Elliptic Cryptosystem

Elizabeth Dettrey
elizabeth.dettrey@gmail.com

ELLIPTIC CRYPTOSYSTEM

by

Elizabeth Dettrey

Bachelor of Science (B.Sc.)
University of Nevada, Las Vegas
2016

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2018

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

April 10, 2018

This thesis prepared by

Elizabeth Dettrey

entitled

Elliptic Cryptosystem

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Evangelos Yfantis, Ph.D.
Examination Committee Chair

Hal Berghel, Ph.D.
Examination Committee Member

Andreas Stefik, Ph.D.
Examination Committee Member

Sarah Harris, Ph.D.
Graduate College Faculty Representative

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Abstract

The elliptic cryptographic algorithm first presented in a paper by E. F. Dettrey and E. A. Yfantis is examined and explained in this thesis. The algorithm is based on the group operations of a set of points generated from an ellipse of arbitrary radii, and arbitrary center in the case of the generalized version, modulo a large prime. The security of the algorithm depends on the difficulty of solving a discrete logarithm in the groups used by this algorithm. While the elliptic cryptographic algorithm is not the most secure among the discrete logarithm based paradigm of cryptosystems for a given prime, the algorithm can reach relatively high levels of security using a very large prime. It has similar security to RSA, which is still widely used, when the prime in elliptic cryptography is of a similar size to the modulus in RSA. It should be noted that the elliptic cryptographic algorithm is not quantum safe.

Table of Contents

List of Tables

List of Figures

List of Equations and Proofs

Chapter 1

Introduction

The mathematical difficulty of calculating discrete logarithms has long made them an easy choice for providing security in cryptographic algorithms. This thesis will present a new cryptographic algorithm using discrete logarithms for security. The elliptic cryptographic algorithm uses groups created using the points on ellipses of varying radii and center, modulo a prime. While elliptic cryptography shares some similarities with elliptic curve cryptography in the flow of the algorithms and the general premise, the use of ellipses rather than elliptic curves yields different groups. It will be shown that elliptic curve cryptography can produce significantly safer groups than this new alternative, though elliptic cryptography is arguably less complicated. The security of the proposed algorithm will be investigated in the following chapters. The proposed algorithm will be fully defined and proved to work for cryptography. The aim is to present a new cryptographic algorithm, examine its security, and thoroughly explain how it works and how to use it.

Chapter 2

Background

Some of the most familiar cryptosystems today rely on the same mathematical problem to provide security. It is the discrete logarithm problem in finite groups. RSA, elliptic curve cryptography, El Gamal, and even the algorithm to be presented in this paper are based on different groups and discrete logarithms. RSA is based on the residues modulo the product of two large primes [1]. El Gamal uses a cyclic group and a Diffie-Hellman key exchange [2]. Encryption and decryption are as simple as combining a piece of the message with the key, or the inverse key for decryption, using the group operation [2]. Elliptic curve cryptography uses a group containing the set of points on an elliptic curve where the possible point values are usually modulo a prime with many points deemed to all equivalently be identities [3]. The elliptic cryptosystem to be presented in the following chapters uses a group based on points on an ellipse modulo a prime. The recent cyclic cryptographic algorithm uses groups based on the points on a circle modulo a prime [4]. The security of each of these systems is based on discrete logarithms in the groups they are based in.

Other paradigms have been investigated. DNA based cryptography works with the idea of using the sequence of a DNA strand as a one-time pad [5]. As is often the case with one-time pads, there might be difficulty getting the pad to be secretly shared in the first place, but DNA has a random element to it that is necessary for one-time pads [5]. Other work in this area includes stenography and cryptanalysis. Using molecular computing to calculate a discrete logarithm, specifically aiming to attack Diffie-Hellman, was investigated as well [6]. DNA-based cryptography and molecular computing is perhaps an area to watch for cryptographic advancements in the future.

The field of cryptography will need to change radically with the dawn of large scale quantum computers. Shor's algorithm can factor in polynomial time, which will allow RSA systems to be broken in reasonable time [7]. Surprisingly, elliptic curve cryptography is even less safe than RSA under the threat of Shor's algorithm, because the main limiting factor for quantum computers in this case is size [8]. Elliptic curve cryptography uses significantly smaller primes, therefore smaller quantum computers than would be necessary for algorithms using larger primes could defeat it [8]. The algorithm presented in this paper is homomorphic to either F_p^* or the $p+1$ subgroup of F_{p^2}, as will be explained in section 3.4.2. This puts it equal to RSA against Shor's algorithm, therefore it is not quantum safe [7]. Current work on the problem of quantum safe cryptography has yielded a few approaches with lattice based algorithms as the most prominent. A recent work by Oded Regev introduced a new cryptographic algorithm which was proved to be as secure as worst case lattice problems by reducing those problems to the learning with errors problem and basing the algorithm on that problem [9].

For now, there is a wide range of viable cryptographic algorithms available. The field of cryptography can change drastically with new hardware, cryptographic paradigms, and cryptanalytic methods. New developments are continually necessary.

Chapter 3

A New Elliptic Cryptographic Algorithm

3.1 Preface

This chapter contains ”A New Elliptic Cryptographic Algorithm”, a conference paper by E. F. Dettrey and E. A. Yfantis which was presented at IEEE CCWC 2018 on January 10th at the University of Nevada Las Vegas. The paper has been formatted to fit the style of this thesis. The contents of this paper are reprinted here with permission from IEEE (see appendix A) and the second author (see appendix B).

3.2 Abstract

A new cryptographic algorithm is presented in this research paper. This algorithm is based on operations defined on the circumference of an ellipse of arbitrary radii a, b and arbitrary center, modulo an arbitrary prime number large enough that the code cannot be broken within a reasonable amount of time. The operations around the ellipse provide phase angle modulation while modulo arithmetic with respect to the chosen prime number provides integer modulation. The phase modulation used in this algorithm is not the same as the phase modulation transmission system for quantum cryptography which was published in Physics journals. The algorithm has only one zero point where the elliptic curve algorithm has many. Encryption-decryption using this algorithm is straight forward, because integer points not in the group can be used. The algorithm has similar security to RSA.

3.3 Introduction

In this research paper we introduce a new cryptographic algorithm based on operations of points on the circumference of an ellipse of arbitrary radii and on modulo arithmetic over prime numbers. Elliptic curve cryptography invites comparison, as the setup and operations of it and the new algorithm presented here are similar [3, 10]. The algorithm presented in this paper has more straight forward encoding than elliptic curve cryptography, because integer points not in the group can be used. Also, it has only one zero point. The algorithm has a major disadvantage compared to elliptic curve cryptography in that it is less secure for equivalent primes. It has similar security as RSA [1]. The algorithm performs phase modulation with operations around the ellipse, not to be confused with the phase modulation algorithms published in physics literature [11]. Key exchange is possible with the algorithm presented here using a method analogous to Diffie-Hellman [12]. The algorithm introduced in this paper can be used for key management and distribution, cryptography, authentication, and digital signatures.

It's an interconnected world that becomes more connected by the minute, making it easy for information to flow and difficult to keep it private. The internet of things (IoT) has created new applications and new opportunities. At the same time, the security challenges related to IoT are enormous. Some examples include: telemedicine, medical doctors of various specialties providing advice to patients over the internet, and teleradiology, teams of radiologists at various locations reading digital images and videos of patients to provide their professional opinions. Some pacemakers are accessible by the patient's physician over the internet. In cases where the lives of people are in the balance, security is paramount. In addition, HIPAA requirements related to patient privacy are very strict. Satellite navigation and guidance, specifically the global positioning system (GPS) and its orbiting satellites, are used on a daily basis by many commercial users and private citizens. These critical navigation systems are subject to intrusion and jamming, so it's imperative to provide proper security for them. As unmanned aerial vehicles (UAVs) become more established tools in warfare and as major defense contractors focus on designing UAVs with enhanced capabilities, they become increasingly necessary for surveillance, carrying passengers, performing explorations, and carrying weapons. Secure transmission of video from UAVs and secure transmission of instructions to UAVs are a must. In addition, preventing unauthorized people from accessing a UAV is extremely important. Supervisory control and data acquisition (SCADA) is a computer system that is commonly used to monitor and control utility plants such as water utilities, power plants and power distribution. The adversary could shut down utilities, or even cause destruction. The list of examples of the need for private information to remain private goes on. Cryptography is the key to privacy, authentication, digital signing and signatures, secure and private communication, secure channel transmission, and server-host security.

3.4 The Elliptic Cryptographic Algorithm

We start by defining a new set of points on an arbitrary ellipse of radii $a, b \in R$, where R denotes the set of real numbers.

Definition 1. Consider the points on an ellipse of radii $a, b \in R, a, b > 0$, in the x and y axis respectively. We define the set $E(a, b)$, as the set of points

$$(x, y) \in R^2 \ni \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1. \tag{3.1}$$

Now we define the addition on the set of points, of $E(a, b)$, as follows:

Definition 2. Let $X_1 = (x_1, y_2) \in E(a, b), X_2 = (x_2, y_2) \in E(a, b)$, with

$$\theta_1 = arctan(\frac{y_1}{x_1}), \theta_2 = arctan(\frac{y_2}{x_2}), \tag{3.2}$$

then we define the addition $X_3 = X_1 + X_2$ as a point $X_3 = (x_3, y_3) \in E(a, b)$, such that

$$\theta_3 = arctan(\frac{y_3}{x_3}) = \theta_1 + \theta_2. \tag{3.3}$$

Theorem 1. Let $E(a, b)$ be the set given by Definition 1, with the addition operation given by Definition 2, then $E(a, b)$, is an Abelian group.

Proof Let $X_1, X_2 \in E(a, b)$ then $X_3 = X_1 + X_2$ or $X_3 = (x_3, y_3)$ with:

$$x_3 = \text{acos}(\theta_3) = \text{acos}(\theta_1 + \theta_2) = \text{acos}\theta_1\text{cos}\theta_2 \text{ - asin}\theta_1\text{sin}\theta_2$$
$$= \frac{1}{a}(x_1 x_2) - \frac{a}{b^2}(y_1 y_2) \tag{3.4}$$

and

$$y_3 = bsin(\theta_3) = bsin(\theta_1 + \theta_2) = bsin\theta_1 cos\theta_2 + bcos\theta_1 sin\theta_2$$
$$= \frac{1}{a}(y_1 x_2 + x_1 y_2) \tag{3.5}$$

Notice that

$$\frac{x_3^2}{a^2} + \frac{y_3^2}{b^2} = \frac{1}{a^2}\Big[\Big(\frac{x_1^2 x_2^2}{a^2} + \frac{a^2 y_1^2 y_2^2}{b^4} - 2\frac{1}{b^2}x_1 x_2 y_1 y_2\Big) + \Big(\frac{x_2^2 y_1^2 + x_1^2 y_2^2 + 2x_1 x_2 y_1 y_2}{b^2}\Big)\Big]$$

$$= \frac{x_1^2 x_2^2}{a^4} + \frac{y_1^2 y_2^2}{b^4} + \frac{x_2^2 y_1^2}{a^2 b^2} + \frac{x_1^2 y_2^2}{a^2 b^2}$$
$$= \left(\frac{x_1^2}{a^2} + \frac{y_1^2}{b^2}\right)\left(\frac{x_2^2}{a^2} + \frac{y_2^2}{b^2}\right) = 1 \tag{3.6}$$

Therefore the point, $(x_3, y_3) \in E(a,b)$, which is closure under addition. Now consider $X_1, X_2, X_3 \in E(a,b)$ then from 3.4 and 3.5:

$$X_1 + (X_2 + X_3) = X_1 + \frac{1}{a}\Big(x_2x_3 - \frac{a^2}{b^2}y_2y_3, x_2y_3 + x_3y_2\Big)$$
$$= \frac{1}{a}\left[\frac{1}{a}\Big(\Big(x_1x_2 - \frac{a^2}{b^2}y_1y_2\Big)x_3 - \frac{a^2}{b^2}\Big(x_1y_2 + x_2y_1\Big)y_3, \Big(x_1y_2 + x_2y_1\Big)x_3 + \Big(x_1x_2 - \frac{a^2}{b^2}y_1y_2\Big)y_3\Big)\right]$$
$$= \frac{1}{a}\Big(x_1x_2 - \frac{a^2}{b^2}y_1y_2, x_1y_2 + x_2y_1\Big) + X_3$$
$$= (X_1 + X_2) + X_3 \tag{3.7}$$

Hence the associative law holds.

