ELLIPTIC CURVE FINAL REPORT

MICHAEL KREL, SHAHAR PAPINI

1. Introduction To Elliptic Curve Theory

Definition 1. An *elliptic curve* E over a field K with characteristic $char\,(K) \neq 2, 3$ is defined as follows:

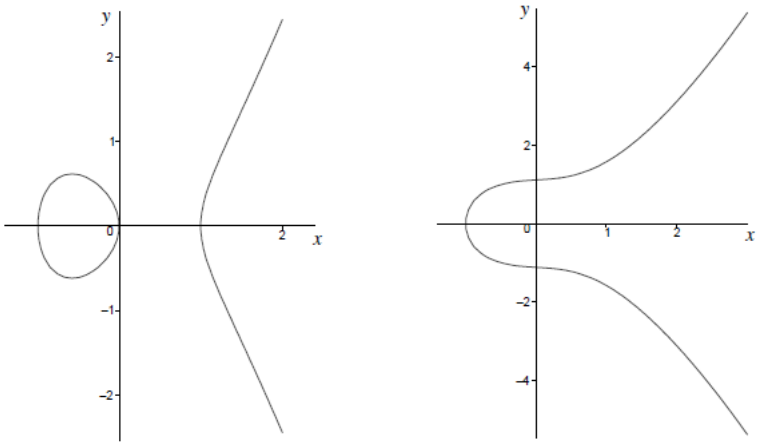
$$E:\ y^2 = x^3 + Ax + B \tag{1.1}$$

for $A,\ B \in K$, s.t. the discriminant $\Delta = -16\left(4a^3 + 27b^2\right)$ is non zero.

If L is any extension field of K, then the set of *L-rational points* on E is

$$E\,(L) = \left\{(x, y) \in L^2\ :\ y^2 = x^3 + Ax + B\right\} \cup \{\infty\}$$

where ∞ is a special point, also referred to as zero.



(a) $E_1 : y^2 = x^3 - x$ (b) $E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$

FIGURE 1.1. Elliptic Curves over $\mathbb{R}$(Taken from [1])

Equation 1.1 is called *Weierstrass equation.*

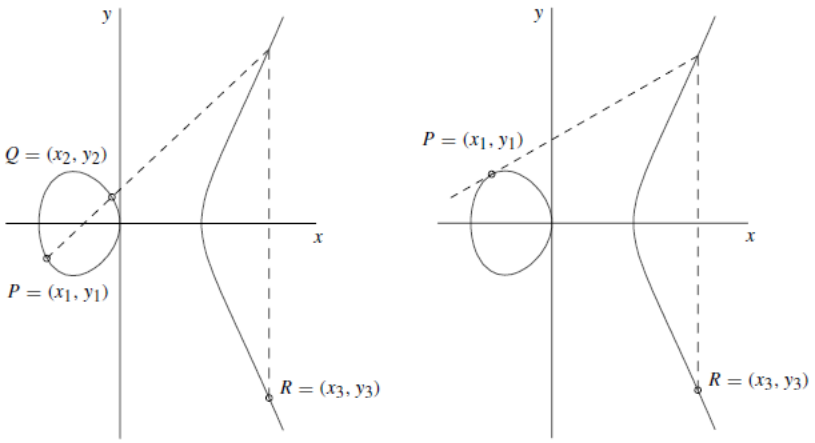
Definition 2. Two elliptic curves E_1 and E_2 defined over K and given by the Weierstrass equations are said to be *isomorphic over* K if there exists $u, r, s, t \in K$ s.t. the change of variables

$$(x, y) \to \left(u^2 x + r, u^3 y + u^2 s x + t\right) \tag{1.2}$$

transforms equation E_1 into equation E_2. The transformation 1.2 is called an admissible change of variables. Equivalently, two curves are isomorphic if they have the same $E(L)$ for any L.

Remark 3. The real definition of elliptic curves is a bit different, and also holds for fields with characteristic 2 or 3, however, for $char(K) \neq 2, 3$, they are all isomorphic to the form given in 1.1.

1.1. **Group Law.** It happens to be the case that we can define an abelian group structure over the points $E(K)$. This is done by introducing an addition operator over two points. The operation can be defined algebraically, but we will first see the geometric definition, when $K = \mathbb{R}$. Given two points with distinct x coordinates, $P_i = (x_i, y_i) \in E(\mathbb{R})$, the operation on these two points can be considered as follows: Take the line connecting the two points, then take the third point on this line that is also on the elliptic curve(promised to exists). Take its reflection along the x axis. This is the resulting point. When the two points are the same point, the operation is called doubling and is obtained using continuity: instead of the line connecting the two points, we take the tangent line to the point. The operations are shown in this figure: If $x_1 = x_2,\ y_1 + y_2 = 0$, then the resulting point is ∞.



(a) Addition: $P + Q = R$. (b) Doubling: $P + P = R$.

FIGURE 1.2.

Geometric addition and doubling of elliptic curve points(Taken from [1])

Algebraically:

(1) Identity. $P + \infty = \infty + P = P$ for all $P \in E(K)$.

(2) Negatives. If $P = (x, y) \in E(K)$, denote $-P = (x, -y)$, the negative point. Then $P + (-P) = \infty$. Note that indeed $-P \in E(K)$. Also, $-\infty = \infty$. This is seen also from the geometric definition, since there is not third point on the lines connecting (x, y) and $(x, -y)$.

(3) Point addition. Let $P = (x_1, y_1) \in E(K)$, $Q = (x_2, y_2) \in E(K)$, $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \text{ and } y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1$$

(4) Point doubling. Let$P = (x_1, y_1) \in E(K)$, where $P \neq -P$. Then $2P = (x_3, y_3)$, where

$$x_3 = \left(\frac{3x_1 + a}{2y_1}\right)^2 - 2x_1 \text{ and } y_3 = \left(\frac{3x_1 + a}{2y_1}\right)(x_1 - x_3) - y_1$$

Remark 4. Notice that the last 2 formulas are a proof for the existance of the third point on the line connecting the points P, Q.

1.2. **Group Order.** From now on, we focus on K being the finite field F_q.

The order of a group, defined as the number of points it contains, is more significant than it might look at the beginning. Various properties of the group depend on this number alone. The order of $E(F_q)$ is denoted $\#E(F_q)$. We will later see that it affects the level of security of cryptographic procedures performed over this group.

