participants obtain no information regarding the shared secret

The second property presents problems because at times having knowledge of $k-1$ pieces or fewer could give an attacker information on what the other pieces may be. This could be done through eliminating possible points by checking whether or not they all interpolate to the same shared secret. Additionally, any point that would result in a shared secret larger or smaller than we can represent in our number system could also be elimated as a possibility. The distribution of our shared secrets(y-intercepts) must be examined so that a possible attacker can not use probabilities to his advantage. This would give an attacker information regarding the shared secret with $k-1$ points or less. Hence, we are looking for a uniform distribution with our shared secrets.

During the interpolation process it will be necessary to round at times and this will effect the distribution of shared secrets. Additionally, as mentioned before, without the use of a finite field we are faced with the issue of dealing with values that may be larger or smaller than we can represent in our floating point number system. Returning to

example 1 in chapter 2, suppose we are interpolating with the points $(1, 7)$ and $(2, 6)$. Then after interpolation we see our *y-intercept* is 8.0. This number is not representable in our number system and a decision must be made for dealing with these scenarios.

Another major problem arises in errors that occur through interpolation. Suppose we have a $(3, 5)$ threshold scheme. We generate 3 random points, and use modified lagrange interpolation to find our shared secret. This shared secret may not be exact, because through our interpolation we are introducing new possible errors. The interpolated polynomial is then used to generate the remaining 2 points. Again, some more error will be introduced depending on our floating point system. This all leads to the possibility that different subsets of 3 points may interpolate to different shared secrets.

The situations discussed above jeopardize the second property that we are seeking for our scheme. The next chapter analyzes interpolation methods and error bounds in an effort to maintain these properties.

3 APPLYING FLOATING POINT ARITHMETIC TO SECRET SHARING

After examining desirable properties for our secret sharing scheme and how floating point arithmetic may make it difficult to attain these, we now analyze different interpolation methods.

3.1 Forward Error Bound and Interpolation Method

At this time we will focus on the forward error of the interpolation method. There is useful research regarding the forward stability of various interpolation techniques at our disposal. One paper offers a few different interpolation options [4]. First, the standard Lagrange Interpolation method which has become regarded as being only of theoretical interest. Then the Modified Lagrange formula, and finally the Barycentric formula. For our purposes we found that the Modified Lagrange formula, although not the most efficient, to be the most stable, forward and backward. This is based on the analysis done in [4]. The paper finds Barycentric Lagrange Interpolation to be forward stable, but not backward. Modified Lagrange Interpolation is shown to be both backward and forward stable. This is important because we would like to choose x-coordinates in such a way that our shared secret stays within a given error bound. Thus, Modified Lagranage Interpolation is best suited for our scenario [4]. We now present the Modified Lagrange Interpolation method:

$$p_n(x) = l(x)\sum_{j=0}^{n} \frac{w_j}{x - x_j} f_j,$$

where

$$l(x) = \prod_{j=0}^{n}(x - x_j)$$

and

$$w_j = \frac{1}{\prod_{k \neq j}(x_j - x_k)}$$

We are interested in computing the forward error bound for the Modified Lagrange Interpolation formula. First, we need to discuss the condition number. We are interested in the problem of finding the polynomial $p_n(x)$ of degree at most n that interpolates to the date f_j at the distinct points x_j,j= 0 : n. We consider fixed interpolation points x_j, a fixed evaluation point x, and a varying vector f. We will therefore also denote $p_n(x)$ by $p_f(x)$. Inequalities between vectors are interpreted componentwise.

To aid the interpretation of our error bounds we need to define and evaluate a condition number for p_n. Recall that the Lagrange form of $p_n(x)$ is

$$p_n(x) = \sum_{j=0}^{n} f_j l_j(x), l_j(x) = \frac{\prod_{k=0,k\neq j}^{n}(x - x_k)}{\prod_{k=0,k\neq j}^{n}(x_j - x_k)}.$$

DEFINITION 3.1 The condition number of p_n at x with respect to f is, for $p_n(x) \neq 0$,

$$cond(x, n, f) = \lim_{\epsilon \to 0} \left\{ \left| \frac{p_f(x) - p_{f+\Delta f}(x)}{\epsilon p_f(x)} \right| : |\Delta f| \leq \epsilon |f| \right\}$$

In the notation $cond(x, n, f)$ the term n indicates the dependence of cond on the points x_j.

LEMMA 3.1

$$cond(x, n, f) = \frac{\sum_{j=0}^{n} |l_j(x) f_j|}{|p_n(x)|} \geq 1$$

and for any Δf with $|\triangle f| \leq \epsilon |f|$ we have

$$\frac{|p_f(x) - p_{f+\Delta f}(x)|}{|p_f(x)|} \leq cond(x, n, f)\epsilon.$$

Proof. see [3]

We have one more remaining piece to the forward error bound puzzle. For error analysis we use the standard model of floating point arithmetic [3] :

$$fl(xopy) = (xopy)(1 + \delta), |\delta| \leq u, op = +, -, *, /,$$

where u is the unit roundoff. Our bounds are expressed in terms of the constant [4] :

$$\gamma_k = \frac{ku}{1 - ku},$$

where $ku < 1$ is assumed. We will not spend time on the particuliars of γ_k , but it becomes very useful in the calculation of the bound we sought. Namely, The forward error bound for Modified Lagrange formula:

$$\frac{|p_n(x) - p'(x)|}{|p_n(x)|} \leq \gamma_{5n+5} cond(x, n, f).$$

Note that this is represented in terms of the relative error. However, we can simply multiply both sides of the equation by $|p_n(x)|$ to obtain the maximum interval for our shared secrets. We now use our knowledge of spacing between normalized floating points (Lemma 1.1), to determine how many points lie in a given interval. For now we use the maximum distance of $\epsilon |x_0|$. Suppose we compute our bound and find it is of length y. We let x_0 be the smallest representable floating that falls in the interval

