Hyperelliptic Curve Cryptosystem

A dissertation submitted in partial fulfillment
of the requirements of M.Tech.(Computer Science)
degree of Indian Statistical Institute, Kolkata
by

Pradeep Kumar Mishra

under the supervision of

Dr. Rana Barua
Stat-Math Unit

Indian Statistical Institute
203, Barrackpore Trunk Road
Kolkata-700 108.

25th July 2002

Acknowledgements

I am profoundly grateful to my supervisor Prof Rana Barua,Stat-Math Unit, ISI, who has given me ample scope to make use of his expertise and experience while I was working for this dissertation. Particularly I am thankful to him for keeping his door open for me whenever I found some stumbling blocks on my way. His busy schedule has never come in my way of getting my doubts cleared. Also I am thankful to Dr Palash Sarkar, Applied Statistics Unit, ISI who has immensely helped me in my works. His pleasant and encouraging words have always kept my spirits up. Also, I can never forget the inspirations I have received from Prof Bimal Roy,Applied Statistics Unit,ISI,Kolkata.

I am greatly indebted to all my family members who have always supported my endeavours. Particulary I am greatly indebted to my wife Jyotsna and son Siddharth who whole-heartedly co-operated and supported my self-exile to ISI,Kolkata to pursue this course.

Also I am thankful to all my classmates and friends(whose list is long) who cooperated in all possible ways and stood by me in hard times. Also I must thank them for all the fun and frolic we had togather in last couple of years.

P K Mishra

Indian Statistical Institute

203, Barrackpore Trunk Road,

Kolkata-700 108.

Certificate of Approval

This is to certify that this thesis titled "**Hyperelliptic Curve Cryptosystem**" submitted by **Pradeep Kumar Mishra** towards partial fulfillment of requirements for the degree of M. Tech in Computer Science at Indian Statistical Institute, Kolkata embodies the work done under my supervision.

[illegible]

(*Rana Barua*)
Professor, Stat-Math Unit,
Indian Statistical Institute,
Kolkata-700 108.

To my father,
I could not see his last days in this world
because of this course.

Abstract

The Internet has become the most convenient, affordable, widely connected and accpted medium for data communication now. While billions of users worldwide are sharing the same network, sensitive information must be adequately protected from 'prying eyes'.

With the discovery of Diffie-Hellman key-exchange protocol in 1976, the Public Key Cryptosystems (PKC's) have almost taken over the charge of providing the much desired security.The search for newer and newer PKC's and their cryptanalysis has become a part of the mainstream of computer scientific research. The reason for search of newer cryptosystems is twofold: firstly,to increase the ease at which the task of encryption and decryption can be carried out and secondly, to reduce the computational overhead involved.

In PKC's the breaking of the cryptosystem is the getting hold of one's secret private key with feasible amount of computations. Naturally, the larger the private key, the more secure is the cryptosystem. However the length of the private key is a computational overhead and a smaller key length means more efficiency.

In this trade-off between efficiency and level of security, the hyperelliptic curve cryptosystem (HECC) is a natural winner.It provides the same level of security as many a existing PKC's with much smaller key-length. However,HECC is yet to establish itself as a feasible alternative to existing popular PKC's due to the inherent complexity in the inplementation of certain operations involved.

In this thesis, we examine different aspects of implementation of a HECC.

Contents

Chapter 1

INTRODUCTION

1.1 What is Cryptology and Why ?

"Knowledge is Power" - is the classic adage most appropriate at the moment and knowledge is 'cooked data'.Computer has now become the store-house of all kinds of data and the Internet is the Information Superhighway for its communication from person to person.Communication of financial or any sensitive data is required to be kept beyond the reach of unscrupulous and unintended receivers. Exactly that is where Cryptology comes into the modern day computer application scenario.

The term Cryptology is derived from two Greek words "kryptos" meaning hidden and "logos" meaning knowledge. So cryptology is the science of secret writing and its analysis. Cryptology has two domains of studies: cryptography and cryptanalysis. Cryptography is "kryptos" +"graphein"(writing). Thus, cryptography is the art of secret writing i.e. the art of ensuring that messages are secure from those recipients to whom it is not addressed. Cryptanalysis is the art of devising methods for getting back the original message from the encrypted messages and analyzing the performance and efficiency of such methods. Significance of cryptography has increased manifolds in this era of the Internet. Now a days it is not only restricted to military use and statecraft, but has percolated to the masses, who use on-line banking, internet stockbroking and e-commerce in general. Cryptology has received much attention since the rise in the use of electronic communication for the exchange of financial information and has been mainly driven by the increased demands for secure transactions over the Internet.

1.2 Cryptography : Block Ciphers and Stream Ciphers

In modern cryptography, ciphers can be broadly divided into :

1. Block ciphers
2. Stream ciphers.

In block ciphers the whole message (plain text) is divided into several blocks of fixed size and each block is encrypted in the same way to yield the ciphertext. The substitution cipher, as described in the next section is an example of block cipher, where a block consists of one letter of the plain text. In stream cipher the whole message is treated as a bit stream. A key bit stream is generated using a specific hardware and each plain text bit is XORed with the corresponding bit of the key bit stream to generate a bit of the cipher text. The receiver, possessing the same hardware generates the same key bit stream and XORs it with the received encrypted message to get the original plaintext. Stream ciphers are useful in transmitting voluminous data like voice mail.

1.3 Cryptography : From Julius Caesar To Neil Koblitz

1.3.1 Private key Cryptography :

Use of cryptography dates back to the period of Romans Empire.History notes down that Julius Caesar was first to use substitution alphabet ciphers, where the letters of the original message (called plain text in the cryptography literature) are substituted by letters at a fixed distance in the alphabet to yield the encrypted message (called the ciphertext). This fixed distance is a secret key. One with the knowledge of the secret key can easily decipher the encrypted message by shifting back each of its letters exactly as much distance in the alphabet as the secret key. Substitution cipher is one example of what is known as secret key cryptosystems in cryptographic literature.

History also says that Kautilya has written about ciphers in his treatise on statecraft ***Arthashastra***. So cryptography was also in use in India at least during the Mauryan period.

In a secret key cryptosystem, first of all, the secret key needs to be communicated to the intended recipient through a secure channel or a trusted courier so that the receiver can decrypt the sent message.

Substitution cipher is not a secure method of encryption. An adversary can resort to exhaustive search method to decipher a message encrypted using substitution cipher. In fact, continual advancement in computer technology, particularly increase in the speed of computation has rendered many a earlier cryptosystems useless, where sometimes, even a naive exhaustive search method to break the encryption becomes feasible.

As newer and newer attacks are discovered to break the existing cryptosystems, the modern day cryptographic techniques have become more and more sophisticated. Several new avenues, which were earlier pursued by mathematicians and researchers for purely aesthetic reasons, have been explored and have been successfully employed in implementing newer cryptosystems to keep "the peeping eyes" at bay. For example block ciphers now employ elements of finite group (which as sophisticated as points of an elliptic curve or divisors of an hyperelliptic curve) to encrypt messages, where the key length is of the order of 160 to 1024 bits. Also, this motivates for the study of other cryptosystems in the hope to increase the transmission efficiency and to make the cryptosystems implementable in resource constrained machines by decreasing the number of bits required for the key while maintaining the same level of security.

1.3.2 Public key Cryptography

One drawback of the secret key cryptosystems was that the secret key is to be communicated to the intended receiver through a secure channel, before any communication can actually begin. Till seventies all the cryptosystems were private key cryptosystems. The secure channel was not available always or was too expensive for the common man. This difficulty lead to the discovery of public key cryptosystems. In 1976, the idea of public key cryptosystem(PKC) was proposed by Diffie and Hellman in their famous paper [1].This added an entirely new dimension to cryptography and later became the mainstream of it. Public key cryptography, not only obviated the need of sending a secret key to the receiver, but also put the method of encryption itself above board.

