A DISCRETE LOGARITHM ATTACK ON ELLIPTIC CURVES

OTTO JOHNSTON

ABSTRACT. We give an improved index calculus attack for a large class of elliptic curves. Our algorithm works by efficiently transferring the group structure of an elliptic curve to a weaker group. The running time of our attack poses a significant and realistic threat to the security of the elliptic curves in this class. As a consequence of our construction, we will also derive entirely new point counting algorithms. These algorithms set new run-time complexity records. We discuss implementations of these algorithms and give examples.

1. INTRODUCTION

A discrete logarithm problem for a group G simply asks for the computation of x in a pair (g, g^x) for $g \in G$. In most cases, we want G to be efficiently representable, finite, and *almost* of prime order. This almost of prime order property means that G should contain a large prime order subgroup relative to its size. Such a subgroup is necessarily cyclic, and if G is very large, then efficiently computing discrete logarithm instances cannot be accomplished by using general finite group theory (e.g., Sylow's theorems, Chinese Remainder Theorem).

On a high level, what we look at is the case where we have another group H and a surjective homomorphism $f : H \to G$. In this case, there exists an $h \in H$ such that $f(h) = g$ and consequently $f(h^x) = g^x$. Thus, computing x for a pair (h, h^x) is at least as difficult as computing it for (g, g^x) provided that the difficulty of computing f^{-1} does not get in the way. In the case of elliptic curve cryptography, the group G is replaced by an elliptic curve E. Only the group structure of this curve is important, so one can think of E as a new letter for the group G. The group E is always efficiently representable and finite, and it is sampled so that it is almost of prime order. Most importantly, the computation of discrete logarithm problems on E is a notoriously difficult problem that has resisted decades of attacks. What we show in this paper is how to efficiently construct a new group H such that there is a surjective homomorphism $H \to E$ with an efficiently computable inverse. We then show how to use the structure of H to make it easier to compute discrete logarithms for a large family of elliptic curves.

1.1. Historical Overview of Index Calculus Attacks on Elliptic Curves.

We now give a brief historical overview of the results we will need. Since elliptic curves are centuries old and the central topic of numerous papers each year, our overview is far from complete. See [33] for a more complete survey. For general elliptic curves, the best known algorithm to compute discrete logarithms is the Pollard-Rho algorithm, which computes the discrete logarithms of a group G in time equal to $O(\sqrt{p})$, where p is the largest prime factor of $\#G$. There have been numerous efforts to do better than this when G is an elliptic curve. J. Silverman in [34] was the first to propose an index calculus variant for elliptic curves that was hoped to have a faster running time than Pollard-Rho. Very shortly thereafter, this approach was shown to be slower (see [17]). Roughly a decade earlier, N. Koblitz had proposed a generalization of elliptic curve cryptography to hyperelliptic curves in [21]. This system was brought into question by P. Gaudry in [14], where it was shown that index calculus for hyperelliptic curves worked with a remarkable advantage: in [14], he was able to run his algorithm to calculate a challenge discrete logarithm problem using only modest hardware. Once P. Gaudry proposed his attack against hyperelliptic curves, many generalizations quickly followed (see [7], [9], [16], [35]).

Although Gaudry's methods did not seem to have any direct advantage when used for elliptic curves, an attack against elliptic curves surfaced anyway. The idea was to *transfer* the group structure of an elliptic curve to one of the weaker hyperelliptic curves. The idea goes as follows. The running time of the generic discrete logarithm attacks against an elliptic curve over $\mathbb{F}_q$ is roughly $O(\sqrt{q})$, whereas the running time of index calculus on a hyperelliptic curve over the same field is roughly $O(q^{2-2/g})$ for $g \geq 2$. This second attack does not help at all, unless we had a situation where the elliptic curve, call it E, was defined over $\mathbb{F}_{q^n}$ and the hyperelliptic curve, call it C, was defined over the subfield $\mathbb{F}_q$. If the group structures related to E and C were isomorphic and if $n > 2$, we could gain a

Date: November 11th, 2010.

computional advantage by using index calculus on the hyperelliptic curve if we could compute this isomorphism in reasonable time. This is the basic idea behind the so-called *descent* theory of index calculus on elliptic curves. This descent procedure stimulated a great deal of investigation (see [12]) and resulted in a remarkable reduction in the security of many elliptic curves over even extensions of finite fields in characteristic 2. However, in odd characteristic, this attack was more difficult.

1.2. **The mathematics behind the descent of index calculus.** The main difficulty in transferring a discrete logarithm problem from an elliptic curve to a higher genus curve is at the heart of a very deep problem in algebraic geometry. From a computational point of view, whether one is interested in point counts or cryptographic applications, a curve serves as a platform to construct an object known as a Jacobian. The Jacobian is the real object of interest in virtually any application point of view; the curve merely serves as a method to construct it. A Jacobian lives in a more general class of objects known as abelian varieties. A very difficult and fundamental problem in algebraic geometry is to understand the interplay between the Jacobians and the abelian varieties that are not Jacobians, and, most importantly for us, a deep problem asks about surjective homomorphisms with small kernels between these objects.

The surjective homomorphisms we are looking for most naturally come from the theory of isogenies. An isogeny is a morphism between abelian varieties that is both surjective and has a finite kernel. These maps give us more freedom in trying to deform the elliptic curve into the Jacobian of a higher genus curve without destroying the underlying discrete logarithm problem. In particular, if an isogeny $f : A \to B$ has a kernel in some part of A that is small, say a few elements of order 2, then one could hope that by studying f one could transfer discrete logarithms with more freedom than if we restricted to isomorphisms. In geometric language, we would be studying the so-called isogeny classes.