of length y. Now the distance to the next normalized floating point is $\epsilon|x_0|$, and then our next normalized floating point is $x_1 = (x_0 + \epsilon|x_0|)$. To find x_2, we use $x_2 = x_1 + \epsilon|x_1| = (x_0 + \epsilon|x_0|) + \epsilon|(x_0 + \epsilon|x_0|)| = (1+\epsilon)^2|x_0|$. Similiarly, we find that $x_3 = (1+\epsilon)^3|x_0|$ and $x_n = (1+\epsilon)^n|x_0|$. Using this we can easily solve for n, and obtain an upper bound for the number of normalized floating points fall in our interval. Given an error bound z, let $\beta = 2z$ be an interval. Then the following holds as an upper bound for the number of normalized floating points n :

$$n = \frac{ln(\frac{\beta}{x_0})}{ln(1+\epsilon)}$$

where x_0 is the smallest normalized floating point in β.

Implenting our result with randomly generated numbers, experimental results suggest that the number of points n in our interval β, remains fairly constant when $cond(x, n, f) \approx 1$. This is ideal because as we have mentioned before we would like as close to a uniform distribution as possible. So our next task is to see if there are "smart" choices for the x values of our points that will yield a condition number close to 1.

4 EXPERIMENTS

In Chapter 2 we discussed the properties we seek for our secret sharing scheme and how the characteristics of floating point arithmetic could make it difficult to achieve these properties. In this chapter experiments were run to seek solutions to these problems as well as gain some information regarding their plausability, but first we clarify the details of how we will run our experiments.

4.1 Defining Our Arithmetic

For our purposes we found Sage to be well-suited for our experiments. Sage uses the IEEE standard arithmetic for floating point numbers, but we feel it is necessary to point how some critical operations are performed [3]. They can be seen in Table 1.

Table 1: Defining arithmetic

Exception Type	Example	Default Result
Invalid Operation	$0/0$, $0^*\infty$, $\sqrt{-1}$,$(+\infty)+(-\infty)$,∞/∞	NaN
Overflow		$\pm\infty$
Divide by zero	Finite nonzero$/0$	$\pm\infty$
Underflow		Subnormal numbers
Inexact	whenever $fl(x \text{ op } y) \neq x$ op y	Correctly rounded result

It is worth pointing out that with sage some things such as, $NaN - NaN$ are defined. This situation arose during interpolation of certain points in our experiments. What ends up happening is, at times we run into $NaN - NaN = 0$. This is fine because we usually get cancellation where we wanted it anyway. On the other hand, when we

encounter the situation, NaN op k, where k is a floating point number, Sage simply stores it as NaN op k. Now we are forced to make a decision. We decide to take any expression involving the term NaN, and group them together. A clear example of this will appear in the Experimental Results section. You will notice one of our y-intercepts is labeled NaN.

4.2 Sage

All of our experiments will be done using Sage, so we present some relevant information regarding Sage's floating point. In Sage (as in MPFR), floating-point numbers of precision p are of the form $sm2^{e-p}$, where $s = \{-1, 1\}$, $2^{p-1} \leq m \leq 2^p$, and $-2^{30} + 1 \leq e \leq 2^{30} - 1$; plus the special values +0, -0, +infinity, -infinity, and NaN (which stands for Not-a-Number). In order to uniformly generate numbers of a certain precision we will individually randomly generate s, m, and e. These will all be integer values with the exception $sm2^{e-p}$ will be a floating point number of precision p. Remember that in Chapter 2 we gave a definition for a floating point number system. Sage's representation of floating point number's is nearly identical with the exception that Sage represents the precision with p as opposed to t. We would also like to be able to specify the exponent range. Rather than always using $-2^{30}+1 \leq e \leq 2^{30}-1$, we will specify an *emin* and *emax* in each experiment. Thus, we will have $emin \leq e \leq emax$ and there will be no discrepancies between the two representations.

In Sage numbers will be stored in binary representation, but Sage will give us a truncated result in Decimal form. For example, the number -0.4453125 is stored as -0.0111001 in Sage. However, depending on the precision Sage will represent -0.4453125 differently in decimal form. Using a precision of six, Sage represents -0.4453125 in decimal form as -0.45. This example gives us a clear picture of how Sage stores floating point numbers. We can see that the binary storage gives us a

much more precise representation than what Sage prints in decimal representation. We must keep this in mind when determining the number of bits we lose through the interpolation process.

4.3 Experimental Results

Now that some of the details of how Sage works and our floating point arithmetic is defined, we are ready to look at some experimental results. We have outlined some desirable properties for our potential secret sharing scheme. The experiments in this section focus on specific properties. The experiments in Section 4.3.1 examine how different subsets of k points interpolate, and in Section 4.3.2 we analyze the number of bits of error that occur using a specified shared secret rounding method. When it comes time to evaluate a potential scheme we will specify in the experiment one of the two following schemes:

Scheme 1: The rounding method used in these experiments simply truncates half of the number of bits of our precision. For example, if we have $p = 9$ we chop off the last $floor(\frac{9}{2}) = 4$ bits of our secret key. In a (k, n) threshold scheme the x-values will be assigned in order from 1 to n. For example, the third point will have an x-value of 3. As for the y-values, the first k will be generated uniformly at random from our floating point number system unless otherwise specified. How they are generated was discussed in Setion 4.2. The remaining $n - k$ y-values will be generated by evaluating the remaining $n - k$ x-values in the interpolated polynomial. $\pm infinity$ will not be considered valid keys in this scheme.