Next consider the element $e = (a, 0)$ then $e \in E(a,b)$ and for any element $X = (x, y) \in E(a,b)$,

$$X + e = \frac{1}{a}(xa, ya) = (x, y) = X. \tag{3.8}$$

Hence $e = (a, 0)$ is the identity element. For every element $X = (x, y) \in E(a,b)$ the element $X' = (x, -y) \in E(a,b)$ is the inverse element of X. Since

$$X + X' = \Big(\frac{x^2}{a} + \frac{a}{b^2}y_2, \frac{1}{a}(xy - yx)\Big) = (a, 0) = e. \tag{3.9}$$

Finally let $X_1 = (x_1, y_1), X_2 = (x_2, y_2) \in E(a,b)$ then

$$X_1 + X_2 = \frac{1}{a}\Big(x_1x_2 - \frac{a^2}{b^2}y_1y_2, y_1x_2 + x_1y_2\Big)$$
$$= \frac{1}{a}\Big(x_2x_1 - \frac{a^2}{b^2}y_2y_1, y_2x_1 + x_2y_1\Big) = X_2 + X_1. \tag{3.10}$$

$E(a,b)$, therefore, with the addition defined by definition 2, is an Abelian group. In order to use $E(a,b)$ for cryptography, the radii a and b have to be integer, and a relatively large prime number P has to be selected. Then the new formulation is:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} \; mod \; P = 1 \; mod \; P \tag{3.11}$$

$$y^2 \; mod \; P = \Big(b^2 - \frac{b^2}{a^2}x^2\Big) \; mod \; P. \tag{3.12}$$

Equations 3.11 and 3.12 along with the addition operation defined above provide phase modulations due to the fact that the operations are around the ellipse, and quadratic modulation due to the y^2 modulo prime number P operation.

3.4.1 Ability of the Algorithm to Encrypt-Decrypt Data

The following theorem is fundamental in establishing that the elliptic cryptographic algorithm is capable of encrypting and decrypting any data, including integer points not in $E_p(a,b)$.

Theorem 2. The unit element $(a,0) \in E_p(a,b)$, under the addition

$$(x_1,y_1)+(x_2,y_2)=\frac{1}{a}\Big(x_1x_2-\frac{a^2}{b^2}y_1y_2,x_1y_2+x_2y_1\Big) \tag{3.13}$$

is also a unit element for every element $(x,y) \in R^2$, where R^2 represents the two dimensional space of real numbers. In addition, if (x,y), are integer numbers with $(x,y) \in E_p(a,b)$, where P is prime positive integer and a and b are integers, then $(x,P-y) \in E_p(a,b)$ and

$$(x,y)+(x,P-Y)=(a,0). \tag{3.14}$$

Therefore $(x,P-y)$ is the additive inverse of (x,y). Also for every $c \in R, (x,y) \in R^2$,

$$(c,0)+(x,y)=\frac{c}{a}(x,y). \tag{3.15}$$

Finally if c,x,y are integer numbers then

$$(c,0)+(x,y) \ mod \ P=\Big(\frac{c}{a}(x,y)\Big) \ mod \ P. \tag{3.16}$$

Proof

First we will prove that $(a,0)+(x,y)=(a,y)$, where $(a,0) \in E_p(a,b)$, and $(x,y) \in R^2$. We have that

$$(a,0)+(x,y)=\frac{1}{a}(xa,ya)=(x,y). \tag{3.17}$$

Hence $(a,0)$ is not only the identity element for the points in $E_p(a,b)$, but all the points in R^2.

Second we will prove that if (x,y) are integer numbers with $(x,y) \in E_p(a,b)$, where P is a prime positive integer and a and b are integers, then $(x,P-y) \in E_p(a,b)$. Since $(x,y) \in E_p(a,b)$, then:

$$y^2 \ mod \ P=\Big(b^2-\frac{b^2}{a^2}x^2\Big) \ mod \ P.$$

Now,

$$(P-y)^2 \ mod \ P=(P^2-2Py+y^2) \ mod \ P=y^2 \ mod \ P=\Big(b^2-\frac{b^2}{a^2}x^2\Big) \ mod \ P$$

hence, $(x,P-y) \in E_p(a,b)$. We have:

$$(x,y)+(x,P-y) \ mod \ P=\frac{1}{a}\Big(x^2-\frac{a^2}{b^2}(Py-y^2),xP-xy+yx\Big) \ mod \ P$$

$$= \frac{1}{a}\left(a^2\Big(\frac{x^2}{a^2}+\frac{y^2}{b^2}\Big), 0\right) \; mod\ P = (a,0) \bmod P. \tag{3.18}$$

Therefore $(x, P-y)$ is the additive inverse of (x,y).

Third we will prove that for every $c \in R, (x,y) \in R^2$,

$$(c,0)+(x,y) = \frac{c}{a}(x,y).$$

We have:

$$(c,0)+(x,y) = \frac{1}{a}(cx,cy) = \frac{c}{a}(x,y). \tag{3.19}$$

Fourth we will prove that if c, x, y are integers then

$$(c,0)+(x,y) \; mod\ P = \Big(\frac{c}{a}(x,y)\Big) \bmod P.$$

We have that:

$$(c,0)+(x,y) \; mod\ P = \Big(\frac{1}{a}(cx,cy)\Big) \bmod P = \Big(\frac{c}{a}(x,y)\Big) \bmod P. \tag{3.20}$$

The importance of this theorem is that in order to encode and decode with the cyclic encoder $E_p(a,b)$ we have to choose a key $(k_x,k_y) \in E_p(a,b)$ with additive inverse $(k_x, P-k_y) \in E_p(a,b)$. We break the message into pairs of integers (m_x,m_y), where $m_x, m_y < P$ regardless of whether (m_x,m_y) belongs to $E_p(a,b)$ or not. We have:

$$(c_x,c_y) = (m_x,m_y)+(k_x,k_y) \bmod P = \frac{1}{a}\Big(m_xk_x - \frac{a^2}{b^2}m_yk_y, m_xk_y+m_yk_x\Big) \bmod P. \tag{3.21}$$

To decode the encoded message we add the inverse key to the ciphertext and we obtain:

$$(c_x,c_y)+(k_x,P-k_y) = \frac{1}{a}\Big(c_xk_x - \frac{a^2}{b^2}c_y(P-k_y), c_x(P-k_y)+c_yk_x\Big) \bmod P$$

$$= \frac{1}{a^2}\Big(m_xk_x^2 - \frac{a^2}{b^2}(m_yk_yk_x + (m_xk_y+m_yk_x)(P-k_y)),$$

$$(m_xk_x - \frac{a^2}{b^2}m_yk_y)(P-k_y) + (m_xk_y+m_yk_x)k_x\Big) \bmod P$$

$$= \frac{1}{a^2}\left(a^2m_x\Big(\frac{k_x^2}{a^2}+\frac{k_y^2}{b^2}\Big), a^2m_y\Big(\frac{k_x^2}{a^2}+\frac{k_y^2}{b^2}\Big)\right) \bmod P = \frac{a^2}{a^2}(m_x,m_y) \bmod P = (m_x,m_y) \tag{3.22}$$

So the algorithm can encode and decode. The third and fourth part of the theorem gives a shortcut formula for additions between two elements of $E_p(a,b)$, where one element has a y component zero.

3.4.2 The Discrete Logarithm Problem in Elliptic Cryptography

If we consider a point $(x_1, y_1) \in E_p(a,b)$, then the order n of the point G is the smallest positive integer n such that

$$nG = (a,0). \tag{3.23}$$

If we consider the equation

$$Q = kP, \tag{3.24}$$

where $P, Q \in E_p(a,b)$, and $k < n$, then it is relatively easy to calculate Q, given k and P, but is relatively hard to find k, given Q and P. This is the discrete logarithm problem in elliptic cryptography.

Let F_p be the finite field of order p with p being an odd prime. The discrete logarithm problem for genus 0 curves of the form

$$x^2 - Dy^2 = 1, \tag{3.25}$$

where the solutions $(x,y) \in F_p x F_p$ are points in the group and $D \in F_p, D \neq 0$, was analyzed by Menezes and Vanstone in 1992 [13]. Ellipses are genus 0 curves, and if we let $r = \frac{x}{a}, s = \frac{y}{b}$, and $D = -1$, we have

$$r^2 - Ds^2 = 1 = \frac{x^2}{a^2} + \frac{y^2}{b^2}. \tag{3.26}$$

Thus, the homomorphisms presented in their paper work for elliptic cryptography. Therefore, if -1 is a quadratic residue in F_p, then $E_p(a,b)$ is homomorphic to F_p^* with the reduction needing constant time after first computing $\sqrt{-1}$ [13]. If -1 is not a quadratic residue in F_p, then $E_p(a,b)$ is homomorphic to the $p+1$ subgroup of F_{p^2} with the reduction needing constant time [13]. Now the discrete logarithm problem in $E_p(a,b)$ is equivalent to the discrete logarithm problem in either F_p, with the necessary precomputation, or F_{p^2}.

Discrete logarithms in F_p have been computable using the number field sieve in the subexponential time of $L_p\left[\frac{1}{3}; 3^{2/3}\right]$ since 1993 [14]. Subexponential time is represented here by:

$$L_{n\to\infty}\left[v;c\right] = exp\Big\{\big(c+o(1)\big)(logn)^v(loglogn)^{1-v}\Big\} \tag{3.27}$$

[14]. The value of c has dropped with optimizations through the years. Since 2006, discrete logarithms in F_p have been computable using two stages: a precomputation of $L_p\left[\frac{1}{3}; \left(\frac{64}{9}\right)^{1/3}\right]$, and $L_p\left[\frac{1}{3}; 3^{1/3}\right]$ for each subsequent logarithm to compute [15]. An odd number can be factored in

$L_p\left[\frac{1}{3};\left(\frac{64}{9}\right)^{1/3}\right]$ using the general number field sieve [16]. Therefore an RSA system modulus n has an equivalent level of security as $E_p(a,b)$, where n and p have roughly the same number of bits and -1 is not a quadratic residue in F_p.

The case of F_{p^k} has seen progress in the last few years. Up until quite recently, there was a notion that "an RSA modulus n and a finite field F_{p^k} therefore offer about the same level of security if n and p^k are of the same order of magnitude" [17]. This would imply that the F_{p^2} case would have greater security than RSA and the F_p case for p and n of the same order of magnitude. Barbulescu et al. investigated this claim for F_{p^k} where k is a small integer using their new methods of finding polynomials to define extension fields [18]. They found that it was much easier to compute a discrete logarithm in F_{p^2} where p was 90 bits than it was in F_p where p where was 180 bits [18]. Complexity-wise current algorithms break up the DLP in F_{p^k} by comparing p to $L\left[l;c\right]$ with varying values of l as the delimiters [19]. In all cases, the complexity is $L_{p^k}\left[\frac{1}{3};c\right]$ where c has improved recently [19]. For the large case, when $p = L_p\left[l;c\right]$ and $\frac{2}{3} < l \leq 1$, Sarkar and Singh's *NFS - A* algorithm has complexity $L_{p^k}\left[\frac{1}{3};\left(\frac{64}{9}\right)^{1/3}\right]$ [20]. For the medium case of $\frac{1}{3} < l < \frac{2}{3}$, Kim and Jeong generalized the extended tower number field sieve to get a complexity of $L_{p^k}\left[\frac{1}{3};\left(\frac{48}{9}\right)^{1/3}\right]$ [21]. The complexity of the boundary case, $l = \frac{2}{3}$, had the lowest c value for the *NFS - A* algorithm [20]. The final case, where $0 \leq l \leq \frac{1}{3}$, the complexity is $L\left[\frac{1}{4};o(1)\right]$ [22]. Therefore, the notion that the discrete logarithm problem in F_{p^k} is as secure as RSA when n and p^k are of the same order of magnitude can only be true if p is a large prime in F_{p^k}.