Theorem 5. *(Hasse) Let E be an elliptic curve defined over F_q. Then*

$$|q - 1 - \#E(F_q)| \leq 2\sqrt{q}$$

We have $\#E(F_q) = q - 1 + t$, where $|t| \leq 2\sqrt{q}$. t is called the trace.

Remark 6. Notice that it means $\#E(F_q) \approx q$, since $\sqrt{q} \ll q$.

2. Point representation and the group law

Formulas for adding two elliptic points were presented in 1.1 for the elliptic curves $y^2 = x^3+ax+b$ defined over a field K of characteristic that is neither 2 nor 3, and for $y^2+xy = x^3+ax^2+b$ defined over a binary field K. For both curves, the formulas for point addition (i.e., adding two distinct finite points that are not negatives of each other) and point doubling require a field inversion and several field multiplications. If inversion in K is significantly more expensive than multiplication, then it may be advantageous to represent points using projective coordinates.

2.1. **Projective coordinates.** In the projective coordinates representation, we define an equivalence class of points. every jacobean point is of the form $(X : Y : Z)$. It corresponds to the point on the curve, $\left(XZ^{-c}, YZ^{-d}\right)$. This is the equivalent from projective coordinates of real manifolds. $Z = 0$ corresponds to ∞. The Jacobean coordiantes are the special case of $c = 2$, $d = 3$. The reason for using these coordinates in that the group law can be formulated without inverses(which is an expensive operation).

We denote $xZ^2 = X$ and $yZ^3 = Y$. Given two points, $P_i = (X_i : Y_i : Z_i)$, the point addition formula is given by

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \text{ and } y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1$$

We will play with these a bit:

$$\begin{aligned}
x_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \\
x_3 (x_2 - x_1)^2 &= (y_2 - y_1)^2 - (x_1 + x_2)(x_2 - x_1)^2 \\
Z_1^2 Z_2^2 x_3 \left(Z_1^2 Z_2^2 x_2 - Z_1^2 Z_2^2 x_1\right)^2 &= \left(Z_1^3 Z_2^3 y_2 - Z_1^3 Z_2^3 y_1\right)^2 \\
&\quad - \left(Z_1^2 Z_2^2 x_1 + Z_1^2 Z_2^2 x_2\right)\left(Z_1^2 Z_2^2 x_2 - Z_1^2 Z_2^2 x_1\right)^2 \\
Z_1^2 Z_2^2 x_3 \left(Z_1^2 Z_2^2 x_2 - Z_1^2 Z_2^2 x_1\right)^2 &= \left(Z_1^3 Y_2 - Z_2^3 Y_1\right)^2 - \left(Z_2^2 X_1 + Z_1^2 X_2\right)\left(Z_1^2 X_2 - Z_2^2 X_1\right)^2
\end{aligned}$$

Defining $Z_3 = Z_1 Z_2 \left(Z_1^2 X_2 - Z_2^2 X_1\right)$, we get

$$X_3 = \left(Z_1^3 Y_2 - Z_2^3 Y_1\right)^2 - \left(Z_2^2 X_1 + Z_1^2 X_2\right)\left(Z_1^2 X_2 - Z_2^2 X_1\right)^2$$

Denoting $Z_1^2 X_2 = s$, $Z_2^2 X_1 = r$, $Y_1 Z_2^3 = t$, $Y_2 Z_1^3 = u$, $v = s - r$, $w = u - t$, we get $Z_3 = Z_1 Z_2 (s - t) = Z_1 Z_2 v$.

$$\begin{aligned}
X_3 &= (u - t)^2 - (s + r)(s - r)^2 \\
&= w^2 - (s - r + 2r)(s - r)^2 \\
&= w^2 - v^3 - 2rv^2
\end{aligned}$$

Similarly,

$$Y_3 = -tv^3 + \left(rv^2 - X_3\right) w$$

2.2. **Compressed coordinates.** This representation is meant to save space rather than speed. We observe that storing the pair (x, y) has a lot of redundancy. Given x, we know $y^2 = x^3 + ax + b$. Given x, this equation has at most two solutions for y, which differ by sign alone. Therefore, it is enough to store only x, and another bit which holds whether we want the greater y value among the two. This method is mainly for storing, and not for calculations, so when calculations are needed, we transform the point to standard coordinates

3. The elliptic curve $y^2 = x^3 + ax + b$

3.1. **Point multiplication.** This section considers methods for computing kP, where k is an integer and P is a point on an elliptic curve E defined over a field Fq . This operation is called point multiplication or scalar multiplication, and dominates the execution time of elliptic curve cryptographic schemes.

Non-adjacent form (NAF)

We all know the standard binary exponentiation algorithm for computing kP. In this method we look at the binary representation of k, and repeatedly square P(to get $P, 2P, 4P, 8P, \cdots$) and the desired quantities. However, we did not use the fact that subtraction is as efficient as addition. So, we might get better results, when trying to represent k as $\sum_{i=0}^{l-1} k_i 2^i$ where $k_i \in \{-1, 0, 1\}$. This is where we use NAF(non adjacent form).

Definition 7. A non-adjacent form (NAF) of a positive integer k is an expression$k = \sum_{i=0}^{l-1} k_i 2^i$ where $k_i \in \{-1, 0, 1\}, k_{l-1} \neq 0$, and no two consecutive digits k_i are nonzero. The length of the NAF is l.

Theorem 8. *(properties of NAFs) Let k be a positive integer.*

(1) k has a unique NAF denoted NAF(k).
(2) NAF(k) has the fewest nonzero digits of any signed digit representation of k.
(3) The length of NAF(k) is at most one more than the length of the binary representation of k.
(4) If the length of NAF(k) is l, then $2^l/3 < k < 2^{l+1}/3$.
(5) The average density of nonzero digits among all NAFs of length l is approximately 1/3.

The algoeithm we wrote converts the number to NAF on the fly, as it does the exponentiation:

Algorithm:: Binary NAF method for point multiplication
INPUT:: Positive integer $k, P \in E(F_q)$.
OUTPUT:: kP.