Scheme 2: This scheme is identical to scheme 1 with the exception that our y-values will be selected from a smaller interval. Ordinarily, when we randomly generate our first k y-values we permit the exponent to be between *emin* and *emax*. In this scheme the exponent must be chosen to be between *emin* and 0.

4.3.1 Interval Arithmetic

Throughout interpolation we experience error due to rounding. All points within a given interval will round to a representable floating point number in our system. Referring back to the experiment in Section 4.3.1, there are large gaps between our representable floating numbers. For example, the next representable floating point number after 0.31 is 0.38. When a calculation yields 0.33 it gets rounded to 0.31 because it is closer to 0.31 then 0.38. Thus, 0.33 is in the interval of 0.31. Using the maximum and minimum elements of particuliar intervals we can than predict our worst case rounding scenarios. The maximum element in the interval of 0.31 would be 0.345 and the minimum element would be 0.28. As we will see in this section we can perform operations with the interval $[0.28, 0.345]$ and get an understanding of the worst case rounding scenarios during the interpolation process.

First, we define the basic operations for Interval Arithmetic. Whereas classical arithmetic defines operations on individual numbers, interval arithmetic defines a set of operations on intervals [1]:

$T * S = \{x|$ there is some y in T, and some z in S, such that $x = y \cdot z\}$.

The basic operations of interval arithmetic are, for two intervals $[a, b]$ and $[c, d]$ that are subsets of the real line $(-\infty, \infty)$,

$[a, b]+[c, d] =[\min(a+c, a+d, b+c, b+d), \max(a+c, a+d, b+c, b+d)] = [a+c, b+d]$

$[a, b]-[c, d] = [\min(a-c, a-d, b-c, b-d), \max(a-c, a-d, b-c, b-d)] = [a-d, b-c]$

$[a, b]\times[c, d] = [\min(a\times c, a\times d, b\times c, b\times d), \max(a\times c, a\times d, b\times c, b\times d)]$

$[a, b]\div[c, d] = [\min(a\div c, a\div d, b\div c, b\div d), \max(a\div c, a\div d, b\div c, b\div d)]$ when 0 is not in $[c, d]$.

Interval Arithmetic with 2 points

We return to our ongoing example using the set of floating point numbers from Section 2.2. In this case we are using $\beta = 2$, $p = 3$, e_{min} = -1 and $e_{max} = 3$. We select our y-values from the floating point number system below:

{-7.0, -6.0, -5.0, -4.0, -3.5 , -3.0, -2.5, -2.0, -1.8, -1.5, -1.3, -1.0, -.88, -.75, -.62, -.5, .-.44, -.38, -.31, -.25, 0, .25, .31, .38, .44, .5, .62, .75, .88, 1.0, 1.3, 1.5, 1.8, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 6.0, 7.0}

This time we will use interval arithmetic to get a range of y_2 values that will interpolate to the proper shared secret. What we do is first interpolate using (x_0, y_0) and (x_1, y_1). We get our shared secret k. However, k gets rounded to the nearest representable floating point. Like earlier, we will refer to this as *newk*. Now, using the interval of points that round to *newk*, say $[\delta, \delta_1]$, we have the point $(0, [\delta, \delta_1])$. Taking $(0, [\delta, \delta_1])$ and (x_0, y_0) we get a new equation for a line. If we write the equation for this line in slope-intercept form, it will contain an interval as part of the slope and also the y-intercept. Similiarly, using $(0, [\delta, \delta_1])$ and (x_1, y_1) we get another equation for a line. We are interested in the possible values for y_2, given $x_2 = 3$ in each scenario. Of course, in each scenario y_2 will be a range. Through our use of interval arithmetic, this range will include the possible points for y_2 that can be used to properly interpolate to *newk*. Since (x_0, y_0) and (x_1, y_1) are theoretically supposed to be on the same line we want to find y_2 values that are common to both of our ranges. Table 2 provides a summary of our findings using every possible combination of values for y_0 and y_1. We list the frequencies of the number of points in common of the two ranges. "*kinfinity*" indicates that our shared secret was either *+infinity* or *-infinity*. On the other hand, "*yinfinity*" indicates that either of the two ranges for y_2 contained all values that were either *+infinity* or *-infinity*. Table 2 lists frequencies of the number of points in common.

Table 2: Interval arithmetic with 2 points and p=3

Number of common points	Frequency	Percentage of Total(1681)	Percentage -"kinfinity" and "yinfinity"	Percentage excluding "yinfinity"
"yinfinity"	182	10.8%	NA	NA
"kinfinity"	398	23.7%	NA	26.6%
0	223	13.3%	20.3%	14.9%
1	305	18.1%	27.7%	20.3%
2	91	5.4%	8.3%	6%
3	157	9.3%	14.3%	10.5%
4	59	3.5%	5.4%	3.9%
5	99	5.9%	9%	6.6%
6	17	1%	1.5%	1.1%
7	46	2.7%	4.2%	3%
8	10	0.6%	0.9%	0.7%
9	6	0.4%	0.5%	0.4%
10	22	1.3%	2%	1.5%
11	6	0.4%	0.5%	0.44%
12	2	0.12%	0.18%	0.1%
13	10	0.6%	0.9%	0.7%
14	12	0.7%	1%	0.8%
17	12	0.7%	1%	0.8%
18	6	0.4%	0.5%	0.4%
19	6	0.4%	0.5%	0.4%
25	8	0.5%	0.72%	0.5%
26	4	0.2%	0.4%	0.25%

If the intervals have any points in common it is possible that both subsets of 2 points will interpolate to the same shared secret. However, if the intervals have no points in common there is no chance that the subsets of points will interpolate to the same key. Table 2 tells us that 47.8% of the time we will not be able to interpolate to the same key. This number results from the frequencies of *kinfinity*, *yinfinity*, and "0". Although the other 52.2% have atleast one point in common, we see a wide variation in the number of points in common.