Elliptic cryptography compares well with RSA in that it is equivalently secure when -1 is not a quadratic residue in F_p for n and p of roughly the same number of bits, and possibly more secure when -1 is a quadratic residue in F_p and p is a large prime in F_{p^2}. XTR uses a subgroup of F_{p^6} which should make is as secure as RSA when n and p^6 have roughly the same number of bits [23]. However this is only true of the case where p is a large prime in F_{p^6} [20, 21, 22]. XTR should, in general, be more secure for a given prime than elliptic cryptography. Elliptic curve cryptography currently only has exponential attacks, the fastest with complexity $O\left(\sqrt{p}\right)$ [24]. For the same p it is much more secure than cryptosystems that are susceptible to the number field sieve, like RSA and elliptic cryptography.

3.4.3 Key Exchange Analogous to Diffie-Hellman

We first pick a large prime number P and two radii a and b. This defines the cyclic group $E_p(a,b)$. Then we choose a base point $G = (x,y) \in E_p(a,b)$. Since the prime number P is large and the number of elements in the cyclic group $E_p(a,b)$ is greater than P, G can be chosen such that its order is greater than P. So G has a relatively larger order. A key exchange between users A and B can be accomplished as follows:

1. A selects an integer k_A less than n to be A's private key. Subsequently A generates a public key $K_A = k_A \cdot G$. A's public key belongs to the group $E_p(a,b)$.

2. B selects an integer k_B less than n to be B's private key. Subsequently B generates a public key $K_B = k_B \cdot G$. B's public key belongs to the group $E_p(a,b)$.

3. A generates the secret key $K = k_A \cdot K_B$. B generates the secret key $K = k_B \cdot K_A$.

Notice that $K = k_a \cdot K_B = k_A k_B G = k_B k_A G = k_B K_A$. For an attacker to break this algorithm they would need to be able to compute k_A, given G and K_A, which we refer to as the elliptic cryptographic logarithm problem, it is hard when the chosen prime number P is 128 bits or more.

3.4.4 Encryption-Decryption using Elliptic Cryptography

If user A wants to send a message to user B, first user A selects a sufficiently large prime number P and two integer radii a and b. Then A selects a base pair $G \in E_p(a,b)$ with maximum order n. A then initiates a Diffie-Helman key exchange with B, to compute a common key as described in the previous section. A then breaks the message into blocks of pairs and encrypts each block of pairs using the common key until the entire message is encrypted. A transmits the encrypted message to B. Based on the common key, B computes the additive inverse key. B breaks the ciphertext into blocks of pairs. To each pair, B adds the inverse key until the entire message is decrypted.

3.4.5 Digital Signing

To encrypt, we use a key $(x_1, y_1) \in E_p(a,b)$ that is distributed during the key exchange to the participants only, and for decryption the unique inverse key is used. Only the two people participating in the message exchange know the key, therefore the person receiving the message knows that the message came from the only other person that has the key. Therefore, the algorithm is good for digital signing.

3.5 Generalization of the Algorithm to an Ellipse of Arbitrary Center

The algorithm can be easily extended to use ellipses with arbitrary center (c, d). If we denote the set of elements on the circumference of an ellipse with center (c, d) and radii a and b by $E(a, b, c, d)$ and define the addition of any two elements $(x_1, y_1), (x_2, y_2)$ of this set as:

$$(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$$
$$= \left(c + \frac{1}{a}\Big((x_1 - c)(x_2 - c) - \frac{a^2}{b^2}(y_1 - d)(y_2 - d)\Big), d + \frac{1}{a}\big((x_1 - c)(y_2 - d) + (x_2 - c)(y_1 - d)\big)\right) \quad (3.28)$$

then

$$\frac{(x_3 - c)^2}{a^2} + \frac{(y_3 - d)^2}{b^2}$$
$$= \frac{(x_1 - c)^2(x_2 - c)^2}{a^4} + \frac{(y_1 - d)^2(y_2 - d)^2}{b^4} + \frac{(x_2 - c)^2(y_1 - d)^2}{a^2b^2} + \frac{(x_1 - c)^2(y_2 - d)^2}{a^2b^2}$$
$$= \Big(\frac{(x_1 - c)^2}{a^2} + \frac{(y_1 - d)^2}{b^2}\Big)\Big(\frac{(x_2 - c)^2}{a^2} + \frac{(y_2 - d)^2}{b^2}\Big) = 1 \quad (3.29)$$

Hence $(x_3, y_3) \in E(a, b, c, d)$. Closure under the addition operation as it is defined here is therefore satisfied.

Next we verify that the identity element is $e = (a + c, d) \in E(a, b, c, d)$. Consider any element $S = (x, y) \in E(a, b, c, d)$, then based on the closure property we just proved, the sum $S + e$ is also an element of $E(a, b, c, d)$. We have that:

$$S + e = e + S = \Big(c + \frac{1}{a}\big((x - c)a\big), d + \frac{1}{a}\big(a(y - d)\big)\Big) = (x, y). \quad (3.30)$$

Hence e is the identity element of the set $E(a, b, c, d)$. If there is also another identity element $e' \in E(a, b, c, d)$, then since e is an identity element,

$$e + e' = e'.$$

Now since e' is and identity element,

$$e + e' = e$$

which implies

$$e = e'. \quad (3.31)$$

Hence, the identity element is unique.

Next we show that for every element $S = (x, y) \in E(a, b, c, d), S' = (x, d - y) \in E(a, b, c, d)$ is the additive inverse of S. We have that:

$$S + S' = \left(c + \frac{1}{a}\Big((x - c)^2 + \frac{a^2}{b^2}(y - d)^2\Big), d + \frac{1}{a}\Big((x - c)(d - y) + (x - c)(y - d)\Big)\right)$$

$$= \left(c + a\Big(\frac{(x-c)^2}{a^2} + \frac{(y-d^2)}{b^2}\Big), d\right) = (a+c, d) = e \tag{3.32}$$

For every $S = (x, y)$ there is only one inverse S'. Since if there was another inverse S'' then

$$S'' + S = S' + S = e$$

or

$$S'' = S' + S - S$$

meaning

$$S'' = S'. \tag{3.33}$$

Next we prove that the associative law holds. We will show that if $S_1 = (x_1, y_1), S_2 = (x_2, y_2)$ and $S_3 = (x_3, y_3)$, where $S_1, S_2, S_3 \in E(a, b, c, d)$ then $S_1 + (S_2 + S_3) = (S_1 + S_2) + S_3$. We have:

$$\begin{gathered}
S_1 + (S_2 + S_3) \\
= S_1 + \left(c + \frac{1}{a}\Big((x_2-c)(x_3-c) - \frac{a^2}{b^2}(y_2-d)(y_3-d)\Big), d + \frac{1}{a}\big((x_2-c)(y_3-d) + (x_3-c)(y_2-d)\big)\right) \\
= \left(c + \frac{1}{a^2}\Big((x_3-c)\big((x_1-c)(x_2-c) - \frac{a^2}{b^2}(y_1-d)(y_2-d)\big) - \frac{a^2}{b^2}(y_3-d)\big((x_1-c)(y_2-d) + (y_1-d)(x_2-c)\big)\Big),\right. \\
\left. d + \frac{1}{a^2}\Big((y_3-d)\Big((x_1-c)(x_2-c) - \frac{a^2}{b^2}(y_1-d)(y_2-d)\Big) + (x_3-c)\big((x_1-c)(y_2-d) + (y_1-d)(x_2-c)\big)\Big)\right) \\
= (S_1 + S_2) + S_3 = S_1 + S_2 + S_3
\end{gathered} \tag{3.34}$$

Finally, for any two elements, $S_1 = (x_1, y_1), S_2 = (x_2, y_2) \in E(a, b, c, d)$, we have:

$$\begin{gathered}
S_1 + S_2 = (x_1, y_1) + (x_2, y_2) \\
= \left(c + \frac{1}{a}\Big((x_1-c)(x_2-c) - \frac{a^2}{b^2}(y_1-d)(y_2-d)\Big), d + \frac{1}{a}\big((x_1-c)(y_2-d) + (x_2-c)(y_1-d)\big)\right) \\
= S_2 + S_1
\end{gathered} \tag{3.35}$$

Hence $E(a, b, c, d)$ is an Abelian group.

In order for $E(a, b, c, d)$ to be used for cryptography, a, b, c, and d have to be integers, and a relatively large number P has to be selected. Then the new formulation suitable for cryptography is of the form:

$$\frac{(x-c)^2}{a^2} + \frac{(y-d)^2}{b^2} \bmod P = 1 \bmod P \tag{3.36}$$

The key exchange and management are similar to that of 3.4.3 where $(0, 0)$ is the center of the ellipse. Digital signing follows in a similar fashion to that of the previous section.

3.6 Conclusion

A new cryptographic algorithm was introduced in this research paper. The algorithm incorporates phase modulation and modulation with respect to a relatively large prime number. The use of integer points not in the group, rather than being limited to only points in the group, increases the ease of encoding. The algorithm can be broken in subexponential time, with a worst case of $L_p\left[\frac{1}{3}; \left(\frac{64}{9}\right)^{1/3}\right]$. The algorithm can also be used for key exchange, key management, key distribution, digital signing and digital signatures, authentication, as well as encryption and decryption.

Chapter 4

Examples

To see the math in a less abstract manner, small examples for both the $(0,0)$ and arbitrary center versions of the elliptic cryptographic algorithm will be presented. The details of finding the points in the group and how to determine the order of the elements will be discussed. A base point will be picked and used to create a shared key with two secret keys, each less than the order of the base point. The shared key will be used to encrypt a small familiar message. Finally, the message will be decrypted.

4.1 Example Using $E_p(a, b)$

First the prime number and radii are chosen to be 11, 5, and 6 respectively. The equation defining $E_{11}(5, 6)$ is:

$$y^2 = 3 - x^2 \bmod 11. \tag{4.1}$$

To find the points that solve this equation, it is helpful to note which integers modulo 11 are quadratic residues, or perfect squares. These are 1, 3, 4, 5, and 9. Now testing for all the possible values of x mod 11 will reveal the points.

x	$y^2 = 3 - x^2 \bmod 11$	Residue	y
0	3	yes	5, 6
1	2	no	—
2	10	no	—
3	5	yes	4, 7
4	9	yes	3, 8
5	0	yes	0
6	0	yes	0
7	9	yes	3, 8
8	5	yes	4, 7
9	10	no	—
10	2	no	—

Table 4.1: Points in the $E_{11}(5,6)$ Group

$E_{11}(5,6)$ has 12 points as shown in table 4.1. One of these points will chosen as the base point. The base point should have a high order within the group. Recall the definition of order from section 3.4.2 (equation 3.23). Ideally the base point should have as high an order as possible. For this small example, it is simple to examine the Cayley table (table 4.3) of the group to see all possible addition pairs and results within the group. Elements with the highest order are candidates for the base point.