In public key cryptosystems, each of the users is required to publish his/her public key in directory accessible to everybody. Also, the whole infrastructure and method used for encryption is published. Each user also possesses a private key. A message addressed to a user can be accessed only if one has the knowledge of the user's private key. Needless to mention, a user generally does not share his/her private key with anyone else. At the heart of PKC's lies a special kind of functions, called trapdoor oneway functions. Oneway functions are the ones whose values are easy to compute. But given a value of the function it is computationally infeasible to compute its inverse without the knowledge of a special key value, called the trapdoor. Number theory says that given two large primes p and q, it is easy to compute $n = pq$. But given n, it is computationally very difficult to find p or q. This hard problem is called the integer factorization problem. In a cyclic group of large order with base α,given p, it is very simple to find $f(p) = \alpha^p$. But, given $\beta = f(p)$, it is very difficult to find p. This function f is another example of oneway functions.The problem of inverting f is called the discreet logarithm problem. The Diffie-Hellman key exchange protocol and so many other PKC's considered secure now-a-days are based on discreet logarithm problem.

Since the discovery of Diffie-Hellman key exchange protocol, several public key cryptosystems have established themselves commercially. The Knapsack systems were the simplest ones, whose security rested on the NP-completeness of the knapsack problem. However, in 1986, Shamir showed how knapsack problem can be effectively broken and then it became a part of the history. Two other most popular PKC's are RSA and ElGamal cryptosystems. The RSA cryptosystems discovered by Rivest, Shamir and Addleman derives its strength from the hardness of the integer factorization problem, while, the ElGamal cryptosystems depends on the hardness of the discrete logarithm problem.

In 1987, Neal Koblitz[2] and Victor Miller [13] independently discovered the elliptic curve cryptosystem. Although elliptic curves were extensively studied for over a hundred of years prior to that and there was a vast literature on them, it was done purely for aesthetic reasons. The genius of Koblitz and Miller could see that they could be efficiently used for creating public key cryptosystems. Elliptic curve cryptosystems could achieve the same level of security as RSA with much smaller key-length.

Hyperelliptic curves, however, did not receive that much of attention as elliptic curves, from the research community. There were very few results specific to the hyperelliptic curves and most of the results were general in nature to algebraic curves. In 1989, Koblitz in his epoch-making paper [3], demonstrated how the divisor class of hyperlliptic curves can serve as a rich source of finite abelian groups suitable for implementing public key cryptosystems. Since then, a lot of work is going on to explore hyperelliptic curve cryptosystems, on their performance, efficiency and on cryptanalytic attacks which can be launched on them.

In this work we have implemented a hyperelliptic curve cryptosystem over prime fields.

Chapter 2

AN INTRODUCTION TO HYPERELLIPTIC CURVES

Hyperelliptic curves are a generalization of elliptic curves, both of which are classified as algebraic curves. The elliptic curves are hyperelliptic curves of genus g = 1. Hyperelliptic curves are of higher genus $g \geq 1$.The fact that these curves can be used efficiently for message encryption and decryption was invented in late eighties only. Prior to that, the elliptic curves had fascinated mathematicians and researchers and a vast literature existed on them. However any treatise on hyperelliptic curves was of very general in nature, even the underlying field being the field of complex numbers or an algebraically closed field. After discovery of hyperelliptic curve cryptography by Neil Koblitz in 1989, these curves have begun to draw attentions from cryptologists and researcher working on similar areas. Recently, the hyperelliptic curves have found applications in several other areas of research activities like primality proving, design of error correcting codes and integer factorization problem.In this chapter we give a brief introduction to the theory of hyperelliptic curves.

2.1 Definition and Elementary Concepts

Definition 2.1 Let F be a field and let K be the algebraic closure of F. A hyperelliptic curve C of genus g ($g \geq 1$)over F is an equation of the form

$$C : v^2 + h(u)v = f(u) \tag{2.1}$$

where $h(u)$ in F[u] is a polynomial of degree at most g, $f(u)$ in F[u] is a monic polynomial of degree 2g + 1, and there are no solutions (u, v) in K x K, which simultaneously satisfy the equations

$$v^2 + h(u)v = f(u) \tag{2.2}$$
$$C : 2v + h(u) = 0 \tag{2.3}$$
$$h'(u)v - f'(u) = 0. \tag{2.4}$$

A singular point on C is a solution (u, v) $\in$ K x K which simultaneously satisfies the equations 2.2, 2.3 and 2.4. Thus, a hyperelliptic curve, by definition, does not have any singular points.

It is simple to prove that :

Proposition 2.2 For a hyperelliptic curve C over F,

- If the polynomial h(u) is identically 0, then characteristic of F is not 2.
- If characteristic of K is not 2, then the change of variables $u \to u, v \to (v - h(u)/2)$ transforms C to the form v^2 = f(u) where $deg_u f = 2g + 1$.
- Let C be an equation of the form (1) with h(u) = 0 and characteristic of K is not 2. Then C is a hyperelliptic curve if and only if f(u) has no repeated roots in K.

Definition 2.3 Let L be an extension field of F. The set of L−rational points on C, denoted C(L), is the set

$$\{(x, y) \in L \times L : y^2 + h(x)y = f(x)\} \cup \{\mathcal{O}\}$$

where $\mathcal{O}$ is a special point, called the point at infinity . The set of points C(F) will simply be denoted by C. The points in C other than $\mathcal{O}$ are called finite points. The opposite point of a finite point P(x, y)on the curve C is defined to be the point $\tilde{P}$(x, -y - h(x)). (Note that $\tilde{P}$ is indeed on C). We also define the opposite of $\mathcal{O}$ is to be itself. If a finite point P satisfies P = $\tilde{P}$ then the point is said to be a special point; otherwise, the point P is said to be ordinary.

In Cryptography, we are particularly interested in Hyperelliptic curves over finite fields.

2.2 The field of rational functions on C

Definition 2.4 Let I be the ideal of F[u, v] generated by the polynomial $v^2 + h(u)v - f(u)$ i.e. I = $(v^2 + h(u)v - f(u))$.The quotient ring of F[u,v] /I is called the coordinate ring of C over F, denoted by F[C]. Similarly, the coordinate ring of C over K is defined as

$$K[C] = K[u, v]/(v^2 + h(u)v - f(u)).$$

An element of K[C] is called a polynomial function on C.

Since the polynomial $p(u, v) = v^2 + h(u)v - f(u)$is irreducible over K, K[C], is an integral domain. Further, we observe that, since the polynomial p(u, v) is of degree 2 in v and K[C] contains polynomials modulo p(u, v), degree of v in any polynomial function G(u, v) in K[C] can be at most one. In other words, every polynomial G(u, v) in K[u, v] has a unique representation of the form $G(u, v) = a(u) - b(u)v$, for some polynomials a(u), b(u) in K[u].

Next, we define the conjugate and norm of a polynomial function. These concepts are similar to that of conjugate and modulus of a complex number and enjoy similar properties.

Definition 2.5 For a polynomial function G(u, v) = a(u) - b(u)v be in K[C],the conjugate of G(u, v) is defined to be the polynomial function $\overline{G}(u, v) = a(u) + b(u)(h(u) + v)$.The norm of G, denoted by

N(G), is defined to be G.$\overline{G}$ i.e. N(G) = G$\overline{G}$.

The next proposition follows directly from the definition of norm.