(1) $Q \leftarrow 0$
(2) While k>0
    (a) If k ends with 0: $k = k/2$, $P = 2P$.
    (b) Else

(i) If k ends with 11: $k \leftarrow k+1$, $Q = Q - P$.
(ii) If k ends with 01: $k \leftarrow k-1$, $Q = Q + P$.
(iii) $k = k/4$, $P = 4P$.
(3) Return(Q)

Notice that the key here is the invariant $Q + kP$. At the beginning this is kP, the result we need. We keep this invariant until $k = 0$, and then this is just Q, so it holds the desired result.

4. Why elliptic curve cryptography?

The discreate logarithm problem is the hardness of finding the number x which holds $g^x = h$ in a group G when $h, g \in G,\ x \in \mathbb{N}$. The hardness of finding x is the basics of a lot of public-key encryption and signature schemes.

Definition 9. The elliptic curve discrete logarithm problem (ECDLP) is: for elliptic curve E over finite field , when we get to points in the curve $P \in E(F_q), Q \in <P>$we need to find l that satisfy the equation $Q = lP$.

We choose the elliptic curve parameters such that no known attack could be applied efficiently. The exhaustive search solves the ECDLP by computing $2P, 3P...kP$ untill he finds k such that $kP = Q$. In that method the average number of steps is $n/2$ when n is the order of P, Therefore to avoid that attack we need to choose P such that $n > 2^{80}$. The best known attack is combination between Pohlig-Hellman and Polard Rho attacks in this method the average number of steps is $\sqrt{p}$ but again in order to avoid that attack we need to choose $p > 2^{160}$ in order to make even "the best" algorithms for solving the ECDLP unefficent for the modern computers.

In order to use the discreate logarithm we need to find groups that their structure make the discrete logarithm hard to find. Z_p, for a prime p, is the most common group used, since no polynomial algorithm for DLP of this group is known. However, there is a sub exponential algorithm to solve the discreate logarithm for Z_p. Elliptic Curves give another group for which no such algorithm is known. In addition calculations in the EC group are much more complicated than the ones in the $Z_p$field. These properties make EC much more effective for cryptographic algorithms that are based on the hardness of DLP, such as DL, EL-GAMAL.

The following table presents the sizes of keys that give the same security level for Symmetric encryption and public encryption using $Z_p$or ECC.

Symmetric key size	Z_p public key size	Elliptic Curve public key size
80 bit	1024 bit	160 bit
112 bit	2048 bit	224 bit
128 bit	3072 bit	256 bit
192 bit	7680 bit	384 bit
256 bit	15360 bit	512 bit

The table was taken from various sources over the net.

4.1. **Possible Attackes.**

4.1.1. *Pohlig-Hellman attack.* This attack is actually a reduction. We are given P, $Q = lP$ and n, the order of P. We want to find l. This reduction says that it is enough to solve this problem when n is prime. This is how it works: $n = \prod_i p_i^{\alpha_i}$. We will compute $l_i = l \,(mod\ p_i^{\alpha_i})$ for each i, and then using CRT(Chinese remainder theorem) we will get l. We will throw away the index i, since we will only look at $l_i, p_i, \alpha_i, \cdots$ Assume

$$l = \sum_{j=0}^{\alpha-1} b_j p^j \quad b_j \in [0, p-1]$$

We take $\frac{n}{p}P$ $\frac{n}{p^k}Q$. Now, the order of $\frac{n}{p}P$ is p. We have

$$\begin{aligned} \frac{n}{p^k}Q &= l\frac{n}{p^k}P \\ &= \sum_{j=0}^{\alpha-1} b_j p^j \cdot \frac{n}{p^k}P \\ &= \sum_{j=0}^{k-1} b_j \cdot \frac{n}{p^{k-j}}P \end{aligned}$$

Assuming we already know $b_0, \cdots, b_{k-2}$, we can get subtarct first terms and get

$$b_{k-1} \cdot \frac{n}{p}P$$

If we solve ECDLP, we will find b_{k-1} as well. We continue this way untill we have all b coefficients.

4.1.2. *Pollard's rho attack.* We can assume $n = p$ prime, using the previous attack. In this attack we seek to find linear combinations of P and Q with the same value. For example, if we know $aP = bQ = lbP$, then $a = lb \,(mod\ p)$, and $l = ab^{-1} \,(mod\ p)$. The key is how to find these combinations. The naive way is to randomly choose pairs a, b, we save them in the memory with the point $aP + bQ$ when we find two pairs a, b and c, d such that $aP + bQ = cP + dQ$ then we actually found that $(a - c)\, P = (d - b)\, Q \to eP = fQ$ and we can find l. According to the "birthday paradox" it would take $O(\sqrt{n})$ steps. The greatest disadvantage of that method is that we need to keep a lot of data in the memory.

The idea behind Pollard's rho method is to define some function on (a, b) that gives a new pair (a', b'), which is more or less random. We start from some a_0, b_0. Then, iteratively, $(a_{i+1}, b_{i+1}) = f\,(a, b)$. Again, using the birthday paradox, we can expect that after about $O\,(\sqrt{n})$ steps, we will have $a_i P + b_i Q = a_j P + b_j Q$. We wish to find those, without saving a lot of data. What we do here is that in the i_{th} step, we compare $a_i P + b_i Q$ with $a_{2i} P + b_{2i} Q$ and $a_{2i+1} P + b_{2i+1} Q$. This is actually Floyd's Cycle Finding algorithm.

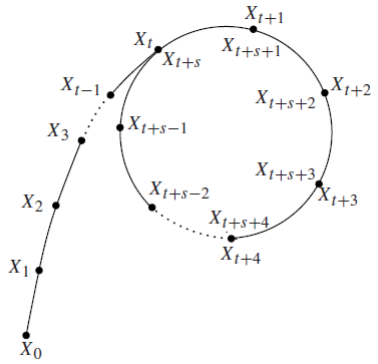


FIGURE 4.1.

Taken from [1]

The question remains: how do we define such a function? A popular way is to partition the points into L sets (A popular choice is $L = 32$, where a curve point is placed in the set with the same index as the last 5 bits of the x coordinate). Each set has two randomly chosen numbers a_i, b_i for $0 \leq i \leq L-1$. The point P is placed in set $S(P)$. Then $f(a,b) = \left(a + a_{S(aP+bQ)}, b + b_{S(aP+bQ)}\right)$.

5. ElGamal With EC

The EL-GAMAL algorithm gets EC E and chosen point P on the curve.

Suppose Bob want to send message to Alice.