Order	Elements
1	(5,0)
2	(6,0)
3	(3,4), (3,7)
4	(0,5), (0,6)
6	(8,4), (8,7)
12	(4,3), (4,8), (7,3), (7,8)

Table 4.2: Order of the Points in the $E_{11}(5,6)$ Group

For this example, the point $(4,3)$ will be the base point with order $n = 12$. A will chose the

	(0,5)	(0,6)	(3,4)	(3,7)	(4,3)	(4,8)	(5,0)	(6,0)	(7,3)	(7,8)	(8,4)	(8,7)
(0,5)	(6,0)	(5,0)	(7,3)	(4,3)	(8,4)	(3,4)	(0,5)	(0,6)	(8,7)	(3,7)	(7,8)	(4,8)
(0,6)	(5,0)	(6,0)	(4,8)	(7,8)	(3,7)	(8,7)	(0,6)	(0,5)	(3,4)	(8,4)	(4,3)	(7,3)
(3,4)	(7,3)	(4,8)	(3,7)	(5,0)	(0,5)	(7,8)	(3,4)	(8,7)	(4,3)	(0,6)	(6,0)	(8,4)
(3,7)	(4,3)	(7,8)	(5,0)	(3,4)	(7,3)	(0,6)	(3,7)	(8,4)	(0,5)	(4,8)	(8,7)	(6,0)
(4,3)	(8,4)	(3,7)	(0,5)	(7,3)	(8,7)	(5,0)	(4,3)	(7,8)	(6,0)	(3,4)	(4,8)	(0,6)
(4,8)	(3,4)	(8,7)	(7,8)	(0,6)	(5,0)	(8,4)	(4,8)	(7,3)	(3,7)	(6,0)	(0,5)	(4,3)
(5,0)	(0,5)	(0,6)	(3,4)	(3,7)	(4,3)	(4,8)	(5,0)	(6,0)	(7,3)	(7,8)	(8,4)	(8,7)
(6,0)	(0,6)	(0,5)	(8,7)	(8,4)	(7,8)	(7,3)	(6,0)	(5,0)	(4,8)	(4,3)	(3,7)	(3,4)
(7,3)	(8,7)	(3,4)	(4,3)	(0,5)	(6,0)	(3,7)	(7,3)	(4,8)	(8,4)	(5,0)	(0,6)	(7,8)
(7,8)	(3,7)	(8,4)	(0,6)	(4,8)	(3,4)	(6,0)	(7,8)	(4,3)	(5,0)	(8,7)	(7,3)	(0,5)
(8,4)	(7,8)	(4,3)	(6,0)	(8,7)	(4,8)	(0,5)	(8,4)	(3,7)	(0,6)	(7,3)	(3,4)	(5,0)
(8,7)	(4,8)	(7,3)	(8,4)	(6,0)	(0,6)	(4,3)	(8,7)	(3,4)	(7,8)	(0,5)	(5,0)	(3,7)

Table 4.3: Cayley Table for the $E_{11}(5,6)$ Group

secret key $2 < 12$, and B will chose the secret key $8 < 12$. A will therefore have the public key:

$$2 * (4,3) = (4,3) + (4,3) = (8,7). \tag{4.2}$$

B calculates a public key, $(3,4)$, using the base point and B's secret key 8 the same way as A. The encryption key will be calculated by both.

$$2 * (3,4) = 8 * (8,7) = (3,7) \tag{4.3}$$

$(3,4)$ will be used to decrypt. These two are inverses, which can be seen in table 4.3. The addition of these two points yields the identity $(5,0)$. The point used for decryption must be the inverse of the encryption key.

To encrypt the message, "hello world," it must first be converted to points but not necessarily elements of $E_{11}(5,6)$. Points can be created by converting the letters to their ASCII codes. In this case, $104, 101, 108, 108, 111, 119, 111, 114, 108$, and 100. For conveniece, 100 will be subtracted from each. The points are: $(0,4), (0,1), (0,8), (0,8), (1,1), (1,9), (1,1), (1,4), (0,8)$ and $(0,0)$. Encryption consists only of adding the key to each point using the addition operation of $E_{11}(5,6)$. For example:

$$9(3*0 - 4*7, 4*3 + 0*7) \bmod 11 = (1,9). \tag{4.4}$$

The resulting encrypted message is: $(1,9), (3,5), (2,7), (2,7), (8,2), (10,9), (8,2), (6,6), (2,7)$, and $(0,0)$.

Decryption is achieved by adding the inverse of the key to the encrypted points.

$$9(3*1 - 4*9, 3*9 + 4*1) \bmod 11 = (0,4) \tag{4.5}$$

The original points are recovered: $(0,4),(0,1),(0,8),(0,8),(1,1),(1,9),(1,1),(1,4),(0,8)$ and $(0,0)$. Different schemes can be used to break the message into points, but values of x and y in the points must be less than the prime. If an x or y value is larger than the prime, the modulus operation will map that value to a value lower than the prime that might already be used in the scheme. The encryption and decryption operations will not be bijective in that case.

4.2 Example Using $E_p(a,b,c,d)$

First the prime number and radii are chosen to be 11, 7, 5, 7, and 9 respectively. The equation defining $E_{11}(7,5,7,9)$ is:

$$(y-9)^2 = 3 - 5(x-7)^2 \bmod 11. \tag{4.6}$$

To figure out which points solve this equation, finding the quadratic residues modulo 11 can be helpful. These are the same as in the previous example: 1, 3, 4, 5, and 9. Now testing all the possible values of x mod 11 will reveal the points. $E_{11}(7,5,7,9)$ has 12 points as shown in table

x	$(y-9)^2 = 3-5(x-7)^2$ mod 11	Residue	y
0	0	yes	9
1	10	no	—
2	10	no	—
3	0	yes	9
4	2	no	—
5	5	yes	2, 5
6	9	yes	1, 6
7	3	yes	3, 4
8	9	yes	1, 6
9	5	yes	2, 5
10	2	no	—

Table 4.4: Points in the $E_{11}(7,5,7,9)$ Group

4.4. Like the previous example, the base point will be chosen from among these points. The points with the highest order should be the possible choices. The points need to be checked one by one for order. Since this example features a small group, a Cayley table (table 4.6) can be constructed and used to check the result of any additions within the group.

Order	Elements
1	(3,9)
2	(0,9)
3	(9,2), (9,5)
4	(7,3), (7,4)
6	(5,2), (5,5)
12	(6,1), (6,6), (8,1), (8,6)

Table 4.5: Order of the Points in the $E_{11}(7,5,7,9)$ Group

	(0,9)	(3,9)	(5,2)	(5,5)	(6,1)	(6,6)	(7,3)	(7,4)	(8,1)	(8,6)	(9,2)	(9,5)
(0,9)	(3,9)	(0,9)	(9,5)	(9,2)	(8,6)	(8,1)	(7,4)	(7,3)	(6,6)	(6,1)	(5,5)	(5,2)
(3,9)	(0,9)	(3,9)	(5,2)	(5,5)	(6,1)	(6,6)	(7,3)	(7,4)	(8,1)	(8,6)	(9,2)	(9,5)
(5,2)	(9,5)	(5,2)	(9,2)	(3,9)	(6,6)	(7,3)	(8,6)	(6,1)	(7,4)	(8,1)	(0,9)	(5,5)
(5,5)	(9,2)	(5,5)	(3,9)	(9,5)	(7,4)	(6,1)	(6,6)	(8,1)	(8,6)	(7,3)	(5,2)	(0,9)
(6,1)	(8,6)	(6,1)	(6,6)	(7,4)	(5,5)	(3,9)	(5,2)	(9,5)	(0,9)	(9,2)	(7,3)	(8,1)
(6,6)	(8,1)	(6,6)	(7,3)	(6,1)	(3,9)	(5,2)	(9,2)	(5,5)	(9,5)	(0,9)	(8,6)	(7,4)
(7,3)	(7,4)	(7,3)	(8,6)	(6,6)	(5,2)	(9,2)	(0,9)	(3,9)	(5,5)	(9,5)	(8,1)	(6,1)
(7,4)	(7,3)	(7,4)	(6,1)	(8,1)	(9,5)	(5,5)	(3,9)	(0,9)	(9,2)	(5,2)	(6,6)	(8,6)
(8,1)	(6,6)	(8,1)	(7,4)	(8,6)	(0,9)	(9,5)	(5,5)	(9,2)	(5,2)	(3,9)	(6,1)	(7,3)
(8,6)	(6,1)	(8,6)	(8,1)	(7,3)	(9,2)	(0,9)	(9,5)	(5,2)	(3,9)	(5,5)	(7,4)	(6,6)
(9,2)	(5,5)	(9,2)	(0,9)	(5,2)	(7,3)	(8,6)	(8,1)	(6,6)	(6,1)	(7,4)	(9,5)	(3,9)
(9,5)	(5,2)	(9,5)	(5,5)	(0,9)	(8,1)	(7,4)	(6,1)	(8,6)	(7,3)	(6,6)	(3,9)	(9,2)

Table 4.6: Cayley Table for the $E_{11}(7,5,7,9)$ Group

For this example, the point $(8,1)$ will be the base point. A will choose the secret key $5 < 12$, and B will chose the secret key $4 < 12$. A's public key is $(6,1)$, and B calculates a public key as shown in equation 4.7.

$$4*(8,1) = (8,1)+(8,1)+(8,1)+(8,1) = (5,2)+(5,2) = (9,2) \tag{4.7}$$

Both calculate the shared encryption key using their own private key and the other's public key as shown in equation 4.8.

$$4*(6,1) = 5*(9,2) = (9,5) \tag{4.8}$$

The decryption key is $(9,2)$. These two are inverses, which can be seen in table 4.6. The addition of these two points yields the identity $(3,9)$. The point used for decryption must be the inverse of the encryption key.

To encrypt the message, "hello world," it must first be converted to points but not necessarily elements of $E_{11}(7,5,7,9)$. Points can be created by converting the letters to their ASCII codes. In this case, $104, 101, 108, 108, 111, 119, 111, 114, 108$, and 100. For conveniece, 100 will be subtracted from each. The points are: $(0,4), (0,1), (0,8), (0,8), (1,1), (1,9), (1,1), (1,4), (0,8)$ and $(0,0)$. Encryption consists only of adding the key to each point using the addition operation of $E_{11}(7,5,7,9)$. For example:

$$\Big(7+8\big((0-7)(9-7)-9(4-9)(5-9)\big), 9+8\big((9-7)(4-9)+(0-7)(5-9)\big)\Big) \bmod 11 = (6,10). \quad (4.9)$$

The resulting encrypted message is: $(6,10), (0,6), (3,8), (3,8), (5,7), (10,3), (5,7), (0,0), (3,8)$, and $(9,1)$.

Decryption is achieved by adding the inverse of the key to the encrypted points.

$$\Big(7+8\big((9-7)(6-7)-9(2-9)(10-9)\big), 9+8\big((9-7)(10-9)+(6-7)(2-9)\big)\Big) \bmod 11 = (0,4) \quad (4.10)$$

The original points are recovered: $(0,4), (0,1), (0,8), (0,8), (1,1), (1,9), (1,1), (1,4), (0,8)$ and $(0,0)$. Different schemes can be used to break the message into points, but values of x and y in the points must be less than the prime for the encryption and decryption to be bijective.

Chapter 5

Code Demonstration

The full process from picking a prime to encrypting and decrypting a file will be explained in this chapter with sample C++ code for a relatively small example using the $E_p(a, b)$ version of the algorithm. For high security purposes, the code should be expanded to work with primes on the order of the key sizes recommended by NIST for RSA, usually primes greater than 2048 bits [25]. The purpose of this example is to demonstrate the process of setting up the algorithm.