1.3. **Our results.** The theoretical contributions of this article are two-fold. The first is a technical result: we prove that under a certain condition, if A is an abelian variety that is isogenous to the Jacobian of a hyperelliptic curve, then so is A^2. The second contribution is that these constructions are computationally efficient. These two results are then applied to two extremely active problems in computational algebraic geometry: the computation of discrete logarithms on elliptic curves and point counting algorithms for hyperelliptic curves.

1.3.1. *Implications to cryptographic schemes.* The main contribution of this paper is to give the first new class of ordinary elliptic curves where index calculus becomes a practical threat. We will show how to sample from this family efficiently. We require no extra side information to apply our attack, nor do we need special oracles or structure information about the elliptic curves we sample. Moreover, every member of the family we create will be vulnerable to our descent attack. We show how to use this construction to build a large and explicit family of elliptic curves where a factor basis for index calculus can be found and used to compute discrete logarithms in time roughly $O(q^{3/8})$, where the generic attacks on the same curve would run in time roughly $O(q^{1/2})$.

The implications to actual in-use schemes are the following. There are three ways for a given security parameter $k \in \mathbb{Z}^+$ to influence the group size of a family of elliptic curves. The first is to use k as the bit-length of a prime p and define elliptic curves over $\mathbb{F}_p$, the second is to fix p and define elliptic curves over $\mathbb{F}_{p^k}$, and the third is some combination of the first two. It was previously known that if we took $p = 2$ and $k > 2$ to be a power of 2 that there were elliptic curves over $\mathbb{F}_{2^k}$ that were weak in the sense described above (see [12]). We extend this result to all primes p. Our results show that elliptic curve cryptography must take special care when k is taken as an exponent of *any* fixed prime or if k influences both the prime size and its exponent. Both of these statements are new.

The explicit family of weak elliptic curves we will construct contains instances of virtually all elliptic curve classes proposed in the literature. Most importantly, if we sample random curves from this family, we find elliptic curves with large prime factors that have not been previously ruled out as cryptographically weak. We will give examples of elliptic curves with a group size in the 160 bit range with a single large prime factor that can be attacked with a substantial advantage over the generic algorithms.

The larger question concerning how many elliptic curve schemes have a non-negligible number of weak elliptic curves is more difficult to answer. In essence, this is a distribution question, and very little is understood about the distribution of the group structure of families of elliptic curves over finite fields; in fact, even the distribution of their orders is unknown (see [13] for conjectures). If used directly, our families are small and will most likely not be sampled at random from most of the proposed schemes. However, in this paper we identify a condition that allows us to descend an elliptic curve to a smaller field. If an elliptic curve E that did not directly satisfy this property

had a surjective map (with a small kernel) to a curve E' that did satisfy this special property, then E would be as vulnerable as E' to the attack we outline in this paper. There is computational evidence that an enormous increase in the number of weak curves might be found in this way, but due to the incomplete picture of how elliptic curve families are distributed among their associated groups, we cannot offer any proofs.

1.3.2. *Point Counting.* A very difficult problem in computational algebraic geometry is to generate large families of curves with a known number of points over a finite field $\mathbb{F}_q$. The parameters one usually wants to consider are q and an invariant of the curve known as its genus. One should view the genus g and the number q as integers that specify intrinsic properties of C. In order to hope for efficient computations of the number of points of C, one must restrict the class of curve in some way.

The current state of the art algorithm for generating genus 2 curves with a known number of points is that given by Satoh in [30]. Satoh's algorithm works by taking two isomorphic elliptic curves and glueing a genus 2 curve from these equations. Relaxing the condition of isomorphism to that of isogeny, we are able to make a new algorithm for this purpose. Our algorithm is superior to Satoh's in the following ways. (1) It outputs the group order rather than merely the largest prime factor. (2) It outputs the group order *with certainty*, rather than merely with some probability that is not well-understood. (3) It allows for a much wider distribution of curves, in contrast to Satoh's algorithm which is restricted to even orders. In particular, our algorithm can be used to find prime-order Jacobians. We also drastically reduce the complexity of Satoh's algorithm from $O(\log^2(q))$ operations over $\mathbb{F}_q$ to $O(\log(q))$ over $\mathbb{F}_q$.

We are also able to propose entirely new point counting algorithms based on this work. One such extension is an algorithm that generates genus 4 curves along with the order of their Jacobian at the cost of running a point counting algorithm on a single elliptic curve. This algorithm is a significant improvement over existing point counting algorithms (see [29]). In fact, we will be able to compute the entire Zeta function for the genus 4 curve, which essentially classifies the genus 4 curve's arithmetic properties.

As a final application to point counts, we will show how to generate a large class of hyperelliptic curves of genus $2g$ over $\mathbb{F}_q$ with exactly $q+1$ points in $O(g\log(q))$ operations over $\mathbb{F}_q$. To generate these curves, we only require a positive integer g and an odd prime power q as an input to our algorithm. This algorithm remains efficient if g grows linearly and q grows exponentially, which is unlike any previous curve generation algorithm for a family as large as the one we will consider (for previous work, see [29], [19]).