We now provide similiar information using our interval arithmetic with increased

precision. we are using $\beta = 2$, $p = 4^*$, e_{min} = -1 and $e_{max} = 3$. Again, a $(2, 3)$ threshold scheme will be used, but it is important to realize that because we increased p, we now have many more representable floating point numbers. In fact we now have 79 representable normalized floating point numbers. Table 3 is a list of frequencies of the number of points in common.

Table 3: Interval arithmetic with 2 points and p=4

Number of common points	Frequency	Percentage of Total(1681)	Percentage -"kinfinity" and "yinfinity"	Percentage excluding "yinfinity"
"yinfinity"	845	12.9%	NA	NA
"kinfinity"	1576	24%	NA	27.6%
0	762	11.6%	18.4%	13.3%
1	1165	17.8%	28.14%	20.38%
2	440	6.7%	10.6%	7.7%
3	454	6.9%	9.9%	7.9%
4	278	4.23%	6.7%	4.9%
5	281	4.3%	6.8%	4.9%
6	121	1.8%	2.9%	2.1%
7	161	2.45%	3.8%	2.81%
8	52	0.8%	1.3%	0.9%
9	110	1.7%	2.65%	1.9%
10	20	0.3%	0.48%	.34%
11	64	0.9%	1.5%	1.1%
12	14	0.21%	0.3%	0.24%
13	58	0.9%	1.4%	1%
15	12	0.18%	.28%	0.21%
16	32	.49%	.8%	.6%
17	4	0.07%	.09%	0.06%
18	18	0.06%	0.4%	0.31%
19	42	0.6%	1.0%	0.7%
23	2	.03%	.04%	.03%
25	12	0.2%	0.3%	0.2%
26	18	0.24%	0.44%	0.31%
27	2	.03%	.04%	.03%
31	8	0.12%	0.2%	0.14%
33	4	0.06%	.09%	0.06%
35	6	0.09%	.17%	.1%

Much like the results from Table 2 where a lower precision was used, we again get a large percentage of situations where there is no chance of rounding to the same shared secret. 48.5% of the time the number of common points were "0", *kinfinity*, or *yinfinity*. Table 3 suggests that an increased precision does not necessarily lead to more intervals with common points .

Interval Arithmetic with 3 point interpolation

Returning to our Interval Arithmetic, we continue with some experiments using a degree two polynomial and interpolating with three points. We will use interval arithmetic to get a range of y_4 values that will interpolate to the proper shared secret. What we do is first interpolate using (x_0, y_0), (x_1, y_1), and (x_2, y_2). We get our shared secret k. However, k gets rounded to the nearest representable floating point. Like earlier we will refer to this as *newk*. Now, using the interval of points that round to *newk*, say $[\delta, \delta_1]$, we have the point $(0, [\delta, \delta_1])$. Taking $(0, [\delta, \delta_1])$,(x_0, y_0), and (x_1, y_1) we get a new equation for a polynomial. We are interested in the possible values for y_3, given $x_3 = 4$ in each scenario. Of course, in each scenario y_3 will be a range. Through our use of interval arithmetic, this range will include the possible points for y_3 that can be used to properly interpolate to *newk*. Since (x_0, y_0),(x_1, y_1), and (x_2, y_2) are theoretically supposed to be on the same polynomial we want to find y_3 values that are common to both of our ranges. Table 4 provides a summary of our findings using randomly generated numbers of certain precisions for y_0, y_1, and y_2. We list the frequencies of the number of points in common of the three ranges. "*kinfinity*" indicates that our secret key was either *+infinity* or *-infinity*. On the other hand, "*yinfinity*" indicates that any of the three ranges for y_3 contained all values that were either *+infinity* or *-infinity*. In Table 4 you will see that we run several experiments using different values for p, e_{min}, and e_{max}. We built our algorithm to allow us to

change the precision p, the $emin$, and the $emax$ values. In changing these values we were hoping to get more interval intersections containing only one point. However, there did not seem to be any noticable difference so only two small examples will be displayed here. First, we look at $p = 4$, $emin = -4$, and $emax = 4$.

With an increased number of points used in interpolation we are bound to find less intervals with common points. Table 4 supports this idea and we see that 90.04% of the time there is no chance of the three subsets of points interpolating to the same shared secret. Next, we use $p = 20$, $emin = -20$, and $emax = 20$ in Table 5. The results are different from Table 4, but there is still few intervals with points in common.

As one would expect the frequency of "*kinfinity*" intervals decreases as the precision is raised. Unfortunately, we do not see any real change in the number of times the three intervals had at least one point in common. 82.78% of the time the intervals had no points in common. This is an improvement from our three point interpolation experiment that used a lower precision, but seeing how the highest precision the IEEE standard uses is 53, it does not appear to be a significant enough improvement.

4.3.2 Bit Error

Another area of concern for our secret sharing scheme is how large of a difference we get in our shared secrets when we interpolate with different subsets of points. For example, suppose we are working with a $(3, 5)$ threshold scheme with precision 6. We create three random points, say $(1, -13.7)$, $(2, -10.4)$, and $(3, -5.1)$. We interpolate and get the polynomial $g(x) = x^2 + .3x - 15$. We then use $g(x)$ to generate two more points $(4, g(4))$ and $(5, g(5))$. If we now interpolate with $(3, -5.1)$, $(4, g(4))$, and $(5, g(5))$ we get a slightly different polynomial, $h(x) = x^2 + .5x - 16$. The reason for this is that our floating point is being represented with precision 6, thus the slight error. This can be seen in figure 3.