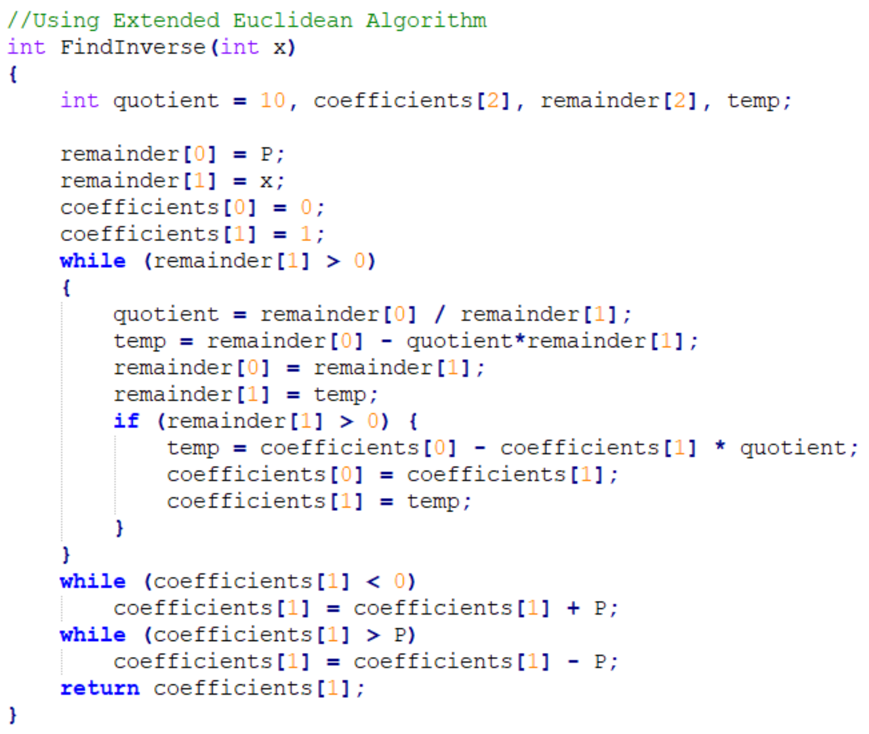
The first step is to choose a prime and radii. For this example 127 was chosen so that the data could be simply read byte by byte. This prime is too small to be safe in most circumstances, but it works for a simple example. The radii were chosen to be 50 and 103. They can be chosen randomly from integers between 0 and the prime.

For the upcoming calculations, it will be necessary to find the inverses of the radii in F_p, or the integer t for each radius a each such that:

$$p * l - a * t = 1. \tag{5.1}$$

These can be computed using the extended Euclidean algorithm as shown in figure 5.1.

The next step is to find points in the group $E_{127}(50, 103)$ that could be chosen as the base point or generator. To find points in the group, it is helpful to keep track of the all the possible quadratic residues, or perfect squares. This makes determining whether there is a solution for y in the equation that defines the group easier. There are $\frac{p-1}{2}$ quadratic roots for a given prime p. Note that positive and negative integers of the same absolute value have the same square. Figure 5.2 has code for calculating and storing the quadratic residues. The type Square is a structured type containing two integers: a square, and one of its roots. After the squares are found, they are sorted by the square values, so binary search can be used to retrieve specific squares. In the full



```
//Using Extended Euclidean Algorithm
int FindInverse(int x)
{
    int quotient = 10, coefficients[2], remainder[2], temp;

    remainder[0] = P;
    remainder[1] = x;
    coefficients[0] = 0;
    coefficients[1] = 1;
    while (remainder[1] > 0)
    {
        quotient = remainder[0] / remainder[1];
        temp = remainder[0] - quotient*remainder[1];
        remainder[0] = remainder[1];
        remainder[1] = temp;
        if (remainder[1] > 0) {
            temp = coefficients[0] - coefficients[1] * quotient;
            coefficients[0] = coefficients[1];
            coefficients[1] = temp;
        }
    }
    while (coefficients[1] < 0)
        coefficients[1] = coefficients[1] + P;
    while (coefficients[1] > P)
        coefficients[1] = coefficients[1] - P;
    return coefficients[1];
}
```

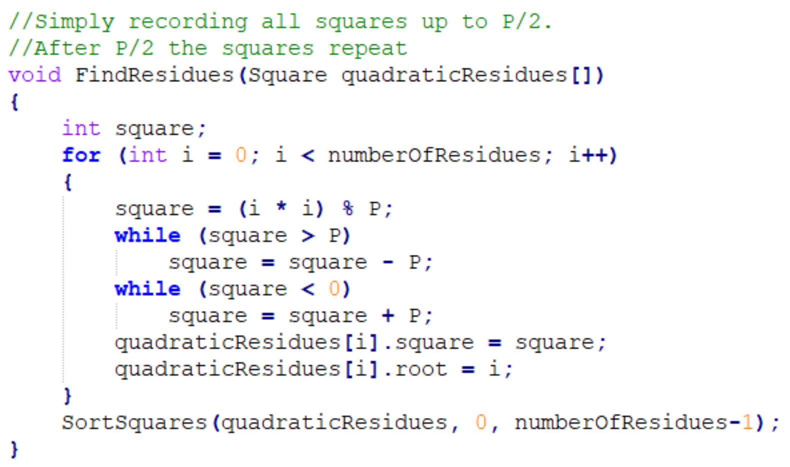
Figure 5.1: This is C++ code for finding the inverses of the radii.

code found in appendix C, quicksort is used. If the prime is relatively large, any sorting algorithm using recursion might cause the stack to overflow, so an iterative approach might work better in that case.

The base point should be a point in the group with high order; the best case would be the same order as the group. To find the order of a point, the point is added to itself, using the group's addition operation, until the sum is the identity element. Figure 5.3 shows a function for adding two points where Point is a structured type containing two integers: x, and y.

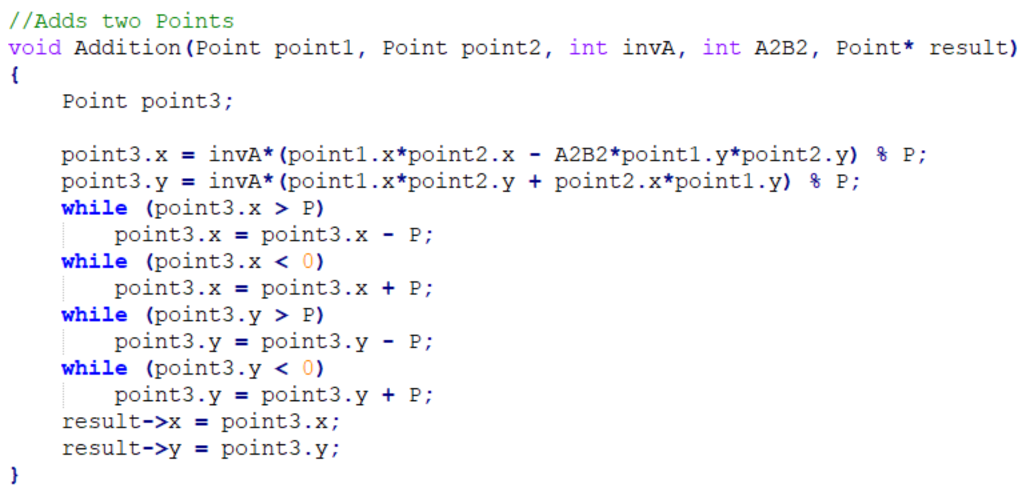
Now a base point can be searched for. To find the points of the group, all possible values of x, integers from 0 to $p - 1$, can be checked to see if a corresponding y exists. Plugging x into the equation for points in the group (see equation 3.12) will yield a value for y^2. If this value is a quadratic residue, then the point exists. If not, then there is no point in the group with the given x value. Once a point is found, it is added to itself until the sum equals the identity so that the order of the point can be determined. If the order of an acceptable size, the point is chosen as the base point. Figure 5.4 shows this process.

All the steps upto this point are part of a precomputation phase. With a base point of known order and a group, $E_p(a, b)$ generated by its parameters, p, a, and b, encryption and decryption can be accomplished freely without needing to find a new base point. A relatively large prime will



```
//Simply recording all squares up to P/2.
//After P/2 the squares repeat
void FindResidues(Square quadraticResidues[])
{
    int square;
    for (int i = 0; i < numberOfResidues; i++)
    {
        square = (i * i) % P;
        while (square > P)
            square = square - P;
        while (square < 0)
            square = square + P;
        quadraticResidues[i].square = square;
        quadraticResidues[i].root = i;
    }
    SortSquares(quadraticResidues, 0, numberOfResidues-1);
}
```

Figure 5.2: This is C++ code for finding the quadratic residues in F_p.



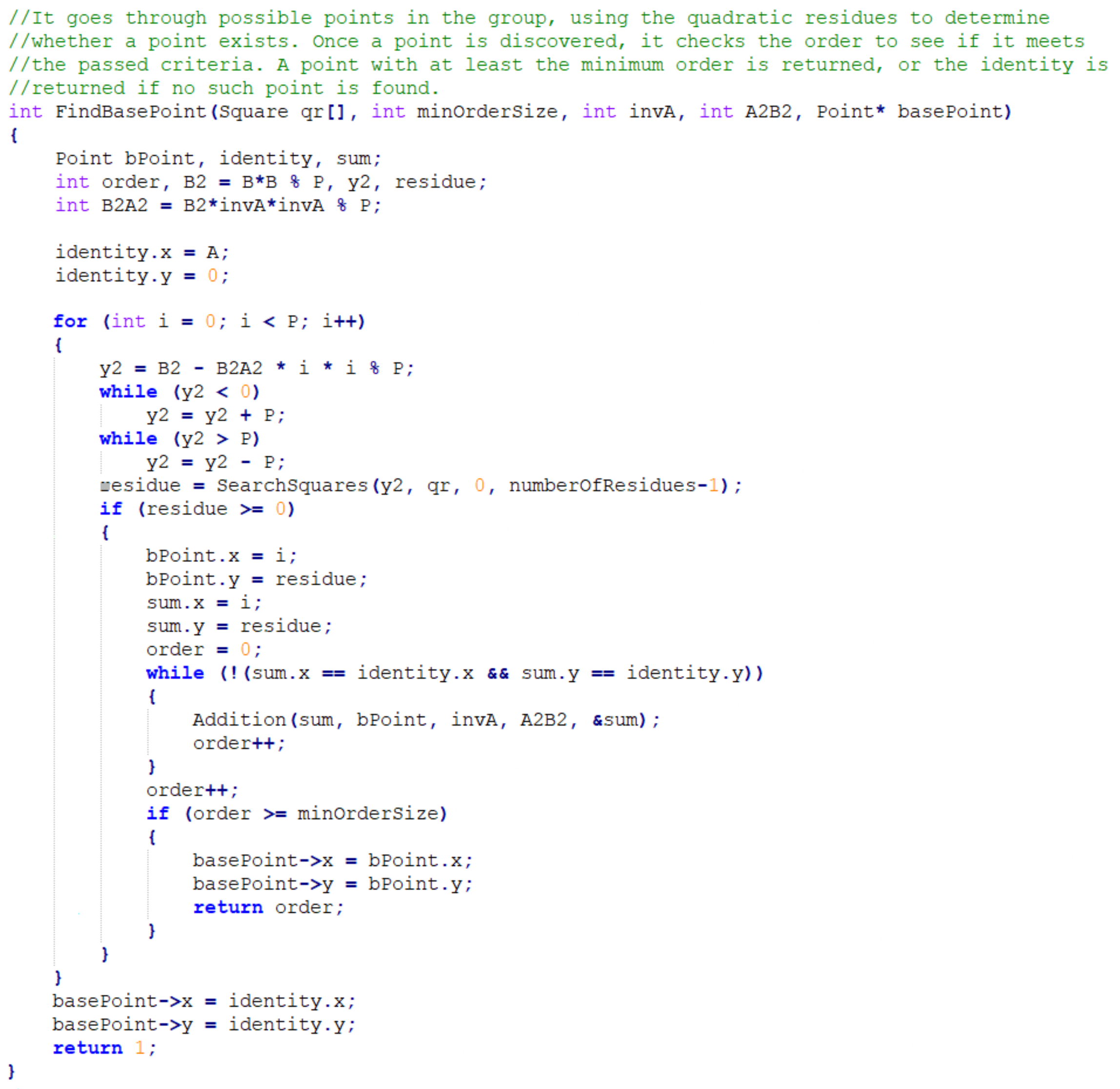
```
//Adds two Points
void Addition(Point point1, Point point2, int invA, int A2B2, Point* result)
{
    Point point3;

    point3.x = invA*(point1.x*point2.x - A2B2*point1.y*point2.y) % P;
    point3.y = invA*(point1.x*point2.y + point2.x*point1.y) % P;
    while (point3.x > P)
        point3.x = point3.x - P;
    while (point3.x < 0)
        point3.x = point3.x + P;
    while (point3.y > P)
        point3.y = point3.y - P;
    while (point3.y < 0)
        point3.y = point3.y + P;
    result->x = point3.x;
    result->y = point3.y;
}
```

Figure 5.3: This is C++ code for adding two points in $E_{127}(50, 103)$.

cause this phase to take some time to compute a base point.