Then Alice creates cryptographically strong EC E and chooses an appropriate point P on the curve(the selection of the point will be explained later).

Alice randomly chooses an integer s and the public-key of Alice is the EC E, the point P and sP.

In order to find s from the public key, one needs to to solve the DLP problem, which is hard.

Now Bob sends the message M which is encoded as a point on the elliptic curve (the encoding method will be explained later) In the following way:

(1) randomly choose an integer k
(2) compute $c_1 = kP$
(3) compute $c_2 = M + kQ$
(4) send to Alice the message (c_1, c_2)

To decrypt the message Alice computes $c_2 - sc_1 = M + kQ - skP = M + ksQ - skP = M$ and gets the message M.

We can see that Alice sends only one message while Bob sends her two cipher messages. We can see the encryption time is twice the decryption time.

5.1. **How to select the point P?.** Each point P on the elliptic curve E is part of a subgroup of E our goal is to choose point that the order of the group has a very large prime factor. One of the reason for this is that otherwise, one could easily brute force for s and decrypt the message M. Assuming our elliptic curve is strong, of order $p \cdot k$ (see 6.1), we want to find a point P with order divisible by p . In order to do that, we find a random point P. If $P^k \neq \infty$, we know this is the case, and we return the point.

5.2. **How to encode a message to a point on the EC?.** Given a specific x value, there is 50% chance that x^3+ax+b is a resdiue modulo p. So, our process of converting a message M into a point on the curve is this. Choose $0 \leq k < t$. Take $x = tM + k$. If $x^3 + ax + b$ is a residue, then this is a proper point(along with the corresponding y value) that represents M. We look for any k that holds this. The probability that none of these t values is a proper x value of a point is 2^{-t}, which is negligable. To decode it back, we take the x value of the point, and divide it by t(taking the floor value). In our implementation, we took $t = 100$. A big message is thus divided into blocks.

6. Elliptic Curve Generation

When using elliptic curve cryptography, it is important to use a strong elliptic curve. Otherwise, it may be vulnerable to attacks. Usually we need just one point on this curve, with a large order. For this, we need to know the order of our curve (group). We require this order to have some very large prime number as a factor.

6.1. **Security issues.** We require the order of the elliptic curve to be $p \cdot k$ for prime p and a very small k(not more than 10). The bound on k is variable that presents the tradeoff between generation efficiency, and curve strength. Most applications use some specific point P as a cyclic group generator. We require the order of this point to be at least p. The prime p is usually several hundred bits long.

6.2. **Point counting approach.** The obvious method is to generate a random elliptic curve, find its order, and test if it is strong enough. The problem is that finding the order of the curve is usually a hard task, though there are some algorithms to do it. Those algorithms are labeled under "Point Counting" algorithms. They are usually very slow compared to the other alternative presented below. The advantage of using this approach is that we have a uniform distribution over all the possible strong curves. The disadvantage is that it is usually slow.

6.3. **Complex multiplication approach.** This method tackles the problem from another angle. Instead of generating a random curve and calculating its order, this method first determines the desired order, and then generates a corresponding curve. Although this method is relatively very fast, it does not generate every possible elliptic curve with even probability. So far, it doesn't seem to be a major flaw in this approach, since these curves have no known common weakness. This method is explained in detail in the next section.

7. Complex Multiplication

Definition 10. Given an elliptic curve E over a field K, an endomorphism is an homomorphism from $E\left(\overline{K}\right)$ to itself that can be represented as a rational function of the coordinates. $\overline{K}$ is the algebraic closure of K, and $E\left(\overline{K}\right)$ is the set of points $(x, y) \in \overline{K} \times \overline{K}$ that hold Weierstrass equation 1.1.

Remark 11. This is a more specific definition for endomorphisms, but this is the one used regarding elliptic curves.

Example 12. $\phi(P) = nP$ is a trivial endomorphism. That is because Point addition and doubling are rational functions(as we saw in the group law definition), and so ϕ is a rational function as well. We can easily see it preserves addition.

It appears that the set of endomorphisms on a curve E from a ring structure under point addition($(\phi + \psi)(P) = \phi(P) + \psi(P)$), and composition($(\phi \circ \psi)(P) = \phi(\psi(P))$). If there are non-trivial endomorphisms on E, then E is said to have Complex Multiplication, and the endomorphisms ring is isomorphic to a an order in a quadratic imaginary field, O_D. D is the discriminant of E, $D < 0$ and $D = 0, 1 \pmod 4$.

Remark 13. When the underlying field in a finite field, as in our use, there are always non-trivial endomorphisms. For example, the Frobenius endomorphism, defined as $\phi(x, y) = (x^q, y^q)$, is a non-trivial endomorphism when $K = F_q$.

Let the number of points in the curve, $\#E = p+1+a$. We can write $X^2+aX+p = (X-\alpha)(X-\beta)$. Since a, p are integers, $\alpha = \overline{\beta}$. It turns out α is a member of the quadratic imaginary field, that is, $\alpha = \frac{a+b\sqrt{D}}{2}$ where $a, b \in \mathbb{Z}$. We get

$$\begin{aligned} p &= \alpha\overline{\alpha} = \frac{a^2 + b^2|D|}{4} \\ 4p &= a^2 + b^2|D| \end{aligned}$$

When we have p and D, we can solve this diophantic equation(that is, a, b are integers), and get$|a|$. So, when we find p, D such that $p + 1 + a$ or $p + 1 - a$ are orders we want, we can try to find an elliptic curve with discriminant D over F_p.

7.1. Cornacchia's algorithm.

Solving the diophantic equation $p = a^2 + b^2|D|$, is a known mathematical problem. Cornacchia suggested the following algorithm: First, notice that D must be a square mod p($D = \left(\frac{a}{b}\right)^2 \pmod p$). So, we find a root modulo p of D, r. We use Euclid's algorithm on p and r until we get a linear combination of them, s, lower than $\sqrt{p}$. We then return $\left(s, \sqrt{\frac{p-s^2}{|D|}}\right)$ is applicable(turns outs $\frac{p-s^2}{|D|}$ must be a square if there is a solution). Since we have $4p$ instead of p there are slight modifications.

7.2. **The j-invariant.**

Definition 14. Given an elliptic curve $E: \ y^2 = x^3 + ax + b$, the j-invariant is $j(E) = 1728\frac{4a^3}{4a^3+27b^2}$. It appears the there are no more than two distinct elliptic curves up to isomorphism (see definition 2) with a specific j-invariant.