Proposition 2.6 Let G, H ϵ K[C] be polynomial functions.

- N(G) is a polynomial in K [u].
- N(G) = N($\overline{G}$).
- N(GH) = N(G)N(H).

We define the degree of a polynomial function in K[C] as follows; let degree of u be 2 and that of v be 2g + 1. It sounds fair considering the powers of u and v occurring in the equation of a hyperelliptic curve. Then obviously,

Definition 2.8. Let G(u, v) = a(u) - b(u)v be a non-zero polynomial function in K[C]. The degree of G is defined as

$$deg(G) = max\{2deg_u a(u), 2deg_u b(u) + 2g + 1)\}.$$

The next proposition is a routine one following this definition.

Proposition 2.9. Let G, H be in K[C].

- $deg(G) = deg_u(N(G))$.
- $deg(G) = deg(\overline{G})$ where $\overline{G}$ is the conjugate of G.
- deg(GH) = deg(G) + deg(H).

The rational functions on a hyperelliptic curve play a vital role in the hyperelliptic curve cryptosystems. Next, we define them.

Definition 2.7. The function field F(C) (resp K(C)) of C over F (resp K) is the field of all fractions of polynomial functions in F[C] (resp K[C]). An element of K(C) is called a rational function on C. A polynomial function is also a rational function. That is so as K[C] is a subring of K(C).

The value of a rational function at infinity is defined in a similar fashion as real rational functions are defined at infinity in the extended real number system. We define,

Definition 2.8. Let R = G/H be a rational function on C. The value of R at ∞

- is 0 if $deg(G) < deg(H)$.
- is ∞ if $deg(G) > deg(H)$.
- is the ratio of leading coefficients (with respect to the degree function) of G and H if deg(G) = deg(H).

We define order of a rational function R to be non-zero at the points where it vanishes or where it becomes infinity. Orders of R at these points determine what is called the divisor of R. The divisors of rational functions play a important role in hyperelliptic curve cryptography. Before going that far, we define:

Definition 2.9. Let $R \in K(C)$, be a rational function on C and let $P \in C$, be a point on C. The point P is said to be a zero of R if R(P) = 0 and P is a pole of R if $R(P) = \infty$.

The order of a polynomial function at a point is defined in terms of a rational function, called the uniformising parameter of the related point. The next theorem defines and establishes the existence uniformising parameter.

Theorem 2.10. Let P ∈ C. A rational function U ∈ K(C) with U(P) = 0 is said to be a ***uniformising parameter*** for P if the following condition holds: let G ∈ K[C]* be any non-zero polynomial function on C. Then there exists an integer d and a function S ∈ K(C) which neither has a pole nor a zero at P and $G = U^d S$. Moreover, the value of d is independent of the choice of U.

The existence of uniformising parameter can be proved by construction. If P(x,y) is an ordinary point then it can be seen that U = (u - x) can be a uniformising parameter. For a special point U = (v - y) perfectly fits the bill. If P = ∞ then $U = (u^g)/v$ does the same. A detailed and elegant proof of the theorem can be found in [4].

Uniformising parameters are used to define the order of rational functions.

Definition 2.11. Let G ∈ K[C]* be a polynomial over C. The order of G at P is defined to be d, if $G = U^d S$, where U is a uniformising parameter of P and S is a rational function such that S(P) ≠ 0 nor S(P)≠ ∞ . If R = G/H then order of R at P is denoted by $ord_P(R)$ and is defined to be $ord_P(R) = ord_P(G) - ord_P(H)$.

The order of a rational function satisfies certain important properties. We note down them below as a proposition for future reference.

Proposition 2.12. Let R_1, R_2 be non-zero rational functions. Then
(i) $ord_P(R_1 R_2) = ord_P(R_1) + ord_P(R_2)$.
(ii) Let $ord_P(R_1) = o_1, ord_P(R_2) = o_2$ and $R_1 \neq -R_2$. If $o_1 \neq o_2$ then

$$ord_P(R_1 + R_2) = min\{o_1, o_2\}.$$

Otherwise,

$$ord_P(R_1 + R_2) = min\{o_1, o_2\}$$

(iii) Any non-zero rational function R has a finite number of zeroes and poles. Moreover, $\sum_{P \in C} ord_P(R) = 0$.
(iv) If R is a non-zero polynomial function then $ord_P(R) = ord_{\tilde{P}}(\overline{R})$, where $\overline{R}$ the conjugate of R and $\tilde{P}$ is the opposite point of P.

2.3 Divisors

The most important and somewhat complicated concept in hyperelliptic curve cryptography is that of divisors, which come into picture because, unlike the points on an elliptic curve, the points of an hyperelliptic curve do not form a group. So to implement a cryptosystem like the one like ElGamal, at the heart of which lies a finite cyclic group of suitable order, we take a subgroup of the free abelian group generated by the set of points on an hyperelliptic curve C, each element of which is a finite formal sum of points on C with integer coefficients. Each of these formal sums is called a divisor. The set of divisors form a group under an additive operation, defined pointwise. More formally,

Definition 2.13.A divisor D is a formal sum like,

$$D = \sum_{P \in C} m_P P, \qquad m_P \in Z \text{ and } P \in C$$

where Z is the set of integers and only finitely many of m_P's are non zero. The degree of D is the integer

$$\sum_{P \in C} m_P P$$

and order of D at P is, $ord_P(D) = m_P$. Let $\mathcal{D}$ stand for the set of all divisors. For $D_1 = \sum_{P \in C} m_P P$ and $D_2 = \sum_{P \in C} n_P P$ in $\mathcal{D}$, we define,

$$D_1 + D_2 = \sum_{P \in C} (m_P + n_P) P.$$

It is mere mathematical common sense to see that ($\mathcal{D}$, +) is a group. Also, the set $\mathcal{D}_0$ of all divisors of degree 0 forms a subgroup of $\mathcal{D}$.

Also we define the greatest common divisor (gcd) of two divisors $D_1 = \sum_{P \in C} m_P P$ and $D_2 = \sum_{P \in C} n_P P$ in $\mathcal{D}$ as

$$gcd(D_1, D_2) = \sum_{P \in C} min(m_P, n_P) P.$$

A rational function has an order at each point on the curve, which is non-zero only at a finite number of points. Moreover, by proposition 2.12(iii), the sum of orders of a rational function over all points on C is zero. Hence, given a rational function R, we can define a divisor $D = \sum_{P \in C} m_P P$, where $m_P = ord_P(R)$. By proposition 2.12(iii), D $\in \mathcal{D}_0$. Such a divisor D is called the divisor of the rational function R and it belongs to an important class of divisors, called the principal divisors. Put more succinctly,

Definition 2.14. Let $R \in K(C)_*$. The divisor of R is

$$div(R) = \sum_{P \in C} ord_P(R) P.$$

A divisor D is called a principal divisor if D = div(R) for some rational function R. The set of all principal divisors $\mathcal{P}$ is a subgroup of $\mathcal{D}_l$. If $D_1, D_2 \in \mathcal{D}_l$ then D_1 and D_2 are said to be equivalent, written as $D_1 \sim D_2$, if $D_1 - D_2 \in \mathcal{P}$. It is easy to see that, $\sim$ is an equivalence relation on the set $\mathcal{D}_l$. The set $\mathcal{J}$ of all equivalence classes is called the jacobian of the curve C. In other words, $\mathcal{J}$ is the quotient group $\mathcal{D}_l/\mathcal{P}$.

The following proposition on divisors of rational function is a simple consequence of properties of order.

Proposition 2.15. Let $R = G/H \in K(C)^*$ then
$$\text{div(R)} = \text{div(G)} - \text{div(H)}.$$
Also if $R, S \in K(C)^*$ then div(RS) = div(R) + div(S).