Now A and B, the two entities wishing to communicate, can use a Diffie-Hellman key exchange to generate a shared key. The details of this phase are described in section 3.4.3. Each must first pick a private key between 0 and the prime. Then they use it to calculate public keys that they share with each other. Using their own private key and the other's public key, both can compute the same shared key. This shared key will be used for encryption, and its inverse will be used for decryption. Figure 5.5 shows how A calculates the necessary keys.

A can encrypt a message for B with the shared key simply by breaking the message into points, adding the shared key to each point, and sending the sums to B. Note that the points derived from the message need not be points in the group that A and B have chosen, namely $E_{127}(50, 103)$ in this example. Figure 5.6 shows simple code for encryption using $E_{127}(50, 103)$ and the shared key,



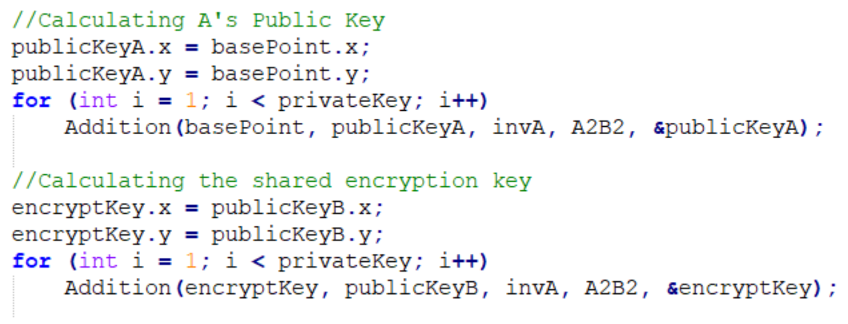
```
//It goes through possible points in the group, using the quadratic residues to determine
//whether a point exists. Once a point is discovered, it checks the order to see if it meets
//the passed criteria. A point with at least the minimum order is returned, or the identity is
//returned if no such point is found.
int FindBasePoint(Square qr[], int minOrderSize, int invA, int A2B2, Point* basePoint)
{
    Point bPoint, identity, sum;
    int order, B2 = B*B % P, y2, residue;
    int B2A2 = B2*invA*invA % P;

    identity.x = A;
    identity.y = 0;

    for (int i = 0; i < P; i++)
    {
        y2 = B2 - B2A2 * i * i % P;
        while (y2 < 0)
            y2 = y2 + P;
        while (y2 > P)
            y2 = y2 - P;
        residue = SearchSquares(y2, qr, 0, numberOfResidues-1);
        if (residue >= 0)
        {
            bPoint.x = i;
            bPoint.y = residue;
            sum.x = i;
            sum.y = residue;
            order = 0;
            while (!(sum.x == identity.x && sum.y == identity.y))
            {
                Addition(sum, bPoint, invA, A2B2, &sum);
                order++;
            }
            order++;
            if (order >= minOrderSize)
            {
                basePoint->x = bPoint.x;
                basePoint->y = bPoint.y;
                return order;
            }
        }
    }
    basePoint->x = identity.x;
    basePoint->y = identity.y;
    return 1;
}
```

Figure 5.4: This is C++ code for finding a base point using $E_{127}(50, 103)$.

$(40, 11)$.

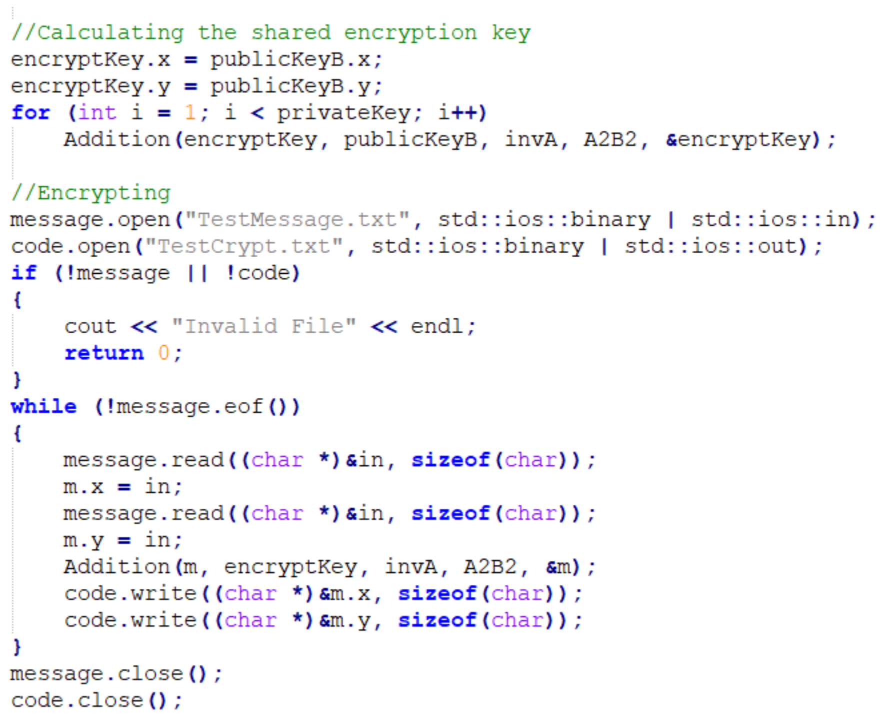
The code for decryption is similar except the decryption key is added to the ciphertext instead of the shared key. The decryption key is simply the inverse of the shared key. Figure 5.7 shows code for decryption.

The full code for this little example can be found in appendix C.



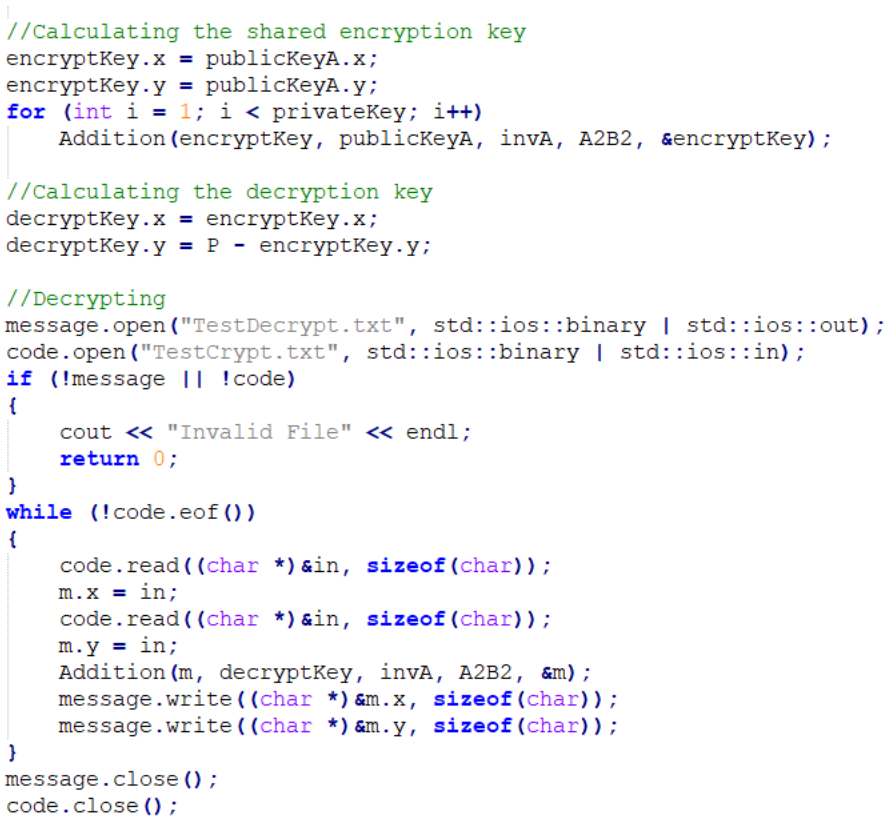
```
//Calculating A's Public Key
publicKeyA.x = basePoint.x;
publicKeyA.y = basePoint.y;
for (int i = 1; i < privateKey; i++)
    Addition(basePoint, publicKeyA, invA, A2B2, &publicKeyA);

//Calculating the shared encryption key
encryptKey.x = publicKeyB.x;
encryptKey.y = publicKeyB.y;
for (int i = 1; i < privateKey; i++)
    Addition(encryptKey, publicKeyB, invA, A2B2, &encryptKey);
```

Figure 5.5: This is C++ code for calculating A's public key and the shared key using $E_{127}(50, 103)$.



```
//Calculating the shared encryption key
encryptKey.x = publicKeyB.x;
encryptKey.y = publicKeyB.y;
for (int i = 1; i < privateKey; i++)
    Addition(encryptKey, publicKeyB, invA, A2B2, &encryptKey);

//Encrypting
message.open("TestMessage.txt", std::ios::binary | std::ios::in);
code.open("TestCrypt.txt", std::ios::binary | std::ios::out);
if (!message || !code)
{
    cout << "Invalid File" << endl;
    return 0;
}
while (!message.eof())
{
    message.read((char *)&in, sizeof(char));
    m.x = in;
    message.read((char *)&in, sizeof(char));
    m.y = in;
    Addition(m, encryptKey, invA, A2B2, &m);
    code.write((char *)&m.x, sizeof(char));
    code.write((char *)&m.y, sizeof(char));
}
message.close();
code.close();
```

Figure 5.6: This is C++ code for encryption using $E_{127}(50, 103)$.



```cpp
//Calculating the shared encryption key
encryptKey.x = publicKeyA.x;
encryptKey.y = publicKeyA.y;
for (int i = 1; i < privateKey; i++)
    Addition(encryptKey, publicKeyA, invA, A2B2, &encryptKey);

//Calculating the decryption key
decryptKey.x = encryptKey.x;
decryptKey.y = P - encryptKey.y;

//Decrypting
message.open("TestDecrypt.txt", std::ios::binary | std::ios::out);
code.open("TestCrypt.txt", std::ios::binary | std::ios::in);
if (!message || !code)
{
    cout << "Invalid File" << endl;
    return 0;
}
while (!code.eof())
{
    code.read((char *)&in, sizeof(char));
    m.x = in;
    code.read((char *)&in, sizeof(char));
    m.y = in;
    Addition(m, decryptKey, invA, A2B2, &m);
    message.write((char *)&m.x, sizeof(char));
    message.write((char *)&m.y, sizeof(char));
}
message.close();
code.close();
```

Figure 5.7: This is C++ code for decryption using $E_{127}(50, 103)$.