1.4. **Organization.** In the next section, we will collect the results we need at a high level to run our attack. We will then proceed to prove them in detail. In Section 3, we will write out our point counting algorithms using the material in Section 2. In Section 4, we give an algorithm that efficiently computes the transfer map explained earlier in this introduction and we outline the discrete logarithm attack. In Section 5, we discuss the distributions of the elliptic curves we sample. We conclude with a link to implementations of our algorithms.

2. A DESCENT OF HYPERELLIPTIC CURVES

2.1. **An outline.** Before we begin with a formal discussion, we will give an outline of what we prove and how we use it. We will first fix a finite field k' in odd characteristic. In this discussion, we have two objects. The first are hyperelliptic curves. On the level of equations, a hyperelliptic curve is given by a formula of the form $y^2 = f(x)$, where f is a separable polynomial in the variable x. The second class of objects we deal with are Jacobians of hyperelliptic curves. These objects can be thought of as unpacking the "group structure" of the equation for C. For instance, in elliptic or hyperelliptic curve cryptography, the curve serves only as a means to efficiently represent the group structure of its Jacobian. It is in this group that all of the calculations are done. In coding theory, this unpacking procedure is used to construct vector spaces with interesting parameters.

In our setting, we will let k be a quadratic extension of k'. It is well known that k can be formed by taking the square root of a non-square in k', and we will let w be this square root. If we let C be the hyperelliptic curve defined by

$$y^2 = (x-w)f(x),$$

we can show that there is another hyperelliptic curve C' of the form

$$y^2 = g(x)$$

defined over k'. The curve C' has the special property that the Jacobian of C has a homomorphism to the Jacobian of C' on the level of groups. This map, call it δ, is almost an isomorphism: in Theorem 2, we show that it is an isomorphism modulo points of order 2. Another critical property is that δ is computationally efficient to compute. These two results give us a descent. We are able to construct this C' from Theorem 1, and we can read off the polynomial $g(x)$ above from Corollary 1.

The map δ is almost all we need for the discrete logarithm attack. To make the attack practical, we also need to know the order of the Jacobian of C'. This takes a little more effort, but Corollary 3 gives us this order. Our strategy is to then find elliptic curves E and to build the curves C'. Once we have C', we try to apply the descent again. If we can descend once more, we then have a computationally efficient map from E to the Jacobian of a genus 4 hyperelliptic curve. Since index calculus works with a remarkable advantage with these curves (see [35]), we have successfully attacked E.

2.2. **The mathematics behind the descent.** We will now proceed to prove these ideas, but as we move toward the details we are forced to become more precise. We will let k be a finite field in odd characteristic. By a *curve* over k, we will mean a projective, geometrically irreducible variety of dimension 1 over k that has at least one non-singular rational point. We do not assume our curves are non-singular. Given a curve C, we will let $\widetilde{C}$ denote the normalization of C, and we will let any arrow $\widetilde{C} \to C$ reference the implicit normalization map whenever it is needed. For a curve C, we will let $\mathrm{Jac}(C)$ denote the Jacobian of $\widetilde{C}$. When a curve is specified by an affine equation, we will always take the projective closure. A *hyperelliptic curve* is a curve that admits a double covering of a conic.

Our first task is to glue two hyperelliptic curves to produce a new curve. Although this is a classical result, we will reprove it with an emphasis on deriving an affine equation for the new curve. In the next series of proofs, we will continue to keep track of equations. Eventually, this will lead us to the new algorithms in the later sections. We will assume that all of our fields are finite fields, even though many of these results are more general.

Lemma 1. Let $C_1 : y^2 = f$ and $C_2 : y^2 = g$ be two hyperelliptic curves defined over k, where f and g are separable polynomials in $k[x]$ with $h = fg/(\gcd(f,g))^2 \notin k$. If we let $C_3 : y^2 = h$, then the curve

$$C : y^4 = 2(f+g)y^2 - (f-g)^2 \tag{2.1}$$

has the property that

$$\mathrm{Jac}(C) \sim \prod_{i=1}^{3} \mathrm{Jac}(C_i).$$

Proof. In order for C to have a function field, it must be geometrically irreducible; by inspecting the roots of the equation for C in $\overline{k(x)}[y]$, we see that this implies that $\sqrt{fg}$ is not in $\overline{k}(x)$, which is where $h \notin k$ is used. Let the function field of C_1 be $k(x)[y_1]$ and let the function field of C_2 be $k(x)[y_2]$. If we take the compositum, $k(x)[y_1, y_2]$, then $y = y_1 + y_2$ is permuted by the Galois actions of $\mathrm{Gal}(k(x)[y_1]/k(x)) = \{1, \psi\}$ and $\mathrm{Gal}(k(x)[y_2]/k(x)) = \{1, \phi\}$. This implies that $k(x)[y_1, y_2] = k(x)[y]$. It is easy to see that the minimal polynomial $\mathrm{Irr}_{k(x)}(y)$ is the equation for C given above: it is simply $\prod(y \pm (y_1 \pm y_2))$. Now, $\mathrm{Gal}(k(x)[y]/k(x)) = \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$, and it is obvious that $C/\psi \cong C_2$, $C/\phi \cong C_1$, and $C/(\psi\phi) \cong C_3$. We apply Theorem 5 of [18] to obtain the k-isogeny

$$\mathrm{Jac}(C) \sim \prod \mathrm{Jac}(C_i).$$

□

We will construct a new hyperelliptic curve that captures the arithmetic data of a smaller genus hyperelliptic curve. Our new hyperelliptic curve will be defined over a smaller field, and it is from here where our descent from smaller genus curves to larger genus curves begins.