Definition 2.16. For the divisor $D = \sum_{P \in C} m_P P$, the support of D is the set,
$$\text{Supp(D)} = \{P \in C : m_P \neq 0\}.$$

Among the divisors, two special types of divisors, semi-reduced and reduced divisors are of much importance to us. Next, we define them.

Definition 2.17. A semi-reduced divisor D is a divisor of the form $\sum m_i P_i - (\sum m_i)$, where,

- each $m_i \geq 0$
- P_i's are finite points such that when $P_i \in supp(D)$ then $P_i \notin supp(D)$
- if $P_i = \tilde{P}_i$ then $m_i = 1$.

It can be proved that, for every divisor of degree 0, there corresponds an equivalent semi-reduced divisor. In fact, truth is stronger than what this statement says. Truth is, to every divisor D there corresponds a semi-reduced divisor D' = $\sum m_i P_i - (\sum m_i)$ such that $\sum m_i < g$, where g is the genus of the curve C. Such a divisor is called a reduced divisor. More formally,

Definition 2.18. A semi-reduced divisor D = $\sum m_i P_i - (\sum m_i)$is said to be a reduced divisor if $\sum m_i < g$.

Theorem 2.19. For each divisor $D \in \mathcal{D}_l$ there exists a unique reduced divisor D_1 such that $D \sim D_1$.

Thus each coset of the quotient group $\mathcal{J} = \mathcal{D}_l\ /\mathcal{P}$ has exactly one reduced divisor. Hence we can identify each coset with its reduced divisor.

Let C be a hyperelliptic curve of genus g defined over a finite field F, whose algebraic closure is K, and let J be the jacobian of C. If $P = (x,y) \in C$, and σ be an automorphism of K over F, then P^σ, defined as (x^σ, y^σ), is also a point on C.

A divisor $D = \sum m_P P$ is said to be defined over K if $D^\sigma = \sum m_P P^\sigma$ is equal to D for all automorphism σ of K over F.

Instead of always looking for an algebraically closed field K we may prefer to work in the field F itself. The algorithm for divisor addition requires divisors to be defined in the underlying field. So, the question is, when is a divisor defined over F ? It is not difficult to see that a principal divisor is defined over F if and only if it is the divisor of a rational function that has coefficients in F. The set $\mathcal{J}$ contains divisor classes each of which can be represented by a reduced divisor. These reduced divisors representing the a class each in $\mathcal{J}$ may not be defined over F. The algorithm for divisor addition requires the input divisors to be defined over the underlying field F. So the divisor addition can not be applied to the whole of $\mathcal{J}$.Let J_C(F) be the set of all divisor classes in $\mathcal{J}$ that have a representative that is defined over F. J_C(F) is a subgroup of $\mathcal{J}$. Each element of J_C(F) has a unique representation as a reduced divisor div(a, b), where $a, b \in K[u]$, $deg(a) \leq g, deg(b) < deg(a)$. Hence $J_C(F)$ is in fact a finite abelian group. Thus we are home! We have defined a finite abelian additive group on which we can implement an ElGamal type cryptosystem. However, we need suitable data structures for representation of reduced divisors and algorithms to efficiently implement addition of reduced divisors. Neil Koblitz, in his original paper had solved both of these problems, although he had not given proof of correctness of his algorithms. Later, Cantor[5] had proved them, but his proofs were not wholly correct. Menezes et al have given the correct proofs in [4].

The next theorem specifies an elegant way to represent semi-reduced divisors as a pair of polynomials over F.

Theorem 2.20. Let $D = \sum m_i P_i - (\sum m_i)$be a semireduced divisor, where $P_i = (x_i, y_i)$. Let $a(u) = \prod(u - x_i)^{m_i}$. Let b(u) be the unique polynomial satisfying:
(i) $degb < dega$,
(ii) $b(x_i) = y_i$ for all i for which $m_i \neq 0$,
(iii) a(u) divides $b(u)^2 + b(u)h(u) - f(u)$.
Then D = gcd(div(a(u)), div(b(u) - v)).

NOTATION: To be brief, we write: gcd(div(a(u)), div(b(u) - v)) as div(a(u), b(u) - v) or, simply as div(a, b). Thus every semi-reduced divisor can be effectively represented by two polynomial functions in F[u].

Now, we need tools to work with divisors in J_C(F). That is, we need, methods to implement addition operation of J_C(F) on its elements, which are reduced divisors. Algorithm 1 precisely does that. It takes two reduced divisors as input and adds them up and outputs the result.

Algorithm 1

Input:
Reduced divisors $D_1 = div(a_1, b_1)$ and $D_2 = div(a_2, b_2)$ both defined over K.
Output:
A semireduced divisor D = div (a, b) defined over K such that $D \sim D_1 + D_2$.

1. Use the extended Euclidean algorithm to find polynomials $d_1, e_2, e_2 \in K[u]$ where $d_1 = gcd(a_1, a_2)$ and $d_1 = e_1 a_1 + e_2 a_2$.

2. Use the extended Euclidean algorithm to find polynomials $d, c_1, c_2 \in$ K[u] where $d = gcd(d_1, b_1 + b_2 + h)$ and $d = c_1 d_2 + c_2(b_1 + b_2 + h)$.

3. Let $s_1 = c_1 e_1, s_2 = c_1 e_2, ands_3 = c_1$, so that
$d = s_1 a_1 + s_2 a_2 + s_3(b_1 + b_2 + h)$

4. Set $a = a_1a_2/d^2$ and

$$b = \frac{s_1a_1b_2 + s_2a_2b_1 + s_3(b_1b_2 + f)}{d} \pmod{a}$$

However, the output of Algorithm 1 is not necessarily a reduced divisor. It is a semireduced one. We need to convert the output of Algorithm 1 into a reduced divisor. There are two algorithms that are used for the reduction step: Gauss reduction and Lagrange reduction [6]. In the first algorithm the computation of the a_k (where a_k is the value of a in the iteration k of the algorithm) involves one multiplication and one division of high degree polynomials. Each step is independent of the previous one. However, as soon as a reduction step has been carried out, the formula for a_k can be rewritten using information from the previous step. Lagrange reduction takes advantage of this fact. Paulus and Stein published the Lagrange reduction algorithm for hyperelliptic curves over a field of odd characteristic [7]. Enge gave a generalized version for arbitrary characteristic in [6]. Algorithms 2.1 and 2.2 summarize Gauss and Lagrange reductions, respectively.

Algorithm 2.1(Gauss Reduction)

Input:
A semireduced divisor D = div (a, b) defined over F.
Output:
The (unique) reduced divisor $D_1 = div(a_1, b_1)$ such that $D_1 \sim D$.

1. Set

   $$a_1 = (f - bh - b^2)/a$$

   and

   $$b_1 = (-h - b) \pmod{a_1}$$

2. If deg $a_1 > g$ then set $a \leftarrow a_1, b \leftarrow b_1$ and go to step 1.

3. Let c be the leading coefficient of a_1, and set $a_1 \leftarrow a_1/c$.

4. Output (a_1, b_1).

Algorithm 2.1(Lagrange Reduction)

Input:
A semireduced divisor D = div (a, b) defined over F.
Output:
The (unique) reduced divisor D_1 = div (a', b') such that $D_1 \sim D$.