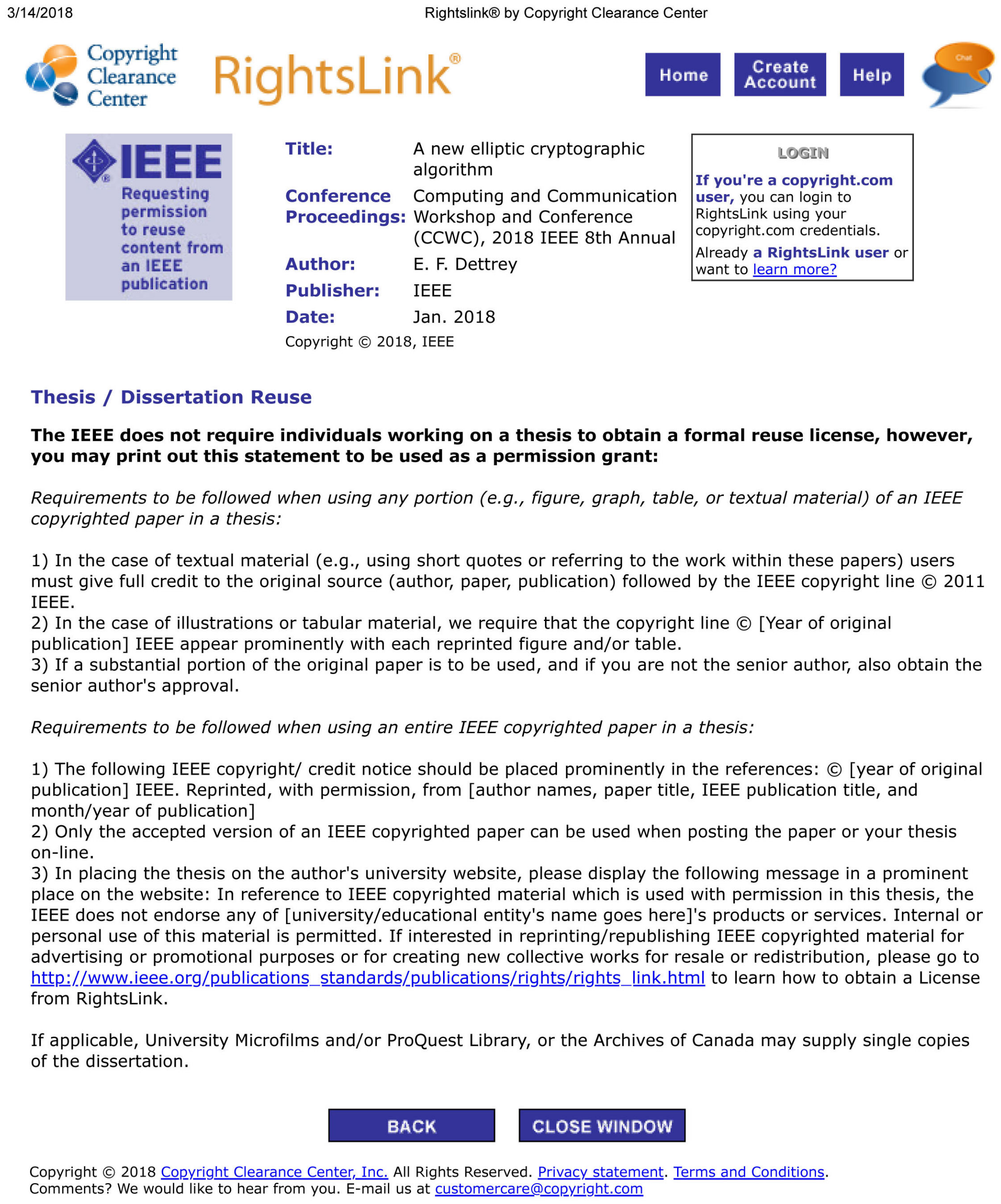
Chapter 6

Conclusion

A new cryptographic algorithm using a group based system relying on the discrete logarithm problem was examined. Using the points on an ellipse of arbitrary center and radii, a group can be created and used for cryptography. The elliptic cryptographic algorithm is fit for encryption, digital signing, and key distribution and management at this time. In the worst case, it can be broken in $L_p\left[\frac{1}{3}; \left(\frac{64}{9}\right)^{1/3}\right]$ time [15]. If quantum computers able to run Shor's algorithm for large groups become prevalent, then this algorithm will no longer be safe. For the mean time, it is recommended that the prime and key chosen follow the sizes recommended for RSA schemes, because the two algorithms have similar security for similar key orders.

Appendix A

IEEE Permission



3/14/2018 Rightslink® by Copyright Clearance Center

Copyright Clearance Center
RightsLink®
Home
Create Account
Help
Chat
IEEE
Requesting permission to reuse content from an IEEE publication

Title:	A new elliptic cryptographic algorithm
Conference Proceedings:	Computing and Communication Workshop and Conference (CCWC), 2018 IEEE 8th Annual
Author:	E. F. Dettrey
Publisher:	IEEE
Date:	Jan. 2018

Copyright © 2018, IEEE

LOGIN

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials.
Already **a RightsLink user** or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

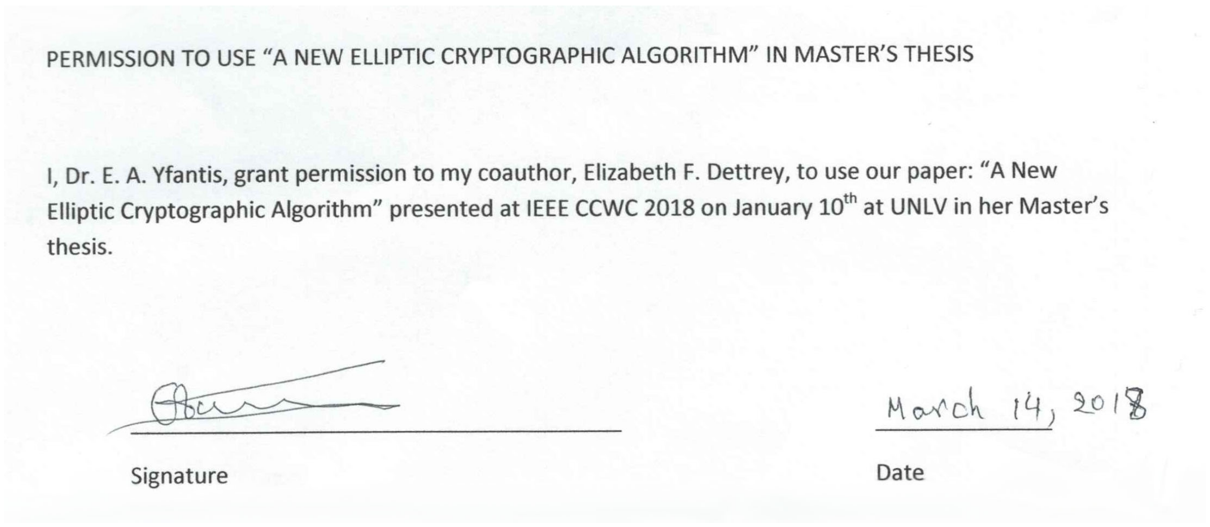
Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Appendix B

Second Author Permission



PERMISSION TO USE "A NEW ELLIPTIC CRYPTOGRAPHIC ALGORITHM" IN MASTER'S THESIS

I, Dr. E. A. Yfantis, grant permission to my coauthor, Elizabeth F. Dettrey, to use our paper: "A New Elliptic Cryptographic Algorithm" presented at IEEE CCWC 2018 on January 10th at UNLV in her Master's thesis.

[illegible]