Definition 15. Given an elliptic curve $E: \ y^2 = x^3 + ax + b$ and a non residue s, the twist of this curve is the curve $E' : y^2 = x^3 + as^2x + bs^3$.

The twist has the same j-invariant. $\#E + \#E' = 2(p+1)$. In other words, if $\#E = p + 1 + a$, then $\#E' = p + 1 - a$. So, for non-zero a, the curve and it's twist are the only curves of this j-invariant, up to isomorphism.

So, we already have the p and D for our curve. Thus, it is enough to find the j-invariant. Given j we can construct a curve with that j-invariant: Let $k = \frac{j}{1728-j}$. Take the curve

$$y^2 = x^3 + 3kx + 2k \tag{7.1}$$

Its j-invariant is

$$1728\frac{4(3k)^3}{4(3k)^3 + 27(2k)^2} \quad = \quad 1728\frac{k}{k+1} = j$$

7.3. **Hilbert Class Polynomial.** Given D, we can generate a polynomial $H_D(x)$, called Hilbert Class Polynomial. This is the minimal polynomial of the set of j-invariants of elliptic curves with discriminant D. It turns out that when taking a root of this polynomial modulo p, we get a j-invariant corresponding to a curve with discriminant D under F_p. So, Once we compute $H_D(x)$, which requires a bit of work with arbitrary precision, we find some root modulo p, j and find using this j-invariant a proper elliptic curve. notice that we may want to take the twist of this curve to get the order we wanted.

7.4. **Weber Class Polynomial.** There is some other polynomial, called Weber Reduced Class Polynomial, $W_D(x)$. It is the minimal polynomial, only of different invariants. The point behind taking this polynomial is that its coefficients are much smaller compared to H_D, and so, we require a lot less precision when calculating it online. This makes its computation a lot more efficient. When we get a root of this polynomials modulo p, we can transform it back to the j-invariant.

7.5. **Combined Algorithm - Atkin Morain.** This algorithm is due to Atkin and Morain.

ALGORITHM:: Hillbert polynomial variant
INPUT:: prime p
OUTPUT:: $(a, b, \#E)$ representing the elliptic curve $E: \ y^2 = x^3 + ax + b$.

(1) Pick a proper value D for the discriminant.
(2) Solve $4p = u^2 + |D|v^2$. If $p + 1 \pm u$ are both not strong enough, goto Step 1.

(3) Calculate $H_D(x)$.
(4) Find a root of H_D modulo p, j.
(5) Use (7.1) to get an elliptic curve with right order.
(6) Return this curve and its twist.

This algorithm is a variant which uses Weber polynomials for the sake of time efficiency

ALGORITHM:: Weber polynomial variant
INPUT:: prime p
OUTPUT:: $(a, b, \#E)$ representing the elliptic curve $E: \ y^2 = x^3 + ax + b$.

(1) Pick a proper value D for the discriminant.
(2) Solve $4p = u^2 + |D| v^2$. If $p + 1 \pm u$ are both not strong enough, goto Step 1.
(3) Calculate $W_D(x)$.
(4) Find a root of W_D modulo p, f.
(5) Convert this root f to the corresponding j-invariant j.
(6) Use (7.1) to get an elliptic curve with right order.
(7) Return this curve and its twist.

8. Weber Class Polynomial

Like the Hilbert polynomial which is the minimal polynomial of the j-invariant over the quadratic field of discriminant D, Weber polynomial is a minimal polynomial of a class invariant. This class invariant, denoted f is one of the Weber functions f_1, f_2, f_3, depending on D. To compute these polynomials we need to iterate over the entire class group. It sppears that the class group can be represented in a form of reduced symmetrix 2x2 matrices:

Definition 16. A *reduced symmetric matrix* is one of the form

$$S = \begin{pmatrix} A & B \\ B & C \end{pmatrix}$$

where the integers A, B, C satisfy the following conditions:

(1) GCD $(A, 2B, C) = 1$
(2) $|2B| \leq A \leq C$
(3) If either $A = |2B|$ or $A = C$, then $B \leq 0$

We denote this matrix as $[A, B, C]$. The determinant of this matrix is $AC - B^2$. It is true that this group is isomorphic to the class group with discriminant D, $H(D)$. We thus want to find out who are the elements in this group. When we do this, we can calculate the class invariant for each element, and find the minimal polynomial, which will then be the desired class polynomial(Hilbert or Weber, depending on chosen invariant).

8.1. Class Group and Class Number.

The following algorithm produces a list of the reduced symmetric matrices of a given determinant D. This algorithm is straight forward, we pretty much iterate over all possibilities, in a smart way. This algorithm is taken from[8].

Input: a squarefree determinant $D > 0$.

Output: the class group $H(D)$.

(1) Let s be the largest integer less than $D/3$.
(2) For B from 0 to s do
    (a) List the positive divisors $A1, \ldots, Ar$ of $D + B^2$ that satisfy $2B \leq A \leq D + B^2$
    (b) For i from 1 to r do
        (i) Set $C \leftarrow (D + B^2)/A_i$
        (ii) If GCD $(A_i, 2B, C) = 1$then: list $[Ai, B, C]$, if $0 < 2B < A_i < C$ then list $[A_i, -B, C]$
(3) Output list.

8.2. **Creating Weber Class Polynomial.** Here we need to show how to calculate the class invariant for the weber polynomial (based on weber function).

Remark 17. This is a slightly modified version of the one presented in [8], as there was an error in the original.

Let

$$F(z) = 1 + \sum_{j=1}^{j=\infty} (-1)^j \left(z^{(3j^2-j)/2} + z^{\left(3j^2+j\right)/2} \right) = 1 - z - z^2 + z^5 + z^7 - z^{12} - z^{15} + ...$$

and

$$\theta = exp\left(\frac{-\sqrt{D} - Bi}{A} \pi \right)$$

Let

$$\begin{gathered} \mathrm{f}_0(A,B,C) = \theta^{\frac{-1}{24}} F(-\theta)/F(\theta^2), \\ \mathrm{f}_1(A,B,C) = \theta^{\frac{-1}{24}} F(\theta)/F(\theta^2) \\ \mathrm{f}_2(A,B,C) = \sqrt{2}\theta^{\frac{1}{12}} F(\theta^4)/F(\theta^2). \end{gathered}$$

Those are the Webere functions.