Table 4: Interval arithmetic with 3 points with p=4

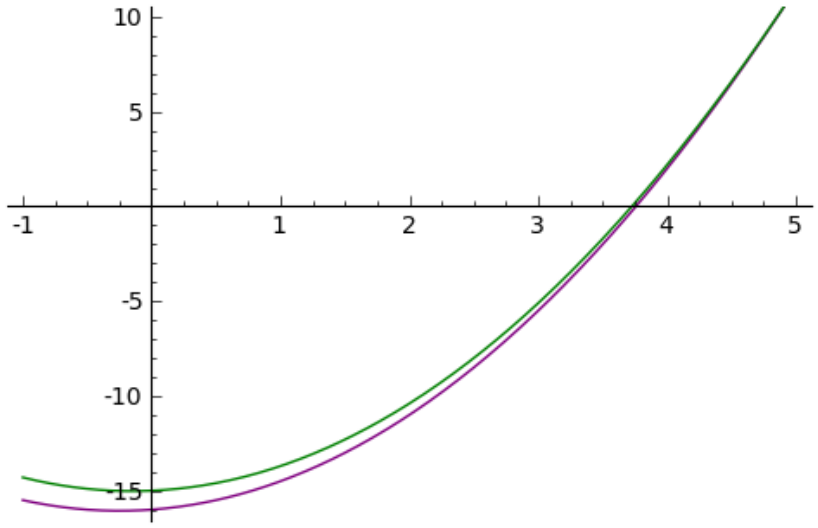
Number of common points	Frequency	Percentage of Total(1681)	Percentage -"kinfinity" and "yinfinity"	Percentage excluding "yinfinity"
"yinfinity"	362	12.07%	NA	NA
"kinfinity"	1583	52.77%	NA	27.6%
0	756	25.2%	71.66%	28.66%
1	40	1.33%	3.79%	1.52%
2	36	1.2%	3.41%	1.36%
3	43	1.43%	4.08%	1.63%
4	34	1.13%	3.22%	1.29%
5	35	0.83%	2.37%	1.02%
6	34	1.13%	3.22%	0.61%
7	27	0.90%	2.56%	0.53%
8	16	0.53%	1.52%	0.61%
9	14	0.47%	1.33%	0.53%
10	11	0.37%	1.04%	0.42%
11	2	0.07%	0.19%	0.08%
12	7	0.23%	0.66%	0.27%
13	1	0.03%	0.09%	0.04%
14	1	0.03%	0.09%	0.04%
15	1	0.03%	0.09%	0.04%
16	1	0.03%	0.09%	0.04%
19	1	0.03%	0.09%	0.04%
20	1	0.03%	0.09%	0.04%
30	1	0.03%	0.09%	0.04%
45	1	0.03%	0.09%	0.04%
47	1	0.03%	0.09%	0.04%
74	3	0.03%	0.28%	0.11%
90	1	0.03%	0.09%	0.04%
93	1	0.03%	0.09%	0.04%
100	1	0.03%	0.09%	0.04%
101	1	0.03%	0.09%	0.04%
107	1	0.03%	0.09%	0.04%

In a (k, n) threshold scheme, we want all subsets of k points to interpolate to the same shared secret. As we see in figure 3, this is not always the case. Naturally, we wish

Table 5: Interval arithmetic with 3 points and p=20

Number of common points	Frequency	Percentage of Total(1681)	Percentage -"kinfinity" and "yinfinity"	Percentage excluding "yinfinity"
"yinfinity"	217	7.23%	NA	NA
"kinfinity"	430	14.33%	NA	15.45%
0	1873	61.23%	78.07%	66.01%
1	48	1.60%	2.04%	1.72%
2	104	3.47%	4.42%	3.74%
3	53	1.77%	2.25%	1.90%
4	36	1.20%	1.53%	1.29%
5	3	0.10%	0.13%	0.11%
6	4	0.13%	0.17%	0.14%
7	262	8.73%	11.13%	9.41%
56	1	0.03%	0.04%	0.04%
14	4	0.13%	0.17%	0.14%
19	1	0.03%	0.04%	0.04%

Supplementary Figure 3: g(x) and f(x)



to know by how many bits do our shared secrets differ. The following experiments use a variety of values for our precision p, *emin*, *emax*, k, and n. ((k, n) threshold scheme). We run the scheme 100 times and record the average number of bit errors, the standard deviation, and the largest error we encounter. We notice a significant increase in error

for larger values of k and n, and very little change in error for increases in p, *emin*, and *emax*. However, this is slightly misleading. The number of incorrect bits is almost identical, but the postition of these bits is much different. For example, using a low precision we may lose 2 bits in the three positions to the right of the decimal point. Then using a higher precision we lose 2 bits about 15 positions from the decimal point. In each case we lose 2 bits, but in the latter case the actual size of the error is significantly smaller. With *max* being our largest shared secret, *min* being our smallest shared secret, and p our precision, we calculate our error in bits using the following formula:

$$approximate\ error\ (in\ bits) = \log_2((max - min)/|max|) + p$$

It is also important to point out that at times we generate y-values and shared secrets that are larger or smaller floating point numbers than we can represent in our system. In this case we cannot calulate an error so we disregard those shared secrets in our calculations. However, if all of our shared secrets or y-values end up out of range the experiment will not be counted as a success or failure, but "NA". With this cleared up we now have a clearer understanding of what our experiments are telling us. The rounding method used in these experiments simply truncates half of the number of bits of our precision. For example, if we have $p = 9$ we chop off the last $floor(\frac{9}{2}) = 4$ bits of our shared secret. Truncating off more bits may give us cases where our shared secrets are the same, but it will also reduce the number of shared secrets from which to choose from. We count the cases where all subsets of points interpolate to the same shared secret as a "success" and when this does not occur we count it as a "failure". Table 6 uses randomly generated y-values from the entire set of our floating point number system as specified in scheme 1. This is clarified because this will not be the case in later experiments.