1. Set $a_0 = a, b_0 = b$;

2. $a_1 = (f - b_0h - b_2)/a_0$;

```
3. $-b_0 - h = q_1 a_1 + b_1$;
4. while( $deg\ a_k > g$)
        $a_k = a_{k-2} + q_{k-1}(b_{k-1} - b_{k-2})$;
        $-b_{k-1} - h = q_k + b_k$ with $deg\ b_k < deg\ a_k$;
5. output($a' \leftarrow a_k, b' \leftarrow b_k$)
```

The hyperelliptic curve cryptosystem is built on the strength of the discreet logarithm in the jacobian group of reduced divisors of the curve. We end this chapter with the statement of the discreet logarithm problem on the jacobian of an hyperelliptic curve.

Definition 2.21. The hyperelliptic discrete logarithm problem (HCDLP) takes on input a hyperelliptic curve of given genus, an element D_1 of the Jacobian, its order n , and another element D_2 in the subgroup generated by D_1 . The problem is to find an integer λ mod n such that $D_2 = \lambda D_1$.

Chapter 3

IMPLEMENTATION AND SECURITY ISSUES

3.1 Why Hyperelliptic Curve Cryptosystem ?

The discrete logarithm problem (DLP) became important to cryptographists with the invention of public-key cryptography by Diffie and Hellman in 1976 [1]. The best algorithm known for solving DLP in prime fields is the number field sieve [14] which has a subexponential expected running time: $exp((1.923 + o(1))\ (log\ p)^{1/3}(log\ log\ p)^{2/3})$. To circumvent this attack, the prime p should be chosen to be sufficiently large. As of today's computer technology, a prime p of length 1024 bits is recommended for medium-term security. For long-term security, even a larger modulus is recommended. As a consequence of this, the implementation of discrete log cryptosystems using the group Z_p is infeasible or impractical in some resource constrained computational devices like smart cards and hand-held wireless devices, such as cellphones and pagers.

Since the discovery of Public Key Cryptography in 1976, a variety of groups have been proposed for use in discrete log cryptosystems. These include:

1. the multiplicative group of a finite field of characteristic 2 or a proper subgroup of it,
2. the group of units of Z_n , n being a composite integer,
3. the group of points on an elliptic curve defined over a finite field,
4. the Jacobian of a hyperelliptic curve defined over a finite field ,
5. the class group of an imaginary quadratic number field and
6. the Jacobian of a superelliptic curve defined over a finite field .

Why are we considering so many alternative groups for the purpose? There are two primary reasons. First one is, the operation in some groups may be easier to implement in software or in hardware than the operation in other groups. Secondly, the DLP in the group may be harder than the DLP in other groups. Consequently, one could use a group G that is smaller than Z_p while maintaining the same level of security. This potentially gives rise to smaller key sizes, bandwidth savings, and faster implementations.

Among these groups, E(F_q), the group of F_q-rational points of an elliptic curve E, defined over a finite field F_q , is an attractive choice. By Hasse's Theorem, the order of the group is almost equal to q. If the largest prime factor of this order is n, then the best algorithm known for the DLP in E(F_q) (Pollard's rho attack) takes O ($\sqrt{n}$) steps; i.e., the algorithm take fully exponential time. As a result, E(F_q) where q is a 160 bit integer, can achieve the same level of security as when a group Z_p is used with a 1024 bit integer. That is almost a 84 percent saving of bandwidth !

The order of $J_C(F_q$), the Jacobian of a hyperelliptic curve C of genus g defined over F_q , is roughly q^g . Jacobian elements can be compactly represented by a pair of polynomials of degree at most g over F_q , and efficiently added using Cantor's algorithm. When g is large, there is a subexponential algorithm due to Adleman, DeMarrais and Huang [8] for the discrete logarithm problem in $J_C(F_q$). Moreover, when $g \geq 5$, Gaudry's algorithm [10] is faster than Pollard's rho algorithm. If g = 2 or g = 3, and n is the largest prime divisor of $\#J_C(F_q)$, the best algorithm known takes O($\sqrt{n}$) steps, i.e., the algorithm takes fully exponential time. Consequently, a cryptosystem based on a hyperelliptic curve of genus 2 over F_q, where q is a 80 bit integer, can achieve the same level of security as when a group Z_p is used with a 1024 bit integer.

Now, one natural question arises. Why can not we use the jacobian of any other algebraic curve for cryptographic purposes ? In fact we can use any algebraic curve, but we have to address two problems, which arises. One is finding a canonical representation of the elements of the jacobian. Another important one is to find efficient methods to add two elements of the jacobian. In case case of hyperelliptic curves both of these have been elegantly solved. An element of the jacobian i.e. a reduced divisor can be represented as a pair of polynomials and also there are efficient methods for divisor addition and reduction.

One disadvantage of using curves of higher genus instead of elliptic curves is that the group operation in the former may be computationally more expensive and more complex, even though the underlying field size is much smaller. In fact, in cryptography there is still much scope for research in that direction. Also, this disadvantage can be overcome by designing specific hardware, to be used in resource constrained systems [9].

3.2 How Secure a Hyperelliptic curve cryptosystem ?

As we have seen the hyperelliptic discrete logarithm problem (HCDLP) takes on input a hyperelliptic curve of given genus, an element D_1 of the Jacobian, its order n , and another element D_2 in the subgroup generated by D_1 . The problem is to find an integer λ mod n such that $D_2 = \lambda D_1$. Possible attacks on the HCDLP can be viewed as of three types. Some attacks are of general nature, applicable to any DLP. In the second category come the attacks that have been successfully launched on some particular elliptic curve discreet logarithm problem (ECDLP) and there is possibility of generalizing them to hyperelliptic case. Attacks specific to HCDLP can be put in the third category. We give below a list of such attacks, which by no means can be claimed to be exhaustive.

1. **Naive Exhaustive Search:** Compute D_1, $2D_1$, $3D_1$ ...until D_2 is obtained. Obviously it can take n = $O(D_1)$ steps. So taking the value of n large enough is sufficient to check this attack.

2. **Pohlig - Hellman Algorithm:** Exploits factorization of n = $O(D_1)$.The algorithm reduces the problem of recovering λ to the problem of recovering λ modulo each prime factor of λ, λ is then recovered by Chinese remainder theorem. This attack can be avoided by selecting an HC the order of whose jacobian is divisible by a large prime factor n.

3. **Baby-step Giant-step Algorithm :** This is an algorithm to attack DLP on any finite group G. If x = $log_\alpha\beta$ then x = jm + i where $m = \sqrt{n}$ and $0 \leq i < m$. Precompute a list (i,α^i) for $0 \leq i < m$ and sort this list by second component. For each j, $0 \leq j < m$, compute $\beta\alpha^{-jm}$ and check if this element is equal to some second component of some pair in the list. If $\beta\alpha^{-jm} = \alpha^i$ for some i, then $\beta = \alpha^{jm+i}$ and hence x =jm+i and we are done. The algorithm requires a table of m entries.To sort the table and search the table for each j requires O(m log m) operations.If the group order is 10^{40} then in the current state of computer technology this attack is infeasible.