Signature

March 14, 2018

Date

Appendix C

Full Code for the Demonstration in Chapter 5

```
//Program to find a base point with order greater than a given value
#include <iostream>
#include <fstream>;

struct Point {
    int x = 0;
    int y = 0;
};

struct Square {
    int square;
    int root;
};

int FindInverse(int);
void FindResidues(Square[]);
void SortSquares(Square[], int, int);
int SearchSquares(int, Square[], int, int);
void SwapSquares(Square*, Square*);
void Addition(Point, Point, int, int, Point*);
int FindBasePoint(Square[], int, int, int, Point*);

const int P = 127, A = 50, B = 103;
const int numberOfResidues = (int)((P - 1) / 2);

using namespace std;
int main() {
    int invA, invB, A2B2, bpOrder;
    Square quadraticResidues[numberOfResidues];
    Point basePoint;
    char in = 'a';

    //Finding inverses of A and B
    invA = FindInverse(A);
    invB = FindInverse(B);

    //Calculating a^2/b^2 for convenience in later additions
    A2B2 = (A*A %P)*invB*invB % P;
    while (A2B2 > P)
        A2B2 = A2B2 - P;
    while (A2B2 < 0)
        A2B2 = A2B2 + P;

    //Finding all the quadratic residues
    FindResidues(quadraticResidues);

    //Finding a base point
    bpOrder = FindBasePoint(quadraticResidues, P - 1, invA, invB, &basePoint);

    if (bpOrder == 1)
        cout << "Invalid point order. No base point found." << endl;
    else
    {
        cout << "Results Using E" << P << "(" << A << "," << B << "):" << endl;
        cout << "Base Point: (" << basePoint.x << "," << basePoint.y << ")" <<
        endl;
        cout << "Base Point Order: " << bpOrder << endl;
        cout << "a' = " << invA << endl;
        cout << "a^2/b^2 = " << A2B2 << endl;
    }
    cout << "Please enter 'q' when finished reading results." << endl;
    while (in != 'q' && in != 'Q')
    {
        cin >> in;
    }
    return 0;
}

//Using Extended Euclidean Algorithm
int FindInverse(int x)
{
    int quotient = 10, coefficients[2], remainder[2], temp;

    remainder[0] = P;
    remainder[1] = x;
    coefficients[0] = 0;
    coefficients[1] = 1;
    while (remainder[1] > 0)
    {
        quotient = remainder[0] / remainder[1];
        temp = remainder[0] - quotient*remainder[1];
        remainder[0] = remainder[1];
        remainder[1] = temp;
        if (remainder[1] > 0) {
            temp = coefficients[0] - coefficients[1] * quotient;
            coefficients[0] = coefficients[1];
            coefficients[1] = temp;
        }
    }
    while (coefficients[1] < 0)
        coefficients[1] = coefficients[1] + P;
    while (coefficients[1] > P)
        coefficients[1] = coefficients[1] - P;
    return coefficients[1];
}

//Simply recording all squares up to P/2.
//After P/2 the squares repeat
void FindResidues(Square quadraticResidues[])
{
    int square;
    for (int i = 0; i < numberOfResidues; i++)
    {
        square = (i * i) % P;
        while (square > P)
            square = square - P;
        while (square < 0)
            square = square + P;
        quadraticResidues[i].square = square;
        quadraticResidues[i].root = i;
    }
    SortSquares(quadraticResidues, 0, numberOfResidues-1);
}

//Using Binary Search
int SearchSquares(int square, Square qr[], int low, int high)
{
    int mid = (low + high) / 2;
    if (qr[mid].square == square)
        return qr[mid].root;
    if (low >= high)
        return -1;
    if (qr[mid].square < square)
        return SearchSquares(square, qr, mid+1, high);
    else
        return SearchSquares(square, qr, low, mid-1);
}

//Using Quicksort
void SortSquares(Square qr[], int low, int high)
{
    int pivot = low;
    if (high <= low)
        return;
    for (int i = low + 1; i <= high; i++)
    {
        if (qr[pivot].square > qr[i].square)
            if (pivot + 1 == i)
                SwapSquares(&qr[i], &qr[pivot]);
            else
            {
                SwapSquares(&qr[i], &qr[pivot + 1]);
                SwapSquares(&qr[pivot + 1], &qr[pivot]);
            }
    }
    SortSquares(qr, low, pivot - 1);
    SortSquares(qr, pivot + 1, high);
}

//Swaps two Square structs for quicksort
void SwapSquares(Square* first, Square* second)
{
    Square temp;

    temp.square = first->square;
    temp.root = first->root;
    first->square = second->square;
    first->root = second->root;
    second->square = temp.square;
    second->root = temp.root;
}

//Adds two Points
void Addition(Point point1, Point point2, int invA, int invB, Point* result)
{
    Point point3;
    int A2B2 = (A*A % P)*invB*invB % P;

    point3.x = invA*(point1.x*point2.x - A2B2*point1.y*point2.y) % P;
    point3.y = invA*(point1.x*point2.y + point2.x*point1.y) % P;
    while (point3.x > P)
        point3.x = point3.x - P;
    while (point3.x < 0)
        point3.x = point3.x + P;
    while (point3.y > P)
        point3.y = point3.y - P;
    while (point3.y < 0)
        point3.y = point3.y + P;
    result->x = point3.x;
    result->y = point3.y;
```

```
}

//It goes through possible points in the group, using the quadratic residues
to determine
//whether a point exists. Once a point is discovered, it checks the order to
see if it meets
//the passed criteria. A point with at least the minimum order is returned, or
the identity is
//returned if no such point is found.
int FindBasePoint(Square qr[], int minOrderSize, int invA, int invB, Point*
basePoint)
{
    Point bPoint, identity, sum;
    int order, B2 = B*B % P, y2, residue;
    int B2A2 = B2*invA*invA % P;

    identity.x = A;
    identity.y = 0;

    for (int i = 0; i < P; i++)
    {
        y2 = B2 - B2A2 * i * i % P;
        while (y2 < 0)
            y2 = y2 + P;
        while (y2 > P)
            y2 = y2 - P;
        residue = SearchSquares(y2, qr, 0, numberOfResidues-1);
        if (residue >= 0)
        {
            bPoint.x = i;
            bPoint.y = residue;
            sum.x = i;
            sum.y = residue;
            order = 0;
            while (!(sum.x == identity.x && sum.y == identity.y))
            {
                Addition(sum, bPoint, invA, invB, &sum);
                order++;
            }
            order++;
            if (order >= minOrderSize)
            {
                basePoint->x = bPoint.x;
                basePoint->y = bPoint.y;
                return order;
            }
        }
    }
    basePoint->x = identity.x;
    basePoint->y = identity.y;
    return 1;
}
```

```
//A's program to encrypt a message
#include <iostream>
#include <fstream>;

struct Point {
    int x = 0;
    int y = 0;
};

void Addition(Point, Point, int, int, Point*);

const int P = 127, A = 50, B = 103;

using namespace std;
int main() {
    ifstream message;
    ofstream code;
    int invA, A2B2, privateKey, bpOrder;
    Point encryptKey, publicKeyA, publicKeyB, basePoint, m;
    char in = 'a';

    //A chooses a private key less than the order of the base point.
    privateKey = 19;

    //This information was given by the previous program.
    basePoint.x = 4;
    basePoint.y = 20;
    bpOrder = 128;
    invA = 94;
    A2B2 = 104;

    //B shares this public key
    publicKeyB.x = 110;
    publicKeyB.y = 8;

    //Calculating A's Public Key
    publicKeyA.x = basePoint.x;
    publicKeyA.y = basePoint.y;
    for (int i = 1; i < privateKey; i++)
        Addition(basePoint, publicKeyA, invA, A2B2, &publicKeyA);

    //Calculating the shared encryption key
    encryptKey.x = publicKeyB.x;
    encryptKey.y = publicKeyB.y;
    for (int i = 1; i < privateKey; i++)
        Addition(encryptKey, publicKeyB, invA, A2B2, &encryptKey);

    //Encrypting
    message.open("TestMessage.txt", std::ios::binary | std::ios::in);
    code.open("TestCrypt.txt", std::ios::binary | std::ios::out);
    if (!message || !code)
    {
        cout << "Invalid File" << endl;
        return 0;
    }
    while (!message.eof())
    {
        message.read((char *)&in, sizeof(char));
        m.x = in;
        message.read((char *)&in, sizeof(char));
        m.y = in;
        Addition(m, encryptKey, invA, A2B2, &m);
        code.write((char *)&m.x, sizeof(char));
        code.write((char *)&m.y, sizeof(char));
    }
    message.close();
    code.close();
    return 0;
}

//Adds two Points
void Addition(Point point1, Point point2, int invA, int A2B2, Point* result)
{
    Point point3;

    point3.x = invA*(point1.x*point2.x - A2B2*point1.y*point2.y) % P;
    point3.y = invA*(point1.x*point2.y + point2.x*point1.y) % P;
    while (point3.x > P)
        point3.x = point3.x - P;
    while (point3.x < 0)
        point3.x = point3.x + P;
    while (point3.y > P)
        point3.y = point3.y - P;
    while (point3.y < 0)
        point3.y = point3.y + P;
    result->x = point3.x;
    result->y = point3.y;
}
```

```
//B's program to decrypt A's message
#include <iostream>
#include <fstream>;

struct Point {
    int x = 0;
    int y = 0;
};

void Addition(Point, Point, int, int, Point*);

const int P = 127, A = 50, B = 103;

using namespace std;
int main() {
    ofstream message;
    ifstream code;
    int invA, A2B2, privateKey, bpOrder;
    Point encryptKey, publicKeyA, publicKeyB, basePoint, m, decryptKey;
    char in;

    //B chooses a private key less than the order of the base point.
    privateKey = 87;

    //This point was picked by the previous program.
    basePoint.x = 4;
    basePoint.y = 20;
    bpOrder = 128;
    invA = 94;
    A2B2 = 104;

    //A shares this public key
    publicKeyA.x = 91;
    publicKeyA.y = 6;

    //Calculating B's Public Key
    publicKeyB.x = basePoint.x;
    publicKeyB.y = basePoint.y;
    for (int i = 1; i < privateKey; i++)
        Addition(basePoint, publicKeyB, invA, A2B2, &publicKeyB);

    //Calculating the shared encryption key
    encryptKey.x = publicKeyA.x;
    encryptKey.y = publicKeyA.y;
    for (int i = 1; i < privateKey; i++)
        Addition(encryptKey, publicKeyA, invA, A2B2, &encryptKey);

    //Calculating the decryption key
    decryptKey.x = encryptKey.x;
    decryptKey.y = P - encryptKey.y;

    //Decrypting
    message.open("TestDecrypt.txt", std::ios::binary | std::ios::out);
    code.open("TestCrypt.txt", std::ios::binary | std::ios::in);
    if (!message || !code)
    {
        cout << "Invalid File" << endl;
        return 0;
    }
    while (!code.eof())
    {
        code.read((char *)&in, sizeof(char));
        m.x = in;
        code.read((char *)&in, sizeof(char));
        m.y = in;
        Addition(m, decryptKey, invA, A2B2, &m);
        message.write((char *)&m.x, sizeof(char));
        message.write((char *)&m.y, sizeof(char));
    }
    message.close();
    code.close();
    return 0;
}

//Adds two Points
void Addition(Point point1, Point point2, int invA, int A2B2, Point* result)
{
    Point point3;

    point3.x = invA*(point1.x*point2.x - A2B2*point1.y*point2.y) % P;
    point3.y = invA*(point1.x*point2.y + point2.x*point1.y) % P;
    while (point3.x > P)
        point3.x = point3.x - P;
    while (point3.x < 0)
        point3.x = point3.x + P;
    while (point3.y > P)
        point3.y = point3.y - P;
    while (point3.y < 0)
        point3.y = point3.y + P;
    result->x = point3.x;
    result->y = point3.y;
}
```

Bibliography

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. IT-31, July 1985.

[3] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comp.*, vol. 48, pp. 203–209, 1987.

[4] E. A. Yfantis, "A new cyclic cryptographic algorithm," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 20–25, Jan 2018.

[5] A. Gehani, T. LaBean, and J. Reif, *DNA-based Cryptography*, pp. 167–188. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[6] W.-L. Chang, S.-C. Huang, K. W. Lin, and M. S.-H. Ho, "Fast parallel dna-based algorithms for molecular computation: discrete logarithm," *The Journal of Supercomputing*, vol. 56, pp. 129–163, May 2011.

[7] P. W. Shor, "Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Sci. Statist. Comput.*, vol. 26, p. 1484, 1997.

[8] J. Proos and C. Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves," *QIC*, no. 4, pp. 317 – 344, 2003.

[9] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography.," *Journal of the ACM*, vol. 56, no. 6, pp. 34 – 34:40, 2009.

[10] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology CRYPTO 85*, vol. 218 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg), Springer, 1986.

[11] J. M. Merolla, Y. Mazurenko, J. P. Goedgebuer, H. Porte, and W. T. Rhodes, "Phase-modulation Transmission System for Quantum Cryptography," *Optics Letters*, vol. 24, no. 2, pp. 104–106, 1999.

[12] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[13] A. J. Menezes and S. A. Vanstone, "A Note on Cyclic Groups, Finite Fields, and the Discrete Logarithm Problem," *Applicable Algebra in Engineering, Communication and Computing*, vol. 3, no. 1, pp. 67–74, 1992.

[14] D. M. Gordon, "Discrete Logarithms in GF (P) Using the Number Field Sieve," *SIAM Journal on Discrete Mathematics*, vol. 6, no. 1, pp. 124–138, 1993.

[15] A. Commeine and I. Semaev, "An algorithm to solve the discrete logarithm problem with the number field sieve," in *Public Key Cryptography - PKC 2006*, vol. 218 of *Lecture Notes in Computer Science*, 2006.

[16] C. Pomerance, "A Tale of Two Sieves," *Not. Amer. Math. Soc.*, vol. 43, pp. 1473–1485, 1996.

[17] A. K. Lenstra, "Unbelievable security matching aes security using public key systems," in *Advances in Cryptology ASIACRYPT 2001*, Lecture Notes in Computer Science, pp. 67–86, 2001.

[18] R. Barbulesu, P. Gaudry, A. Guillevic, and F. Morain, "Improving nfs for the discrete logarithm problem in non-prime finite fields," in *Advances in Cryptology – EUROCRYPT 2015*, Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 129–155, Springer Berlin Heidelberg, 2015.

[19] A. Joux, R. Lercier, N. Smart, and F. Vercauteren, "The number field sieve in the medium prime case," in *Advances in Cryptology - CRYPTO 2006*, Lecture Notes in Computer Science, p. 326344, 2006.

[20] P. Sarkar and S. Singh, "New complexity trade-offs for the (multiple) number field sieve algorithm in non-prime fields," in *Advances in Cryptology EUROCRYPT 2016*, Lecture Notes in Computer Science, p. 429458, 2016.

[21] T. Kim and J. Jeong, "Extended tower number field sieve with application to finite fields of arbitrary composite extension degree," in *Public-Key Cryptography PKC 2017*, Lecture Notes in Computer Science, p. 388408, 2017.

[22] A. Joux, "A new index calculus algorithm with complexity l(1/4 o(1)) in small characteristic," in *Selected Areas in Cryptography – SAC 2013*, Lecture Notes in Computer Science, p. 355379, 2014.

[23] A. K. Lenstra and E. R. Verheul, "An overview of the xtr public key system," pp. 1–20, 2001.

[24] J. Hoffstein, J. Piper, and J. H. Silverman, *An Introduction to Mathematical Cryptography*, ch. 5, pp. 279–349. New York, NY: Springer, 2014.

[25] E. Barker and Q. Dang, *Recommendation for Key Management Part 3: Application-Specific Key Management Guidance*. NIST, Jan. 2015.

Curriculum Vitae

Graduate College

University of Nevada, Las Vegas

Elizabeth Dettrey

Degrees:

Bachelor of Science in Computer Science and Mathematics 2016

University of Nevada Las Vegas

Thesis Title: Elliptic Cryptosystem

Thesis Examination Committee:

Chairperson, Dr. Evangelos Yfantis, Ph.D.

Committee Member, Dr. Hal Berghel, Ph.D.

Committee Member, Dr. Andreas Stefik, Ph.D.

Graduate Faculty Representative, Dr. Sarah Harris, Ph.D.