Table 6: Success-failure rate using scheme 1

p	*emin*	*emax*	k	n	Mean	Standard Dev	Max Error	NA	Success	Failure
4	-1	3	3	4	0.71	1.66	6.0	44	31	25
6	-3	6	3	4	1.45	2.45	8.8	10	64	26
10	-7	10	3	4	2.04	3.4	13	6	66	28
20	-12	14	3	4	1.69	5.6	23.00	2	89	9
4	-1	3	3	7	2.24	2.26	7.00	43	25	32
10	-7	10	3	7	8.24	4.03	14.81	4	13	83
20	-12	14	3	7	6.59	9.15	24.9	7	58	35
20	-12	14	3	11	14.26	11.8	29.67	7	23	70
20	-12	14	5	6	.304	2.56	22.26	27	72	1
20	-12	14	5	11	6.13	7.6	21.67	17	49	34
20	-12	14	7	9	1.16	3.29	10.99	55	40	5
25	-127	128	7	9	8.6	12.71	32.7	8	61	31

Table 6 does show us that as the precision and exponent are increased our success rate improves as well as the average number of bits of error. It also shows us that the more points that are used for interpolation the more error there is. Another trend that we notice is that the larger the difference between k and n the more error there appears to be. Although there are some cases which exhibit a success rate of close to 90%, there is much inconsistency with this rounding method, especially as $n - k$ increases and k.

These next experiments are of a similiar nature with the exception that in an effort to reduce the number of "NA" cases we generate our numbers from a smaller interval. Ordinarily, when we randomly generate our first k y-values we permit the exponent to be between *emin* and *emax*. Now, we will permit our first k y-values to only be randomly generated using an exponent between *emin* and 0. We specified this earlier as scheme 2. We will be reducing our possible randomly generated y-values by half. This should help reduce the other k through n y-values that are generated from our interpolated polynomial, from being larger or smaller floating point numbers than what

we can represent in our system. Ultimately, this should lead to a decrease in the number of "NA" cases. Table 7 supports this idea.

Table 7: Success-failure rate using scheme 2

p	$emin$	$emax$	k	n	Mean	Standard Dev	Max Error	NA	Success	Failure
4	-1	3	3	4	2.17	1.61	6.8	0	20	80
6	-3	6	3	4	1.72	2.35	12.00	0	60	40
10	-7	10	3	4	1.59	3.26	13.05	0	78	22
20	-12	14	3	4	0.311	1.77	10.58	0	97	3
4	-1	3	3	7	5.48	1.07	3.78	0	0	100
10	-7	10	3	7	8.317	2.18	14.8	0	1	99
20	-12	14	3	7	3.03	4.75	11.26	0	71	29
20	-12	14	3	11	11.19	2.77	14.75	0	5	95
20	-12	14	5	6	.209	1.47	10.89	0	98	2
20	-12	14	7	9	3.007	4.83	12.42	0	72	28
25	-127	128	7	9	5.06	10.38	32.735	0	79	21

By reducing the interval from which we generate our y-values the table above shows a significant increase in the number of *success* scenarios. The average error in bits is smaller and we have elimated all of our "NA" cases.

4.3 Understanding Our Experiments

In Chapter 4 we ran several experiments all with a similiar goal in mind, we wanted to find a way to get an idea of problems we would encounter with a secret sharing scheme. In 4.3.1 we utilized interval arithmetic to better understand how often it would be possible to interpolate with different subsets of points to get the same shared secret. First, we looked at a $(2, 3)$ threshold scheme in Section 4.3.1. We ran experiments with two different precisions. The first experiment shown in Table 2 gave us around 47.8% of our intervals had 0 points in common. Next, when we raised the precision in Table 3, we got around 48.5% of our intervals had 0 points in common. It was clear

in this case that raising the precision did not help us. Moving on to a $(3, 4)$ threshold scheme, it was expected that more intervals would have 0 points in common. This was anticipated mainly because of the fact that we were looking at the intersection of three intervals as opposed to two. Table 4 showed that over 90% of the time the intervals had no points in common. Then when we increased the precision, we saw in Table 5 that our cases of "*yinfinity*" and "*kinfinity*" decreased, but over 60% of the intervals had 0 points in common.

The experiments involving the interval arithmetic served as motivation for determining the numbers of bits we were losing through the interpolation process. It is expected that interpolating with different subsets of points will give us a slight difference in our shared secrets, but we would like to know how much of a difference. In calculating the relative error we developed the formula in Section 4.3.2 to tell us how many bits it was that we were actually losing. Through running several experiments as seen in Table 6 and Table 7, we were able to see on average how many bits we were losing based on different parameters. In Table 6 we randomly generated our y-values from our set of all representable floating point numbers. This yielded several situations in which we arrived at y-values or shared secrets that we could not represent in our system. In effort to minimize this occurence we reduced the set of values from which our randomly generated y-values could be selected. We see in Table 7 that it did indeed help. The next section is devoted to utilizing the information we gathered to see how a possible attacker could exploit some of these characteristics.