If $[A, B, C]$ is a matrix of determinant D, then its class invariant is:

$$C(A,B,C) = (N\lambda^{-BL} 2^{\frac{-I}{6}} (\mathrm{f}_J(A,B,C))^K)^G$$

Where $N, \lambda, B, L, I, J, K, G$ are determened by a finite number of cases depending on D, A, B, C.

Now, as we mentioned before, we take all $C\left(A_i, B_i, C_i\right)$ for the class group $[A_1, B_1, C_1], ..., [A_h, B_h, C_h]$ computed by the last algorithm. We then find their minimal polynomial:

$$W_D = \prod_{j=1}^{h} \left(t - C\left(A_i, B_i, C_i\right) \right)$$

The reduced class polynomial has integer coefficients.

The above computations must be performed with sufficient accuracy to identify each coefficient of the polynomial $w_D(t)$. Since each such coefficient is an integer, this means that the error incurred in calculating each coefficient should be less than $\frac{1}{2}$.

When calculating the weber class polynomial, there is a factor of about $\binom{h}{\frac{h}{2}}$ in the number of precision bits required, where h is the size of the class group.

9. Implementation

We used libraries for basic operations on numbers and polynomials. The elliptic curve arithmetic we implemented ourselves. The used libraries:

- NTL: A Library for doing Number Theory. NTL is a high-performance, portable C++ library providing data structures and algorithms for manipulating signed, arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields. Can be found here: http://www.shoup.net/ntl/.
- ARPREC: This C++ library supports a flexible, arbitrarily high level of numeric precision. High-precision real, integer and complex datatypes are supported. Can be found here: http://crd.lbl.gov/∼dhbailey/mpdist/.

We implemented:

- General EC interface:
  - Jacobean coordinates point representation.
  - Compressed coordinates point representation.
  - Efficient group law in Jacobean coordinates.
  - Efficient point multiplication using NAF.
  - Generate a point on a given strong elliptic curve, suitable for cryptographic purposes.
- Complex multiplication:
  - Cornacchia's algorithm, to solve the diophantic equation
  - Calculate the class group.
  - Generate Weber Polynomial.
  - Transform a Weber polynomial root into a Hilbert polynomials root.
  - Generate a random strong elliptic curve using either precomputed Hilbert polynomials or online calculated Weber polynomials.
- Encryption:
  - ElGamal Private and Public Key Generation.
  - Block encryption and decryption using ElGamal.
  - Encryption and decryption of byte streams.

10. How To Use The Program

10.1. **The EC Library.** The **EC.h** contains the arithmetic functions for handeling EC. class EC contains point on the curve and A,B parameters. The functions that are in the h file are determinante calculation, group arithmetics and return options which allows to see the point. Also their is an option of getting random point on a given curve.

10.2. **Generate Elliptic Curve.** In order to use this function you need to include in your file the **ECGen.h** file.

To generate EC you need to call the function $ECParameters\ \ AtkinMorain(ZZ\ p, bool\&\ Success, int\ MaxDiv, Z$

This function generate EC over the field $Z_p$with $MaxDiv$ as a parameter. The function return the A,B values that are parameters in the struct ECParameters and the order in case of succsess and return false in the bool parameter if it couldn't generate EC.

10.3. **Encryption And Decryption Of Data.** In order to use this functions you need to Include **ECCrypt.h** file.

The following function creating public key for EC EL-GAMAL $void\ EGCreatekey(int\ bits, int\ MaxDiv, ZZ\&\ p,$

The function gets the key size $MaxDiv$and return by reference elliptic point on strong EC from randomly choosen prime field and also return the private key $s$and the public key: point P, point $B = sp$and the EC parameters that are in the struct ECParameters.

The following function encrypt the data: $void\ EGEncryptStream(ECParameters\ ecp, EC\ P, EC\ B, ZZ\ p, istre$

The function gets the public key that created by the previous function and encrypt the data with EL GAMAL algorithm the function read the message from istream in and wright the cipher to ostream out.

The following function decrypt the data: $void\ EGDecryptStream(ECParameters\ ecp, EC\ P, ZZ\_P\ s, ZZ\ p, istr$

The function using the private key created by $EGCreatkey$function and decrypt the data that it read from istream in and wright the decrypted data in ostream out.

To sum it up if Bob want to send message to Alice then Alice call the function $EGCreatekey$send to Bob the public key Bob calles to the function $EGEncryptStream$and send Alice the cipher he created. Then Alice call to $EGDecryptStream$ and get the message.

11. Results

Our goals in the project were to create cryptogtaphically strong elliptic curve of size of $200 - 300$bit in a few seconds and to encrypt and decrypt data of size of 3k in less then 5 seconds.

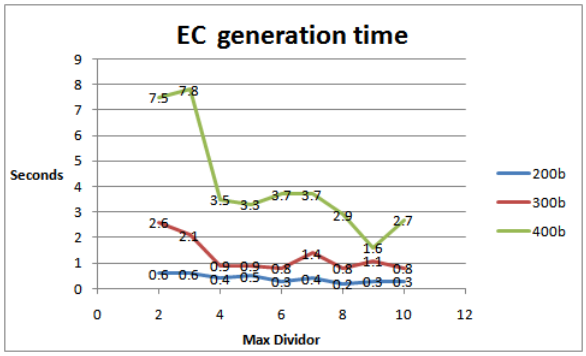


Figure 11.1.

This graph presents the time that takes to generate EC with order of $Max - Divisor \times p$ where p is large prime. the graph presents primes of size of $200, 300, 400$bit.

It shown in the graph that we achive our goal since even for $Max - Divisor = 4$ we manage to create EC in less then a second. Also we can see that if as bigger the $Max - Divisor$ the faster the elliptic curve geberates, also as bigger the prime longer it takes longer it takes to create EC of his size.

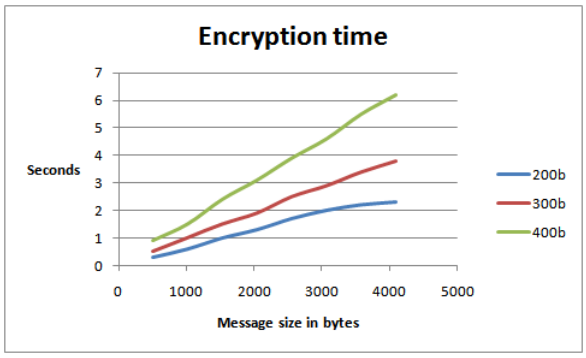


Figure 11.2.