Let C be any curve over k and let k/k' be a finite extension. Give C an affine equation $\sum_{i,j} c_{i,j} x^i y^j = 0$ as before, where only the $c_{i,j}$ are in k. For any automorphism $\sigma \in \mathrm{Aut}(k/k')$, let $\sigma(C)$ denote the smooth, projective curve with affine equation $\sum_{i,j} \sigma(c_{i,j}) x^i y^j$. In order for us to allow σ to act on points of C, we need to work in a larger ambient space. To extend σ, associate each closed point $P \in \mathbb{P}^2_k$ to its corresponding prime ideal I in the standard affine cover; let $\sigma(P)$ be the closed point associated to the prime ideal $\sigma(I)$. All points on curves will now be considered in the ambient space of $\mathbb{P}^2_k$ (this gives meaning to points σ maps off a curve). Finally, let $\Gamma(\cdot)$

denote the forgetful functor $\Gamma : \mathrm{AbSch}/k \to \mathrm{Ab}$ from the category of abelian group schemes over k to the category of abelian groups: it simply takes an abelian group scheme to its underlying group.

Lemma 2. If C is any curve defined over k such that there exists a subfield $k' \subset k$, then for any $\sigma \in \mathrm{Gal}(k/k')$, we have $\mathrm{Jac}(C) \sim \mathrm{Jac}(\sigma(C))$. Moreover, σ extends to an isomorphism $\Gamma(\mathrm{Jac}(C)) \cong \Gamma(\mathrm{Jac}(\sigma(C)))$.

Proof. For any finite extension $k \subset l$, if we have an l-rational point P of C, then $\sigma(P)$ is an l-rational point of $\sigma(C)$ by definition. Thus, σ establishes a bijection between the rational points of C and $\sigma(C)$ over any finite extension of k. This implies that they have the same Zeta function and hence the same L-polynomial. Since the L-polynomial classifies an isogeny class (see Theorem 2.c, Appendix 1, [27]), our result follows.

Any divisorial correspondence $D \sim D'$ comes from a rational function $f \in k(C)$. If we write $D = \sum P_i$ and let $\sigma(D)$ denote $\sum \sigma(P_i)$, we see at once that $\sigma(D) \sim \sigma(D')$ via $\sigma(f)$, where, again, $\sigma(f)$ is simply σ applied to the coefficients of f in k. Clearly, composing σ with σ^{-1} gives us the identity; since it is obviously homomorphic, we have an isomorphism $\Gamma(\mathrm{Jac}(C)) \cong \Gamma(\mathrm{Jac}(\sigma(C)))$. □

Example 1. The map σ above is *not* typically in the image of the Picard functor. In other words, $\mathrm{Jac}(C)$ and $\mathrm{Jac}(\sigma(C))$ are not usually isomorphic as varieties. It is this key property that gives us a very large distribution to sample from, and it is one of the main features where our approach differs from other work. Essentially, we have two Jacobians that are isomorphic as abelian groups but not as varieties. To make sense out of what this means, some examples are in order. Consider the finite field $k = \mathbb{F}_5[w]$, where $w^2 = 2$. Define the elliptic curve

$$E_1 : y^2 = (x - w)(x^2 - x).$$

By our construction ($k' = \mathbb{F}_5$ and σ being absolute Frobenius),

$$E_2 = \sigma(E_1) : y^2 = (x + w)(x^2 - x).$$

The previous result might lead one to think that the Pic^0 functor is represented by the same abelian group scheme for both curves, and hence isomorphic (by the universal property of representability). This would indeed be the case if σ was in the image of the Pic^0 functor. But for E_1 and E_2 to be isomorphic as varieties their j-invariants would have to be the same and this is not the case: $j(E_1) = -w + 2$ and $j(E_2) = w + 2$. What is happening here is that σ defines an isomorphism in the category of abelian groups, but this isomorphism has no pullback to the category of abelian group schemes *over* k. Note that σ does not fix the base k, which is at the heart of this construction. (It is also worth mentioning that σ acts on the entire j-line exactly as one would expect – for instance, notice that $\sigma(j(E_1)) = j(E_2)$ – but we will not develop this further.) One might also compare this with the discussion in 3.2.4 of [23].

Of particular interest to us is when we have a quadratic extension k over k'. To ease notation, let $k = k'[w]$ for $w^2 \in k'$ be this quadratic extension. Let σ denote the non-trivial element in $\mathrm{Gal}(k/k')$, which is just the q-th power Frobenius where q is the size of k'. Let $C \to \mathbb{P}^1_k$ be a double covering of curves over k with ramification divisor R (i.e., C is a hyperelliptic curve with a fixed covering of $\mathbb{P}^1_k$). Recall that R consists of the sum of those closed geometric points on $\mathbb{P}^1_k$ where the inverse image of $C \to \mathbb{P}^1_k$ contains exactly one point. Since k is not closed, some of these geometric points may form a higher degree place over k.

Theorem 1. Let C be as above. If there exists a k-rational point $r \in \mathrm{Supp}\,(R)$ such that $\sigma(r) \notin \mathrm{Supp}\,(R)$ and $\sigma(\mathrm{Supp}\,(R) - \{r\}) = \mathrm{Supp}\,(R) - \{r\}$, then $\mathrm{Jac}(C)^2$ is k-isogenous to the Jacobian of a hyperelliptic curve C' that is defined over k'.