5. POSSIBLE ATTACKS

Chapter 4 provided us with some useful information regarding our ideal secret sharing properties (as discussed in Chapter 2) and the possibility of them holding true for our secret sharing scheme. After analyzing the experimental data, we see how some of these properties may not be achieved and how an attacker could exploit these drawbacks.

5.1 Shared Secret Distribution

We now bring attention to some small examples to get a clear picture of the distribution of shared secrets. If an attacker is able to detect that the likelihood of interpolating to a particuliar shared secret is higher than another, then it gives him a higher probability of guessing the shared secret. We show that this is the case in our previously used floating point number system from Chapter 2. Using our modified lagrange interpolation on two points x_0 and x_1, we look at a $(2, 3)$ threshold scheme. In this case we are using $\beta = 2$, $p = 3$, e_{min} = -1 and $e_{max} = 3$. We take every possible combination of values for y_0 and y_1, from

{-7.0, -6.0, -5.0, -4.0, -3.5 , -3.0, -2.5, -2.0, -1.8, -1.5, -1.3, -1.0, -.88, -.75, -.62, -.5, .-.44, -.38, -.31, -.25, 0, .25, .31, .38, .44, .5, .62, .75, .88, 1.0, 1.3, 1.5, 1.8, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 6.0, 7.0} *(5.1)

to get an idea of what our distribution of third points will be. Of course we do encounter some problems when certain operations in the interpolation yield values above 7 or

below -7. In these situations we will represent any number larger than 7 as *+infinity*, and any number less than -7 as *-infinity*. This is exactly how we defined our arithmetic to work in table 1. In this example we set $x_0 = 1$ and $x_1 = 2$. Then using Sage, we arrived at the following distribution of third points:

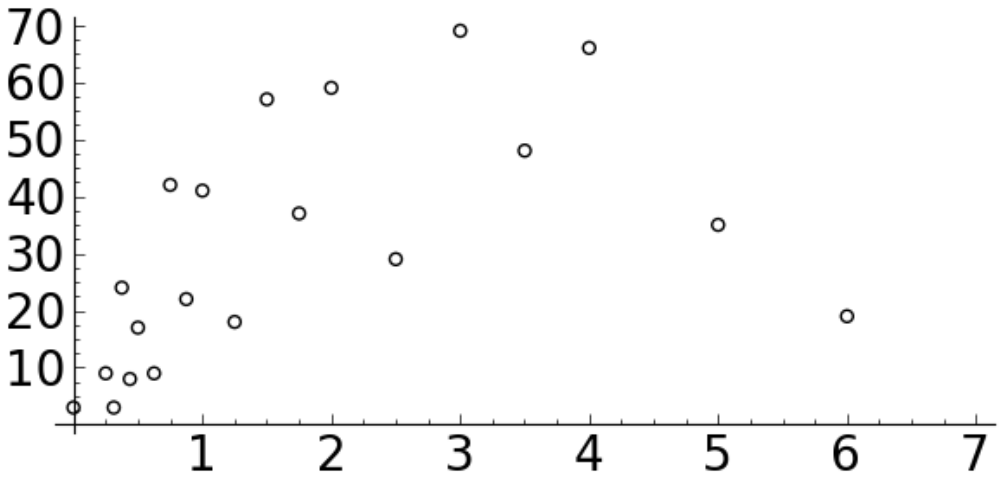
[(-7.0, 35), (-6.0, 66), (-5.0, 48), (-4.0, 69), (-3.5, 29), (-3.0, 59), (-2.5, 37), (-2.0, 57), (-1.8, 18), (-1.5, 41), (-1.2, 22), (-1.0, 42), (-0.88, 9), (-0.75, 17), (-0.62, 8), (-0.50, 24), (-0.44, 3), (-0.38, 9), (-0.31, 3), (-0.25, 19), (0, 34), (0.25, 19), (0.31, 3), (0.38, 9), (0.44, 3), (0.50, 24), (0.62, 8), (0.75, 17), (0.88, 9), (1.0, 42), (1.2, 22), (1.5, 41), (1.8, 18), (2.0, 57), (2.5, 37), (3.0, 59), (3.5, 29), (4.0, 69), (5.0, 48), (6.0, 66), (7.0, 35)(-infinity,199)(+infinity,199)(NaN,19]

Figure 4 is a scatter plot showing the number of times we interpolated to a certain point. Remember, since we are talking all possible combinations for our y-values we will have $41 * 41 = 1681$ possibilities. This is because we have 41 representable floating points as seen above. A uniform distribution would yield a frequency of 41 for each point. Using the graph we can visualize how close we are to achieving that:

Observing the graph and data above we can see that we have not achieved a uniform distribution of third points. This is most interesting when the third point is used as the shared secret. In fact, we had 199 cases where we interpolated to *+infinity* and *-infinity*. Additionally, 19 times we encounterd a situation where our key was NaN (not a number). Additionally, when excluding the *+infinity*, *-infinity*, and NaN cases we calculated the entropy to be 5.02 bits.