4. **Pollard's Rho Algorithm:** J Pollard's method of computing discreet logarithm is probabilistic, but does not require the precomputation of previous algorithm. Here, we partition the group G into 3 sets S_1, S_2 and S_3 of almost equal size. Some care is taken during this partition, like $1 \notin S_2$. Then we define a sequence of group elements as $x_0, x_1, \ldots$ by x_0 =1 and for each $i \geq 1$,
$x_i = \beta x_{i-1}$, if $x_{i-1} \in S_1$,
$= {x_{i-1}}^2$, if $x_{i-1} \in S_2$,
$= \alpha x_{i-1}$, if $x_{i-1} \in S_3$,
The sequence of group elements defines two sequence of integers a_i ana b_i where $x_i = \alpha^{a_i}\beta^{b_i}, i \geq 0$,$a_0 = b_0 = 0$, where $a_{i+1} = a_i + 1$ or $2a_i$ or a_i (mod n) and $b_{i+1} = b_i$, $2b_i$ or b_i+1 (mod n) depending on which set contains x_{i-1}. Making the use of Floyd's cycling algorithm, we can compute the six tuple (x_i, a_i, b_i, x_{2i}, a_{2i}, b_{2i}) for i = 1, 2, ...until $x_i = x_{2i}$.At this stage we have
$$\alpha^s = \beta^r$$
where, r = a_i - a_{2i} and s = b_{2i} - b_i (mod n).This gives $r log_\alpha\beta = s$ (mod n) There are only d = gcd(r, n) possible values for $log_\alpha\beta$. If d is small each of these values can be tested to find the correct value of $log_\alpha\beta$.

If x_i's are assumed to be a random sequence of elements of G, then expected running time of this algorithm is O(m).

5. **Parallelised Pollard's rho algorithm:** executes Pollard's rho algorithm parallelly in r processors. Runs in O(m)/(2r) steps.

6. **Pollard's Lambda Algorithm :** another randomised algorithm due to Pollard. It can also be parallelised with a linear speedup. Slightly slower than parallelised rho method. Faster if λ is known to be in a subinterval [0, b] of [0, n-1] where $b < 0.39n$.

7. **Index-Calculus Method:** First of all we attempt to find the logarithms of elements of a fixed subset

$$S = \{\sigma_1, \sigma_2, \ldots, \sigma_r\}$$

of G, called the factor base. We pick a random integer s and try to express α^s as

$$\sigma_1^{a_1}\sigma_2^{a_2}\ldots\sigma_r^{a_r}$$

. If we are successful, then taking logarithms we get,

$$s = a_1 log_\alpha \sigma_1 + a_2 log_\alpha \sigma_2 + \ldots + a_r log_\alpha \sigma_r \pmod{n}$$

After collecting a sufficient number of such relations, we can hopefully solve for the indeterminates $log_\alpha \sigma_i$. Next, we repeatedly pick random integers s until $\alpha^s\beta$ can be written as a product of elements in S, i.e.

$$\alpha^s\beta = \sigma_1^{b_1}\sigma_2^{b_2}\ldots\sigma_r^{b_r}$$

Taking logarithms in both the sides we get

$$log_\alpha\ \beta = b_1 log_\alpha \sigma_1 + b_2 log_\alpha \sigma_2 + \ldots + b_r log_\alpha \sigma_r - s \pmod{n}$$

The running time for this algorithm is $O(exp((2 + o(1))(lnp)^{1/2}(lnlnp)^{1/2}))$ for F_p.

8. **MOV attack:** Menezes et el and Frey and Ruck showed how ECDLP can be reduced to the DLP in the multiplicative group of some extension field F_{q^k} of F_q, where the number field sieve algorithm applies. To avoid this attack one only needs to check that n, the order of the divisor D_1, does not divide $q^k - 1$ for for all small k, for which DLP in F_{q^k} is tractable.Although this is an algorithm for ECDLP, there is possiblity for it to be generalised to HCDLP.

9. **Attack on HCDLP for higher values of genus g:** When g, the genus of the hyperelliptic curve is large, there is a subexponential algorithm to solve HCDLP, due to Addleman, Huang and DeMarraise [9]. So it is recommended that the hyperelliptic curve chosen for cryptographic purposes should have smaller genus.

10. **Gaudry's Algorithm:** Gaudry [10]has devised an algorithm which is faster than the Pollard's rho method if g is small and $g \geq 5$.In general if g = 2 or g = 3 and n is the largest prime factor of $\#J_C$(F), then the best known algorithm for attacking the HCDLP takes fully exponential time, O(n) time to be precise. Hence one can use a hyperelliptic curve of genus 2 over a finite field F_p, where p is of order 2^{80} to achieve the same level of security as when DLP over a finite field F_q is used with q of the order 2^{1024}.

3.3 Some Implementation Issues

The first task for implementing a discrete log cryptosystem using hyperelliptic curves is , choosing a suitable curve C and underlying finite field F must be selected. Desirable properties of the selected curve and field include the following:

1.Arithmetic in the underlying finite field F should be efficient to implement; finite fields of characteristic 2 are certainly a very attractive choice.

2.To avoid Pohlig Hellman attack, the order of the jacobian J_C(F) of C, denoted by $\#J_C$(F), should be divisible by a large prime number. Under the current scenario of computer technology, it is recommended that $\#J_C$(K) should be divisible by a prime number r of at least 45 decimal digits. In addition, to avoid the reduction attack of Frey and Ruck [11] which reduces the discreet logarithm problem in J_C(F) to the discreet logarithm problem in an extension field of F $= F_q$, r should not divide q^k - 1 for all small k for which the discrete logarithm problem in F_{q^k} is feasible ($1 \leq k \leq 2000/(log_2 q)$ suffices). Several algorithm for calculating $\#J_C$(F) are available in the literature. A hyperelliptic curve should be chosen randomly (see [12]) and a suitable algorithm for calculating $\#J_C$(F) should be used to calculate J_C(F) and on the basis of whether $\#J_C$(F) satisfies the security requirements or not, a curve generated randomly should be accepted for use or rejected.

Next, we describe a technique of selecting a hyperelliptic curve for cryptographic purposes.

Let J be the jacobian of the hyperelliptic curve C defined over F_p, and given by the equation $v^2 + h(u)v = f(u)$. Let F_q denote the degree n extension of F_p, and let N_n denote the order of the (finite) abelian group $J_C(F_q)$. Let M_n denote the number of $F_q$rational points on C. The zetafunction associated with C is, defined as:

Definition 3.1. Let C be a hyperelliptic curve defined over F_q , and let $M_r = \#J_C(F_{q^r})$ for $r \geq 1$. The zetafunction of C is the power series

$$Z_C(t) = exp(\sum_{r \geq 1} M_r t^r / r) \qquad (1)$$

We need the following properties satisfied by the zeta function:

Let C be a hyperelliptic curve of genus g defined over F_q and let Z_C(t) be the zetafunction of C.

Property 1.

$$Z_C(t) \in Z(t)$$

More precisely,

$$Z_C(t) = \frac{P(t)}{(1-t)(1-pt)}$$

where, P(t) is a polynomial of degree 2g with integer coefficients. P(t) is of he form

$$P(t) = 1 + a_1 t + \cdots + a_g t^g + p a_{g-1} t^{g+1} + p^2 a_{g-2} t^{g+2} + \cdots + p^{g-1} a_1 t^{2g-1} + p^g t^{2g}$$

Property 2. P(t) can be factorized as

$$P(t) = \prod_{1 \leq i \leq g} (1 - \alpha_i^t)(1 - \overline{\alpha}_i^t)$$

where α_i is a complex number of modulus p and $\overline{\alpha}_i$ is the complex conjugate of α_i.

Property 3. If $N_n = \#J_C(F_q)$, q = p^n, then

$$N_n = \prod_{1 \leq i \leq g} |1 - \alpha_i^n|^2$$

Hence to compute N_n we have to determine α_i's i.e. coefficients of P(t).These can be obtained by factorising P(t). Multiplying both sides of (1) by (1 - t)(1 - pt), we get,

$$P(t) = Z_C(t)(1 - t)(1 - pt)$$

Taking logarithm and differentiating both sides with respect to t,

$$\frac{P'(t)}{P(t)} = \sum_{r \geq 0}(M_{r+1} - 1 - q^{r+1})t^r$$

By equating the coefficients of $t^0, t^1, \ldots, t^{g-1}$ of both sides, we see that the first g values of M_r suffices to determine the coefficients $a_1, a_2, \ldots, a_g$, from which N_n can be determined.