Proof. Under the hypothesis of the theorem, we can write C as

$$C : y^2 = (x - w)f,$$

where $\sigma(f) = f \in k'[x]$. By Lemma 1 we have a curve C' that covers three curves: the first is C, the second is the curve with affine equation $y^2 = (x + w)f$, and the third is the conic $y^2 = x^2 - w^2$. Since it is clear from the proof of Lemma 1 that C' has an involution such that the quotient is this conic, we have that C' is a hyperelliptic curve by definition. Moreover, it has an affine equation

$$C' : y^4 = 4xfy^2 - 4w^2f^2, \tag{2.2}$$

by Lemma 1 which is fixed by σ, and hence defined over k'. Lemma 2 and the last statement of Lemma 1 gives us the k-isogeny $\mathrm{Jac}(C') \sim \mathrm{Jac}(C)^2$. □

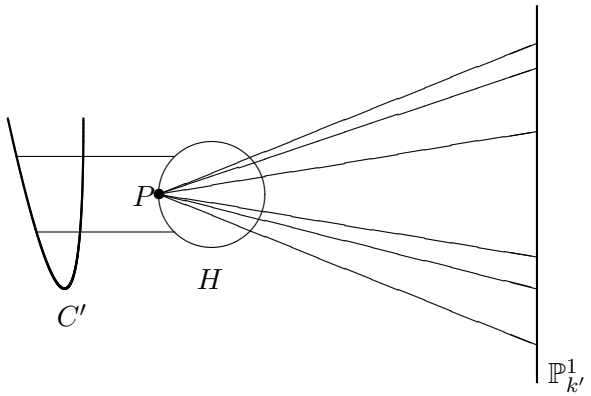

Figure 1. Projection of H.

Our next problem is of a computational nature. The hyperelliptic curve C' defined in (2.2) has two undesirable features: the first is that it is a singular curve and the second is that it is not of the form $y^2 = h(x)$ for h a separable polynomial in a single variable x. It is not practical to work on a curve of the form C' is currently written in. Moreover, for our discrete logarithm transfer, we need to write C' in the form $y^2 = h(x)$ so that we can apply our theorems recursively. So, we look at ways to re-write (2.2) into the form $y^2 = h(x)$.

The standard way of doing this is to find a rational point P and to compute a basis for the Riemann-Roch space $2P$. This gives us a map, known as a pencil, from $C' \to \mathbb{P}^1_k$ that is also a double cover. It is a standard result from here to derive an affine equation $y^2 = h(x)$ for C' (see [1], I.6.A). However, we have a more efficient method; we will apply 19th century geometry that is rarely used to find algorithms.

We begin by looking at how Theorem 1 constructed the hyperelliptic curve C'. The reason C' is hyperelliptic is that it covers the conic $H : y^2 = x^2 - w^2$. Now, if we look a bit closer at H, we see that it is a circle. We can find a k'-rational point on this circle that is independent of the choice of k or k': namely, the point $P = (x_1, y_1)$ given by $x_1 = (-w^2 - 1)/2$ and $y_1 = (w^2 - 1)/2$. If we use this P as a base point on our circle, we can unwrap the circle into a line. Doing so (see e.g. page 7 of [32]), we obtain an isomorphism $\mathbb{P}^1_{k'} \to H$ given (locally) by sending a fixed parameter t for $\mathbb{P}^1_{k'}$ to

$$x = \frac{w^2 + 1}{2} - \frac{t(w^2 t + t + w^2 - 1)}{t^2 - 1} \tag{2.3}$$

$$y = \frac{w^2 - 1}{2} + t\left(w^2 + 1 - \frac{t(w^2 t + t + w^2 - 1)}{t^2 - 1}\right). \tag{2.4}$$

These formulas let us replace H with $\mathbb{P}^1_{k'}$ in the proof of Theorem 1. The curve C' now covers this $\mathbb{P}^1_{k'}$ via this isomorphism (see Figure 1). Returning to the proof of Lemma 1, we can compute the equation of C' that covers H: this is just $(Y - y_1 + y_2)(Y + y_1 - y_2)$ for $y_1 = \sqrt{(x - w)f}$ and $y_2 = \sqrt{(x + w)f}$. Further computation (again using the x and y we computed above) reveals that $(Y - y_1 + y_2)(Y + y_1 - y_2) = Y^2 - 2f(x - y)$. Now we have that C' covers H by $Y^2 = 2(x - y)f(x)$, but the right hand side is not yet a polynomial. To clear denominators, we replace Y by $Y(t - 1)/(t^2 - 1)^d$ where $d = g(C')/2 + 1$. Collecting this into a single statement, we have the following.

Corollary 1. The curve C' in Theorem 1 can be written as

$$Y^2 = \frac{2(x - y)f(x)(t^2 - 1)^{g(C')+2}}{(t - 1)^2}, \tag{2.5}$$

where x and y are given in (2.3) and (2.4) respectively. The right hand side is a polynomial in $k'[t]$.

The equation for C' given in (2.5) is non-singular outside of points at infinity. This is in contrast to the equation for C' given in Theorem 1. We will continue to let C' denote the singular (projective) curve with affine equation given in Theorem 1.