Key Distribution using 3 Points

Supplementary Figure 4: Distribution of shared secrets



Next we take a look at our distribution of shared secrets using 3 points. We will assume the same conditions as the previous experiments in this section where $\beta = 2$, $p = 3$, e_{min} = -1 and $e_{max} = 3$. This time we will interpolate with the points $x_0 = 1$, $x_1 = 2$, and $x_2 = 3$. Similiarly, we take every possible combination of values for y_0, y_1, and y_2. We get the following:

[(-7.0, 0), (-6.0, 2759), (-5.0, 1125), (-4.0, 3296), (-3.5, 0), (-3.0, 1845), (-2.5, 683), (-2.0, 1774), (-1.8, 0), (-1.5, 1230), (-1.2, 187), (-1.0, 773), (-0.88, 0), (-0.75, 854), (-0.62, 40), (-0.50, 293), (-0.44, 0), (-0.38, 437), (-0.31, 6), (-0.25, 266), (0, 101), (0.25, 266), (0.31, 6), (0.38, 437), (0.44, 0), (0.50, 293), (0.62, 40), (0.75, 854), (0.88, 0), (1.0, 773), (1.2, 187), (1.5, 1230), (1.8, 0), (2.0, 1774), (2.5, 683), (3.0, 1845), (3.5, 0), (4.0, 3296), (5.0, 1125), (6.0, 2759), (7.0, 0),(-infinity,18288),(+infinity,18288)]

Again, notice that we are far off our desired uniform distribution for our fourth point. This is knowledge that an attacker could use to narrow down possible fourth points and potential shared secrets. Section 5.2 continues in a similiar direction of showing

that probability of attaining every possible shared secret is not the same.

5.2 K-1 Points

Chapter 2 covered the basics of what makes for a secure secret sharing scheme. Recall, that "any unqualified subset has no information regarding the shared secret". In this section our aim is to show that this is not the case with scheme 1 used in Section 4.3.2. Remember that in this experiment the y-values are chosen with an exponent in the range of *emin* and 0. Also, numbers that are larger than we can represent in our system are denoted by $\pm$*infinity* and are not considered valid shared secrets. We exclude $\pm$*infinity* because it would further disrupt the distribution of shared secrets. The rounding method used is the same as Section 4.3.2, we truncate by using half of our original precision. Similiar to Section 5.1 we use a brute force approach to see which shared secrets have little or no possibility of being hit. This experiment will assume that the first $k-1$ points are known and run through every possible k-th point. With this subset of k points we can interpolate to get a polynomial. We check to make sure this polynomial's y-intercept is representable in our system and then check that the other $n-k$ points are representable as well. If we find that the y-intercept or any of the other y- values are not in our system, then the corresponding k-th point can be ruled out as a possibility. This method is used 100 times to see just how many k-th points are even possible. This is valuable information for an attacker to narrow down possible shared secrets. This experiment will use scheme 1 with the conditions stated above. Table 8 displays the number of representable floating points in our system, the threshold scheme, and the number of k-th points that can be ruled out.

Table 8 shows us that in all 3 schemes atleast one third of the kth interpolation points can be excluded. These results do not coincide with the properties of a *perfect* secret sharing scheme. Do to these results an attacker would have a higher probability

Table 8: knowledge of k-1 points attack

Threshold(k,n)	precision	emin	emax	Number of represen-able points	Average number of k-th points that cannot be represented	Average Entropy
(3,4)	6	-3	6	641	192.01	5.197
(4,5)	6	-3	6	641	270.87	4.282
(5,6)	6	-3	6	641	360.78	1.68

of guessing the k-th interpolation point. Thus, another desirable property of our secret sharing scheme is compromised.

Next, the same experiment is run using Scheme 2. Remember, in Scheme 2 our randomly generated y-values are generated from a smaller interval. Table 9 displays the results.

Table 9: knowledge of k-1 points attack scheme 2

Threshold(k,n)	precision	emin	emax	Number of represenable points	Number of k-th points that cannot be represented
(3,4)	6	-3	6	641	0
(4,5)	6	-3	6	641	0
(5,6)	6	-3	6	641	0
(5,6)	10	-6	10	17409	0

Table 9 shows that we are not gaining any information with knowledge of $k-1$ points. Thus, it appears that scheme 2 maintains property 2 of a *perfect* secret sharing scheme when using these parameters.

6. CONCLUSION

Our goal was to implement Shamir's secret sharing scheme using floating point arithmetic. We discussed the properties we sought for our secret sharing scheme using floating point arithmetic. After outlining some potential threats to these properties we ran experiments to gain some clarity on the plausibility of these properties for our scheme. The experiments indicated that the shared secret distribution was not uniform. Additionally, we saw that not every subset of k points interpolated to the same shared secret. In exploring possible attacks on our scheme it was also shown that knowledge of $k-1$ or fewer points could provide information regarding the shared secret. Although scheme 2 did provide us with more successful results, we still found that these three properties were compromised.

More analysis regarding the size of errors through interpolation may lead to solutions to these problems. We were unable to compute a general bound for these errors, but did get information based on experimental data of how they increased. If an error bound is calculated then the points used for interpolation could be selected to minimize error. Additionally, other rounding methods should be explored. A clever rounding method could provide a better distribution of shared secrets, and minimize error through interpolation. There is still research to be done regarding this topic and further research in these areas may still produce a reliable secret sharing scheme using floating point arithmetic. We hope this thesis will provide a basis for continued research in the area.

BIBLIOGRAPHY

[1] R.E. Boche. An operational interval arithmetic. 1963.

[2] Chen-Mou Cheng Tanja Lange Daniel J. Bernstein, Tien-Ren Chen and Bo-Yin Yang. Ecm on graphics cards? *Academia Sinica, 128 Section 2 Academia Road,*.

[3] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM: Society for Industrial and Applied Mathematics; 2nd edition, 2002.

[4] Nicholas J. Higham. The numerical stability of barycentric lagrange interpolation. *IMA Journal of Numerical Analysis*, pages 547–556, 2004.

[5] Sorin Iftene. *Secret Sharing Schemes with Applications in Security Protocols*. PhD thesis, University of Iasi, 2006.

[6] Adi Shamir. How to share a secret. *Communications ACM*, 22:612–613, 1979.

[7] Hung-Min Sun and Shiuh-Pyng Shieh. Constructing perfect secret sharing schemes for general and uniform access structures. *Journal of Information Science And Engineering*, 15:679–689, 1999.