The following is an important result concerning the size of $J_C(F_{p^n})$; which follows from the properties of $Z_C(t)$, described above.

Theorem 3.1. Let C be a hyperelliptic curve of genus g defined over F_p and let N_n be $\#J_C(F_{p^n})$. Then

$$(p_{n/2} - 1)^{2g} \leq N_n \leq (p^{n/2} + 1)^{2g}$$

Thus,

$$\#J_C(F_{p^n}) = p^{ng}$$

approximately.

Chapter 4

OUR IMPLEMENTATION

This chapter discusses the implementation details of a hyperelliptic curve cryptosystem over prime fields and more precisely over Z_p, where p is prime . In the next few sections, data structures and algorithm details are presented for the three main areas that were implemented, the underlying field and its arithmetic, the polynomial arithmetic and the divisor arithmetic. The implementation of divisors includes defining an algorithm for their addition and their multiplication with a big integer. The code for this project, including all the field and divisor operations was written in C.

4.1 Field Arithmetic Implementation

The elements of the Jacobians are represented as a pair of polynomials whose coefficients are elements of a finite field. To manipulate divisors one needs tools to manipulate polynomials.In order to perform polynomial operations, it is necessary to be able to realize the field operations. We only concentrate on prime fields. The advantage of using a prime field is that with slightest modification the same program can be used for a class of fields and experiment can be carried out. The program can also be easily modified to cater to fields of characteristic 2. In this section we describe implementation of the field operations and in the next section we will focus on the polynomial operations. To perform an addition in the Jacobian we need the field operations: addition, multiplication/squaring and inversion.

To make the best use of the existing facilities of the Solaris system used to implement this project the field size was chosen to be a unsigned long long integer, which is 64 bits in length. A user defined new data type was defined by the C statement:

```
typedef unsigned long long usint;
```

For the implementation of the underlying field operations we used a structure GFP_element to store a field element. Functions were defined to implement unary and binary operations on these field elements.If p is any prime then an element in Z_p is an integer lying between 0 and p-1. So the data structure representing a field element is as simple as :

```
   typedef struct
{
usint val;
} GFP_element;
```

The value of p was maintained as a global variable via a pre-processor directive statement. The following functions were used to manipulate the field elements:

```
   GFP_element addf(GFP_element ,GFP_element );
/* adds two field elements and returns the sum */

   GFP_element subf(GFP_element`,GFP_element );
/* subtracts two field elements and returns the difference */

   GFP_element mulf(GFP_element ,GFP_element );
/* multiplies two field elements and returns the product */

   GFP_element divf(GFP_element ,GFP_element );
/* divides two field elements and returns the quotient */

   GFP_element assinf (GFP_element );
/* assinement of one field element to another */

   int equf(GFP_element ,GFP_element );
/* compares two field elements, returns 1 if equal, 0 otherwise */

   GFP_element getf(usint );
/* takes an integer as input and returns a field element whose value = the integer */

   int iszerof(GFP_element);
/* checks if a given field element is zero */

   void rite(GFP_element);
/* prints the value of a field element */

   usint Inverse( usint );
/* Using Euclid's extended algorithm finds the inverse of a field element */
```

4.2 Implementation of Polynomial Arithmetic

In our program a polynomial was implemented as a C structure as defined below: `typedef struct`

```
{
            int degree;
            GFP_element coeff[MAX_COEFF];
} Poly;
```

Thus, the polynomial structure has two members : degree - an integer representing the degree of the polynomial and coeff - an array of field elements representing the coefficients of the polynomial. MAX_COEFF is a global variable defined by a preprocessor directive limiting the number of terms of a polynomial.

Again, functions were defined to implement polynomial arithmetic. The most important of them being the one finding the gcd of two polynomials using the Extended Euclidean Algorithm. This, in turn, uses another important function finding the quotient and remainder when one polynomial divides another. A list of functions used in the program runs thus:

```
   Poly create_poly(int,GFP_element*);
/* Creates a polynomial from an array of field elements used as coefficients and an integer
representing its degree */

   Poly addp(Poly,Poly);
/* For adding two polynomials */

   Poly subp(Poly,Poly);
/* For subtracting two polynomials */

   Poly mulp (Poly,Poly);
/* For multiplying two polynomials */

   void assinp (Poly *,Poly);
/* Assigning a polynomial to an empty structure */

   int equp (Poly,Poly);
/* For comparing two polynomials, returns 1 if equal, 0 otherwise */

   Poly divp(Poly,GFP_element);
/* Divides a polynomial by a field element*/
Poly square(Poly);
/* Finds square of a Polynomial */

   void show(Poly);
/* prints a polynomial */

   Poly modp(Poly* , Poly* );
/* Finds residue of a polynomial modulo another polynomial */

   Poly gcd(Poly* , Poly* );
/* Finds gcd of two polynomials */

   int iszerop(Poly*);
/* tests if a given polynomial is zero */

   Poly uminp(Poly);
/* Finds the minus of the polynomial */
```

```
   void poly_div(Poly*,Poly*,Poly*,Poly*);
/* Divides a polynomial by another polynomial, finds the quotient and remainder also */

   void exgcd (Poly*, Poly*, Poly*, Poly*,Poly* );
/* Finds gcd of two polynomials using Extended Euclidean Algorithm */

   GFP_element LT(Poly* q);
/* finds the leading coefficient of a polynomial */
```

It is worthwhile to mention the algorithms used to implement the above functions. Addition and subtraction are plain, we do not have any choice, but to do it term by term. Nor it is that expensive computationally. Multiplication and division are certainly expensive operations and we have done them in traditional high school method, leaving a lot of scope for improvement. In fact a lot of optimised algorithms for implementing these operations over GF(2^n)are available in the literature. The functions gcd and xgcd extensively use multiplication and division. So they will also behave optimally once these operations are optimized.

4.3 Implementation of Divisor operations

Divisors are represented in Mumford's notaion in the program. Every divisor is thus a pair of polynomials over the prime field. In C a divisor can be represented by a structure with two polynomial members. Thus in our program we have defined a devisor type variable by

```
   typedef struct
{
           Poly a;
           Poly b;
} Divisor;
```

Three important functions are used to manipulate divisors. One is to add two reduced divisors. Another function reduces a semi-reduced divisor into a reduced divisor using the Gaussian reduction algorithm. The third one multiplies a divisor by a large integer. Besides these, functions are written to create a divisor from two given polynomials, to write a divisor on the console, to assign a divisor to another and for other such mundane tasks.Some of them are :

```
Divisor reduce(Divisor,Poly* ,Poly* ,int* );
/* reduces a given semi-reduced divisor into a reduced dividor by Gauss Algorithm */

   Divisor addDiv(Divisor,Divisor,Poly* ,Poly* ,int* );
/* returns sum of two given reduced divisors */

   Divisor mulDiv(Divisor,usint n,Poly *,Poly* ,int*);
/* finds the product of a reduced divisor by a large integer */

   Divisor assinDiv(Divisor );
/* for divisor assignment */
```

```
   int compDiv(Divisor ,Divisor);
/* compares 2 divisors, returns 1 if equal, 0 otherwise*/

   void showDiv(Divisor);
/* displays a divisor */

   int iszeroDiv(Divisor);
/* compares a divisor with the zero divisor, returns 1 if equal, 0 otherwise.  */
```

The curve itself was represented by two global polynomials F and H. The pointers to these polynomials are passed to any function which needed them for computations. The genus of the curve was set as another global integer variable G. Although this value can be easily changed in the program, considering the fact that the secure hyperelliptic curves for cryptographic purposes are the ones with small genus, we have fixed the value of the global variable to be 2.