What we have collected so far are the basic parts of a new algorithm. At the moment, we can take a hyperelliptic curve C of a special form over a quadratic extension k/k' and we can compute a new hyperelliptic curve C' over

k' that has a k-isogeny $\mathrm{Jac}(C') \sim \mathrm{Jac}(C)^2$. What we need to do next is to determine the relationship between the groups $\Gamma(\mathrm{Jac}(C(k)))$ and $\Gamma(\mathrm{Jac}(C'(k')))$. We will find out that there is a homomorphism between these two groups that has a small kernel. In the next section, we will show how to compute this homomorphism efficiently.

Since our varieties are defined over varying fields, we will consider all curves as defined over a fixed algebraic closure of $\overline{k'}$ (formally, we replace C' by $C' \otimes \mathrm{Spec}\,\overline{k'}$). We will let σ denote the qth power Frobenius where q is the size of k'. For any geometric point $Q = (x_1, y_1)$ on a hyperelliptic curve, let $Q' = (x_1, -y_1)$ (this notation is shorthand for the hyperelliptic involution). We will let $(a, b_1 \pm b_2)$ denote the divisor $(a, b_1 + b_2) + (a, b_1 - b_2)$ on C'. In the case that b_2 is zero, we will simply add the same point twice. If C has two geometric points at infinity, we will let D_∞ denote the sum of them; otherwise, C has one geometric point P_∞ at infinity and we will let D_∞ denote $2P_\infty$. It is clear that if we define D'_∞ to be the same for $\sigma(C)$, then $\sigma(D'_\infty) = D_\infty$.

In the next result, we will look for a way to associate divisors of C over k with divisors of C' over k'. Intuitively, we want some form of a diagonal map on $\mathrm{Jac}(C) \times \mathrm{Jac}(\sigma(C))$ to stabilize the action of σ in the hopes of producing divisors on C' defined over k'. Recall that in the proof of Lemma 1, we had three curves: C_1, C_2, C_3, and the glued curve. We also fixed maps $\tau_i : C_i \to \mathbb{P}^1_k$ and, consequently, the map $\tau : C' \to \mathbb{P}^1_k$. In this section, we have that $C = C_1$, $\sigma(C) = C_2$, and $H = C_3$. Also recall from Lemma 1 that we have two double coverings $\pi_1 : C' \to C$ and $\pi_2 : C' \to \sigma(C)$. For the sake of notation, temporarily let π denote either of these maps. It is well known that $\pi_* \pi^* D = \deg(\pi) D = 2D$ for a divisor D, where $\deg(\pi) = 2$ comes from our choice of π (see IV.ex.2.6 of[15]). The map $\mu : \mathrm{Jac}(C) \times \mathrm{Jac}(\sigma(C)) \to \mathrm{Jac}(C')$ is given by $\pi_1^* \times \pi_2^*$ once we have normalized our singular curves. Likewise, the map $\mathrm{Jac}(C') \to \mathrm{Jac}(C) \times \mathrm{Jac}(\sigma(C))$ is given by $(\pi_1)_* \times (\pi_2)_*$ once we have taken the normalizations. We can see from direct calculation (using classical derivatives) that the affine singular points of C' lie over the ramified points of $\pi_1 : C \to \mathbb{P}^1_k$ which are sent to other ramified points of C by σ. By our construction, these points occur at exactly the roots of the polynomial f given in the definition of C. At these points, the maps π_i will have different tangent directions for the same point P and thus form a node on C'. The degree of the divisor summing up a blown up node on $\widetilde{C'}$ is 2 (this is made explicit in the proof of Corollary 1). Now we can prove our next result.

Theorem 2. There exist homomorphisms $\delta : \Gamma(\mathrm{Jac}(C)(k)) \to \Gamma(\mathrm{Jac}(C')(k'))$ and $\delta' : \Gamma(\mathrm{Jac}(C')(k')) \to \Gamma(\mathrm{Jac}(C)(k))$ such that the kernel of the composition $\delta'\delta$ consists of all elements of order 2 on $\Gamma(\mathrm{Jac}(C)(k))$.

Proof. Let $P = (x_0, y_0)$ be an affine k-rational point on C. We will show how to map P to a divisor on C' over k'. We start by defining the point $Q = (\sigma(x_0), y_1)$ on C lying over $\sigma(x_0)$. The point Q need not be k-rational, but it clearly must be defined in at most a quadratic extension of k. If P is ramified and x_0 is a root of f, then define the divisor D to be the sum of the blown up points over the points P and $\sigma(P)$ (which are defined on both C and $\sigma(C)$). This gives us a divisor D that is fixed by σ. Otherwise, define the divisor

$$D = (x_0, y_0 \pm \sigma(y_1)) + (\sigma(x_0), \sigma(y_0) \pm y_1))$$

on C' (that the points described on D are points of C' follows trivially from the second definition of C' in the proof of Lemma 1). If we look at this second definition of D, it is immediate that σ permutes x_0 with $\sigma(x_0)$ and y_0 with $\sigma(y_0)$ (since x_0 and y_0 are in k). As for y_1, which may be in a quadratic extension L of k, we know that $\tau_1(\sigma(\sigma(Q))) = \sigma(x_0)$, so $\sigma(\sigma(y_1))$ is either y_1 or $-y_1$ by Lemma 2. This shows us that σ permutes the summands of D, hence $\sigma(D) = D$. We now have that in both of our definitions of D (one for P singular and one for P non-singular) that $\sigma(D) = D$. Moreover, σ generates both $\mathrm{Gal}(k/k')$ and $\mathrm{Gal}(L/k')$, so D is fixed by the entire Galois action in either case. This implies that D is defined over k' (see Lemma 4, Appendix 1, [27]). By Lemma 2, we could replace C by $\sigma(C)$ and P by $\sigma(P)$ and still obtain the same divisor D. Thus, we have a map from degree 1 divisors of C to the divisors on C' given by $\delta : P \mapsto (P, \sigma(P)) \mapsto D$ (where D is defined above). Since we have that $\deg(D) = 4$ and we can translate $\mathrm{Pic}^4(C')$ to $\mathrm{Pic}^0(C')$ (over any base field), we can define δ as a map $\Gamma(\mathrm{Jac}(C)(k)) \to \Gamma(\mathrm{Jac}(C')(k'))$.