This graph presents the time that needed to encrypt a data as function of size of data. The graph presents the time for primes of size $200, 300, 400$bit.

We can see that the time for encryption grow linear with the data size. This result is obvious since for each block of encryption we using constant number of operations.

Also the bigger the prime the larger time it takes to encrypt the data since the arithmetics need to handle with bigger numbers.

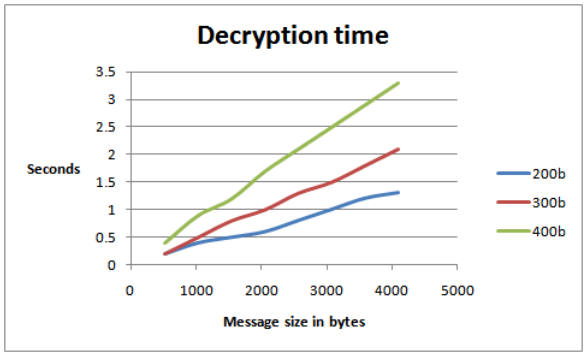


FIGURE 11.3.

This graph presents the time that needed to decrypt data as function of size of data. The graph presents the time for primes of size $200, 300, 400$bit.

Like in the encryption the growth is linear from the same reasons, Also we can see that to decrypt the same amount of data in takes half of the time of the encryption as was explained before.

12. Challenges

We were given two challenges during this project. We present those challegnes here, and their solutions:

12.1. **Challenge 1.**

Description:: Using CM method, construct 2 elliptic curves over 2 twin primes (each at least 400 bits long), with prime number of points each. Select a point at each curve so that the corresponding coordinates of the points are the same. Formally, give an example of the following construction:

- Two twin primes $p < q$ s.t. $\log p \geq 400$.
- Elliptic curves $E(F_p),\ E(F_q)$ s.t. $\#E(F_p)$ and $\#E(F_q)$ are both primes.
- Two points: P_p on $E(F_p)$ and P_q on $E(F_q)$, s.t. $x(P_p) = x(P_q)$ and $y(P_p) = y(P_q)$.

Solution::

- First we find two such primes, $p, p+2$.
- We use Atkin-Morain(The main algorithm behind our project) to find curves with a prime order on F_p and F_{p+2}. Those are denoted $E(F_p),\ E(F_{p+2})$.
- We find any point on $E(F_p)$, (X, Y).
- We then wish to find a curve, isomorphic to the second curve, s.t. (X, Y) (Both are less than p, and so, they are less than $p+2$) is a point on it. The second curve is $y^2 = x^3 + Ax + B$. The isomorphic curve is $y^2 = x^3 + z^2Ax + z^3B$ for a residue z(that is $z = s^2$ for some s). Putting in X and Y, we get a cubic equation in z. There is a neat way to solve cubic equations of the form $z^3 + pz - q = 0$, so we first want to transform it to this form. We take $u = z^{-1}$, and we get

$$\begin{aligned} X^3 + z^2AX + z^3B - Y^2 &= 0 \\ X^3 + u^{-2}AX + u^{-3}B - Y^2 &= 0 \\ u^3\left(X^3 - Y^2\right) + uAX + B &= 0 \\ u^3 + u\frac{AX}{(X^3 - Y^2)} + \frac{B}{(X^3 - Y^2)} &= 0 \end{aligned}$$

- As for solving $z^3 + pz + q$, we use the substitution $z = w - \frac{p}{3w}$. Then, it is reduced to the quadratic equation in w^3, $w^3 - \frac{p^3}{27w^3} - q = 0$. After finding a solution for this w^3, we calculate a third root(actually, we find a root of the poly $x^3 - w^3 = 0$). When we get z, we make sure it's a residue, otherwise it is a twist, not an isomorphism. We might have to pick a few extra random points for (X, Y) until we get a solution with z being a residue. When we find a solution, we win.

Result::

The following are the numbers found:

- P: 668891860309574793850043285804405323428824125067026597585546349063220693410084756966860999920121250714748512293206977069
- A1: 27443352163348964720038072740788959818367131678144207324753030844349393588601869758976920998651042251345432787017719588 7
- B1: 52638387023464448927856270255493534687523918717902458560815014072927974682706498742527124794902601428055923493842223003 7
- A2: 57903841153112154050650011588082014759350564588737176335490884009134078872546573730748019518096066386202918443628861390 7
- B2: 35417876698431654956130299389795672977138323926427939070701976565237878752016822116111493139720788254591053869456798579 8
- X: 27767555992863883965827815563426246813835798595229360000121375543420808116099646725845527501499229021955540585666042581 8
- Y: 65849547569740227105611913812656173309569999461456466772407005458495958670887620518104930216713621352354120850814014606 7
- Order1: 668891860309574793850043285804405323428824125067026597585547192044576659572674720284188924997951536775947954528790208903
- Order2: 668891860309574793850043285804405323428824125067026597585544714778345187916310256469159921843223962161317142953094469747

12.2. **Challenge 2.**

Description:: The setup is a prime field with p being the 13th Mersenne prime: $2^{521}-1$, 6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151.

(1) Can you construct 2 elliptic curves, which are twists of each other, such that their A coefficients are negations of each other, while their B coefficients are inverses of each other? Disregard the case where either A or B are zero.
(2) Can you construct 2 elliptic curves, which are twists of each other, such that their A coefficients are inverses of each other, while their B coefficients are negations of each other? Disregard the case where either A or B are zero.
(3) In both cases above, where construction is possible, give an example of a point with the same X, Y coordinates that belongs to both curves.
(4) [Harder] In question 3, can you construct 2 such curves with *the known order*, and so check the relation between twisted curves orders and prime field characteristic?
(5) [Hardest] Relax the condition of the p, instead, take any prime of 500 digits. Define the "almost-primality" of the pair of curves in question 4 as a multiply of small primes dividing the order of each curve. Construct 2 curves with the minimal almost-primality.