4.4 Implementation of the cryptosystem

On the basis of the groundwork as described above an El Gamal-type cryptosystem was developed. Before describing our implementation we give a brief overview of the ElGamal Cryptosystem over any finite cyclic group G.

ElGamal Cryptosystem

Let G be a cyclic group with generator α. A user Bob chooses an integer λ and computes $\beta = \alpha^\lambda$. He then publishes β in a public directory as his private key. Any user Alice who wishes to send a message $x \in G$ to Bob, chooses a random integer k. Computes α^k and $x\beta^k$ abd sends the pair $(\alpha^k, x\beta^k)$. Bob, on receiving the message decrypts this as follows : He raises α^k to the power λ to obtain β^k.Finds the inverse of β^k and multiplies the inverse with $x\beta^k$ to get back x.

To be more formal, the encryption function is

$$e_k(x) = (y_1, y_2),$$

where $y_1 = \alpha^k$ *and* $y_2 = x\beta^k$ and the decryption function is

$$d_k(y_1, y_2) = y_2(y_1)^k)^{-1}.$$

So to implement an ElGamal Cryptosystem over the group of divisor classes of a hyperelliptic curve we need to convert all information into group elements, i.e. reduced divisors. A reduced divisor can be represented by a pair of polynomials (a(u),b(u)) where, deg a $\leq$ g, the genus of the curve and $deg b < deg a$ and a(u)divides $b(u)^2 + h(u)b(u) - f(u)$, where h(u) and f(u) are the polynomials occuring in the equation of the hyperelliptic curve. Since the genus of the curve in this implementation is 2, a is a polynomial of degree atmost 2 and b is a polynomial of degree atmost 1. Hence a divisor can be viewed as a 5-tuple of field elements satisying certain conditions. To represent a plain text by a divisor we divide the plain text into blocks of 4 bytes each. Each block is then converted into a long long integer depending upon the ASCII value of the characters in it. Three such integers are taken as coefficients of a(u) and 2 such integers are taken to be the coefficients of b(u). Thus every twenty or lesser number of characters give us a divisor. Ofcourse, the polynomials so generated may not satisfy the criterion : a divides $b^2 + hb - f$. To ensure that the polynomials satisfy this condition each coefficients are padded with some random bits which are neglected during the decryption process.

To make the encryption and decryption process more efficient, we twisted the El Gamal scheme a bit. Instead of adding $k\beta$ to the message x during encryption process and subtracting the same from the received message during the decryption process we just XORed $k\beta$ with the message x. This obviated the necessity of converting the message into group elements i.e. divisors. This is may be a compromise from security point of view, but for this simple experiment, it gives a big boost in terms of speed.

4.5 Furthur Developments and Conclusion

Honestly speaking, this a primitive implementation of HECC implementing the basic techniques and algorithms presented in [2].There is a lot of scope for furthur improvements both in terms of performance and security considerations.In this section we describe some aspects where the implementation has scope for improvement.In fact as a consequence of relatively new nature of this field,ther are many aspects of HECC that are not fully reported upon.Even, the algorithms reported in literature for addition and reduction of divisors may not be optimal in nature.However,in this section we are highlighting possible improvements of this implementation only.

Underlying field In this implementation we have chosen prime fields as the underlying field of the cryptosystem.Although that is not a bottleneck in terms of efficiency nor does it compromise on security aspects, we can implement HECC on other fields of characteristic 2 as well.One advantage of using fields of characteristic more than 2 is that over such a field the polynomial h(u) in the equation of the hyperelliptic curve can be taken to be zero(see Proposition 2.2).

Choice of Algorithms Wherever possible, optimised algorithms have been used in this work and their origins have been referenced in this thesis. However, many solutions to problems,of which no precedence could be found, naive solutions have been used. The performance of the program can be greatly improved if optimised algorithms are used at all places.

Use of Lagrangian Reduction Algorithm In this implementation we have used the Gauss' reduction algorithm for reduction of a semi-reduced divisor into a reduced one.The performance can be improved if Lagrangian algorithm is utilised, because it takes advantage of partial computation done in each iteration.

The hyperelliptic Curve The curve chosen for the implementation of the cryptosystem has been fixed in the program. A more secure system will evolve if the curve is generated randomly each time the program is called.(see [12])

GCD Calculation One of the bottlenecks of addition of two reduced divisors is the calculation of gcd of polynomials using the Extended Euclid's algoritm. It is a relatively costly operation. Performance of the implementation will greatly improve if gcd is calculated in a more efficient way. One suggestion in this regard is to try calculating gcd's using the lattices as suggested in D. E. Knuth's *Art of Computer programming.*

Conclusion

We have presented materials relevant to implementation of a hyperelliptic curve cryptosystem.It has defined the initialisation of the cryptosystem and the methods for encryption and decryption of information using the ElGamal Scheme. This thesis has demonstrated the motivation for the study of hyperelliptic curves and shown that they can be a feasible encryption technique. With the considerations presented in this chapter and further developments suggested in this section it is conceivable that HECC could provide a commercially secure alternative to elliptic curve cryptosystems, with atleast the efficiency of elliptic curves, if not more so.

Bibliography

[1] W Diffie and M Hellman; " New Direction in Cryptology";IEEE Transactions on Information Theory; 22,(1976)

[2] N Koblitz "Elliptic Curve Cryptosystem";Mathematics of Computations";48;(1987).

[3] N Koblitz; "Hyperelliptic Curve Cryptosystem";Journal of Cryptology;1,(1989).

[4] A J Menezes,Y H Wu,R J Zuccherato;"An Elementary Introduction to Hyperelliptic curves";Technical Report;University of Waterloo,(1996);CORR-96-19.

[5] D Cantor; "Computing in the Jacobian of the Hyperelliptic Curves";Mathematics of Computations;48;(1987)

[6] A Enge; "The Extended Euclid Algorithm on Polynomials and the Computational Efficiency of hyperelliptic Curves";Nov 1999;Pre-print.

[7] S Paulus and A Stein; "Comparing Real and Imaginary Arithmetic for Divisor Class Groups of Hyperelliptic Curves";Algorithmic Number Theory;;Vol 1423;Springer-Verlag Lecturer Notes on Computer Science.

[8] L Addleman,J DeMarrais.M Huang; " A Sub-exponential Algorithm for Discreet Logarithm over the Rational Subgroup of the Jacobians of Large Genus Hyperelliptic Curves over Finite Fields";Algorithmic Number Theory;Lecture Notes on Computer Science;*77(1994).

[9] T Wollinger; "Computer Architecture for Cryptosystems based on Hyperelliptic curves";M.S. Thesis;Worcestor Polytechnique;UK.

[10] P Gaudry; "An Algorithmfor Solving the Discreet Logarithm Problem on Hyperelliptic Curves"; Advances in Cryptology Eurocrypt 2000;LNCS 1807,2000.

[11] G Frey and Ruck; "A Remark Concerning m-divisibility and the Discreet Logarithms in the Divisor Class Group of Curves";Mathematics of Computations;62;(1994).

[12] F Hess, G Seroussi, N Smart; "Two Topics in Hyperelliptic Curve Cryptography"; HP Tech Report,HPL-2000-118.

[13] V Miller; "Uses of elliptic curves in cryptography"; Advances in Cryptology - Proceedings of Crypto'85, LNCS, 218(1986).

[14] D Gordon "Discreet Logarithm in GF(p_n) using the Number Field Sieve",pre-print(1991).