Now, if we push our divisor D back down to C (resp. $\sigma(C)$) in the case where P is ramified, we get $4P$ and otherwise, we get $(\pi_1)_* D = 2P + Q + Q'$ (resp. $(\pi_2)_* D = 2\sigma(P) + \sigma(Q) + \sigma(Q)'$). Subtracting $2D_\infty$ (resp. $2\sigma(D_\infty)$), we obtain $2P - D_\infty$ (resp. $2\sigma(P) - \sigma(D_\infty)$). This tells us that if $\alpha \in \Gamma(\mathrm{Jac}(C))$, then $(\pi_1)_* \delta(\alpha) = 2\alpha$. So, $\delta' = (\pi_1)_*$ is the map we need to finish the proof. □

Corollary 2. Using the notation above, #Jac$(C)(k)$ =#Jac$(C')(k')$.

Proof. In light of the previous result, we see that #Jac$(C)(k)$ and #Jac$(C')(k')$ are equal up to a factor of the form 2^n. One can show that the points of order 2 on a hyperelliptic curve $y^2 = P(t)$ are completely determined by the factorization of $P(t)$ over the base field (see [2]). In our case, if we look at the equation for C' given in Corollary 1 and write it as $C' : y^2 = P(t)$, we can see by elementary algebra that the linear factor $x - r_i$ of f (from the curve $C : y^2 = (x - w)f$) corresponds to the (blown up) factor $(t - 1)^2 + 2r_i(t^2 - 1) + (t + 1)^2w^2$ of P. Each factor of P is of this form, except if $\deg(f)$ is even, when P has a factor $(t^2 - 1)$ (corresponding to the blown up point at infinity). Now we have a simple relationship between the factors of f and the factors of P. If f is of odd degree d and has s factors, then P has s even degree factors and degree $2d \equiv 2 \bmod 4$. Otherwise, f is of even degree and P has $s + 2$ factors, one of which has odd degree (e.g. $(t - 1)$).

Let $H : y^2 = h(x)$ be a hyperelliptic curve over a finite field and suppose that $h = \prod_{i=1}^{s} h_i$, where each h_i is irreducible (assume that h is separable). We define the 2-rank of Jac(H) to be $\dim_{\mathbb{Z}/2}$ Jac$(H)[2]$ (this is the dimension of the $\mathbb{Z}/2$-vector space of all points of order 2 on Jac(H) over the finite field). In Theorem 1.4 of [5], we have the statement that the 2-rank of Jac(H) is $s - 2$ if $\deg(h)$ is even and some h_i has odd degree; the 2-rank is $s - 1$ if $\deg(h)$ is odd or if $\deg(h) \equiv 2 \bmod 4$ and all h_i have even degree; and the 2-rank is s otherwise. In our case, let $f = \prod_{i=1}^{s} f_i$ for irreducible f_i. If $\deg(f)$ is odd, then $(x - w)f$ is of even degree with an odd factor. In this case, Jac(C) has 2-rank $s - 1$. Since P has s even degree factors and degree $2d \equiv 2 \bmod 4$, we have that Jac(C') also has 2-rank $s - 1$. If $\deg(f)$ is even, then $(x - w)f$ has odd degree and 2-rank s. In this case, P has even degree and $s + 2$ irreducible factors (one of which is linear), so Jac(C') also has 2-rank s. □

Although it is tempting to conclude that Γ(Jac$(C')(k')$) and Γ(Jac$(C)(k)$) are isomorphic, this is rarely the case. What we can show instead is that there is a special abelian variety defined over k' that is isogenous to Jac(C'). Let W be the Weil restriction of Jac(C) (over k) to k'. It is known that W is an abelian variety of dimension $2g$ that is defined over k' (see Section 1 of [26]).

Corollary 3. There is a k'-isogeny $W \sim$ Jac(C'). If $L(t)$ is the L-polynomial of C over k, then $L(t^2)$ is the L-polynomial of C' over k'. Moreover, $\#C'(k') = q + 1$.