Solution::

12.2.1. *Task 1.* Let us have a curve over $\mathbb{Z}_p$, $y^2 = x^3 + Ax + B$ and its twist $y^2 = x^3 + s^2Ax + s^3B$ for a non residue s. We want $s^2A = -A$ and $s^3B^2 = 1$(All in $\mathbb{Z}_p$ of course). We get $s^3 = B^{-2}$(B is non zero since we require it to have an inverse). Let α be a cyclic generator of $\mathbb{Z}_p^*$, $s = \alpha^k$ (since its a non residue, k is odd), and $B^{-1} = \alpha^m$. We have $\alpha^{3k} = \alpha^{2m}$, or $3k = 2m\,(mod\ p-1)$. Since $p-1$ is even, $3k$ and $2m$ have the same parity, although one is odd, and one is even, which is a contradiction. So, it is not possible.

12.2.2. *Task 2.* Let us have a curve over $\mathbb{Z}_p$, $y^2 = x^3 + Ax + B$ and its twist $y^2 = x^3 + s^2Ax + s^3B$ for a non residue s. We want $s^2A^2 = 1$ and $\left(s^3+1\right)B = 0$. From the first we get $A = \pm s^{-1}$. From the second, since this is an integral domain, either $B = 0$, or $s^3 = -1$. We exclude $B = 0$. Since $p = 1\,(\mathrm{mod}\ 3)$, we have three solutions to $s^3 = -1$, namely $-1, r, r^{-1}$. Since $\frac{p-1}{2}$ is odd, those are non residues. So, we get 4 possibilities:

- $y^2 = x^3 \pm x + B$ and $y^2 = x^3 \pm x - B$
- $y^2 = x^3 \pm r^{-1}x + B$ and $y^2 = x^3 \pm rx - B$

The other possibilities give the same pairs.

12.2.3. *Task 3.* We saw that the first case is impossible. For the second, these are the possibilities:

- $y^2 = x^3 \pm x + B$ and $y^2 = x^3 \pm x - B$. Here we require $2B = 0 \pmod{p}$, which is not possible.
- $y^2 = x^3 \pm r^{-1}x + B$ and $y^2 = x^3 \pm rx - B$. Here, we have $\pm r^{-1}x + B = \pm rx - B$, or $x = \pm\frac{2B}{r-r^{-1}}$. We need to plug in those x values and check if $x^3 \pm rx - B$ is a residue.

The following example we found using the algorithm we had in question 4:

- A: 61883019243881434384216853682685789040188610650242732493236376563279868929695340287270017215604948338592024495717282145117297455180883516897404325126786674622
- B: 24489861543167068906071073384861091633618832349318542021547278213027488333937000020571626016001396071811742592279829056082442650580337875194993076848851482929
- X: 63632642784932508604794798136119666328674410002466794801245790927046249925394978086618232854945532865846815446962656547679574430802877711177748757957007156572
- Y: 49691662927046786881490604612716248568769300672600870706394088853909796469301236873257561925637838553255395841543560437469498483772176988837205601494045388742
- R: 61883019243881434384216853682685789040188610650242732493236376563279868929695340287270017215604948338592024495717282145117297455180883516897404325126786674622
- p: 68647976601306097149819007990813932172694353001433054093944634591855431833976560521225596406614545549772963113914808580371219879997166438125740282911150571511
- Order: 686479766013060971498190079908139321726943530014330540939446345918554318339765902063990033837342731571993813934591855431833976590206399003383734273157199381392440005929520745804478578749955656117276894944004

12.2.4. *Task 4.* The algorithm in this task is as follows:

- We are given p (Our prime), and a desired order $O = p + 1 + u$. We want to find a solution for $4p = u^2 + |D|\, v^2$, or $4p - u^2 = |D|\, v^2$. We can do it in the following way: We have a number $4p - u^2 = w$. w split into $x \cdot y^2$ where x is square-free. since we want x to be our discriminant, and we want it to be bound by some low number(Calculating Weber polynomials with huge discriminant is absurd). So, we can start dividing w by low numbers, until we get a square(we have a quare root algorithm, so it's not hard). When we do find such a solution, we can use either Hilbert polynomial or Weber polynomial to find a curve with the right order.
- Now, we want to find an isomorphism of this curve to the form shown in the second case in the solution for Task 3. So, we find r, a non-trivial cubic root of -1. The curve found is $y^2 = x^3 + Ax + B$. We want to find s, s.t. $As^4 = \pm r$ or $As^4 = r^{-1}$. When we do (which happens more often than

not), we make the isomorphism, we take $x = \pm\frac{2B}{r-r^{-1}}$, as explained in the solution to Task 3, and we check if $x^3 \pm rx - B$ is a residue. If so, we find y, we that's it.

12.2.5. *Task 5.* The theoretical side is a bit less complicated here. We want to find a prime p of size 500bit, and u s.t. $p+1+u$ and $p+1-u$ are both strong orders. When we do, we apply the algorithm in Task 4. We do it as follows: We first pick two randomly chosen strong orders with the same parity, $O_1 < O_2$. We find their average, $p+1$. We then have $O_1 = p+1-u$ and $O_2 = p+1+u$. We check wether p is indeed a prime. Since the distribution of prime numbers is approximately $\frac{1}{\ln X}$, and we have 500 bits in our number, then this probability is more than $\frac{1}{500}$, so we should find a prime in approximately 500 steps, When we do, we use the algorithm in Task 4.

References

[1] Guide To Elliptic Curve Cryptography – Hankerson
[2] Prime numbers a computational perspective - Crandall
[3] Elliptic Curves Number Theory and Cryptography - Lawrence C. Washington
[4] On the Use of Weber Polynomials in Elliptic Curve Cryptography – Konstantinou, Stamatiou , Zaroliagis
[5] Elliptic curves and primality proving – Atkin, Morain
[6] A Course in Computational Algebraic Number Theory - Henri Cohen
[7] On the Efficient Generation of Elliptic Curves over Prime Fields - Konstantinou, Stamatiou , Zaroliagis
[8] IEEE P1363 / D13
[9] Generating Elliptic Curves of Prime Order - Erkay Sava, Thomas A. Schmidt, and Cetin K. Koc
[10] www.wikipedia.org
[11] http://mathworld.wolfram.com/ClassNumber.html
[12] http://content.digitalwell.washington.edu/msr/external_release_talks_12_05_2005/16447/lecture.htm