Proof. It is known that W can be defined as the abelian variety that represents the functor $\mathrm{Res}_{k/k'}\mathrm{Jac}(C)(S) = \mathrm{Jac}(C)(S \times_{k'} k)$ from k'-schemes$^{\mathrm{op}}$ to Sets. In particular, if we let $S = \mathrm{Spec}\, k_1$ for some finite extension k_1 of k', then we see that $\#W(k_1) =$#Jac$(C')(k_1)$ (by using Corollary 2 over base extensions). This gives us a k'-isogeny $W \sim$ Jac(C') (use Theorem 2.c, Appendix 1, [27]). Since the L-polynomial of W is $L(t^2)$ if L is the L-polynomial of C (over k), we see that $L(t^2)$ is the L-polynomial of C' over k'. Now, the number of k'-rational points for a curve is $q + 1 - \sum \alpha_i$ where the α_i are the roots of the L-polynomial of the curve over k' (see Appendix C, [15]). Since this sum is 0 on $L(t^2)$, we have that the number of k'-rational points of C' is $q + 1$. □

3. Point Counting Algorithms

In this section, we construct three very simple algorithms for generating curves with a known number of rational points. In the case of genus 2 and 4, we will even get the Zeta functions. Although interesting in their own right, the point counting algorithms will become essential to the discrete logarithm attack we will outline in the next section. For the definition of the L-polynomial and the Zeta function and their relationship, there are many references (see e.g. [25] or [15]), and we will not develop these ideas here (a worthy discussion would take an entire chapter). We will just mention that evaluating the L-polynomial at 1 gives us the order of the Jacobian. As before, we let $\mathbb{F}_q$ be a finite field in odd characteristic and let $\mathbb{F}_{q^2} = \mathbb{F}_q[w]$ be a quadratic extension for $w^2 \in \mathbb{F}_q$. Let C a curve defined by

$$y^2 = (x - w)f \tag{3.1}$$

for any separable $f \in \mathbb{F}_q[x]$ with roots outside $\pm w$.

Our first algorithm can be used to generate curves of any even genus with exactly $q + 1$ points over a finite field.

Algorithm 1.

Input:

(1) an odd prime power q
(2) an integer $g > 0$

Output:
(1) a hyperelliptic curve of genus $2g$ over $\mathbb{F}_q$ with $q+1$ points.
Procedure:
(1) randomly search for a non-square $s \in \mathbb{F}_q$. Set $w^2 = s$.
(2) randomly search for a separable polynomial f of degree $2g+1$.
(3) if f has either w or $-w$ as a root, return to the previous step.
(4) Output equation (2.5) from Corollary 1 using f and w.

Remark 1. The proof of correctness follows from Theorem 1, Corollary 1, and Corollary 3. The complexity of the algorithm amounts to a square root test, sampling separable polynomials of degree $2g+1$, and evaluating them at $\pm w$. It is clear that the cost is $O(g\log(q))$ (see [31]). If g grows linearly and q grows exponentially, we can still generate these curves efficiently.

Our next algorithm can be viewed as a generalization and simplification of Algorithm 1 in [30]. We take in as input any hyperelliptic curve of the special form in Theorem 1. We are then able to construct a new hyperelliptic curve. In the case where we wish to generate genus 2 curves, we simply restrict to elliptic curves of the form given in Theorem 1.

Algorithm 2.
Input:
(1) an odd prime power $q > 3$
(2) the value w
(3) a curve C over $\mathbb{F}_{q^2}$ of the form (3.1)
(4) the L-polynomial L of C.
Output:
(1) a hyperelliptic curve over $\mathbb{F}_q$ with L-polynomial $L(t^2)$.
Procedure:
(1) Output equation (2.5) from Corollary 1 using f and w.

Remark 2. The proof is similar to the previous algorithm; using the results in Section 1, we simply apply Theorem 1 and Corollary 1 to construct C', and then Theorem 2 and Corollary 3 to obtain the relation on the L-polynomial. In the special case where C is an elliptic curve, we obtain a new algorithm for computing genus 2 curves. Beyond reducing the complexity of Algorithm 1 in [30] from $O(\log^2(q))$ operations over $\mathbb{F}_q$ to $O(\log(q))$, our algorithm is able to sample odd and prime order Jacobians, whereas the previous algorithm (in [30]) could only generate even order Jacobians. One important application for these types of algorithms is finding genus 2 curves suitable for public key cryptographic schemes (see [30] for a security analysis for curves of this type). For this purpose, it is important that q be prime. Another application is to generate hyperelliptic curves for pairing based cryptographic schemes. In this case, C would have to be a (geometrically) supersingular hyperelliptic curve. (We will give examples in the Section 5.)

Now that we have seen how to make a genus 2 curve generator using the previous section, we outline how the previous section can generate similar algorithms for higher genus curves. The idea here is to start with an elliptic curve E over some field $\mathbb{F}_{q^{2^k}}$ and construct a series of new curves C_i' over $\mathbb{F}_{q^{2^{k-i}}}$, where each curve C_i' satisfies the condition in Theorem 1. One can expect the equations to become more complicated as i grows, but for small genera one can produce extremely efficient curve generation algorithms using this idea. The genus 4 curve generator below demonstrates this. This algorithm is interesting by itself, and it has the extremely interesting feature that any of the curves it generates are cryptographically weak. Such a curve can be used to attack the elliptic curve used to construct it (see the next section).

To set things up, let $\mathbb{F}_{q^2} \cong \mathbb{F}_q[w_0]$, where w_0 is the square root of a non-square in $\mathbb{F}_q$, and suppose $w_0 - 1$ is a non-square in $\mathbb{F}_q[w_0]$. Let $\mathbb{F}_{q^4} \cong \mathbb{F}_{q^2}[w_1]$ where $w_1^2 = w_0 - 1$. Let E be the elliptic curve

$$E : y^2 = (x - w_1)(x - r_1)(x - r_2), \tag{3.2}$$

where $r_1 = (-2 + w_0 - w_0^2)/(2(w_0 - 1))$ and $r_2 = 1/2(w_1^2 + 1)$.

Algorithm 3.

Input:

(1) w_0

(2) an integer $h = \#E$

Output:

(1) a hyperelliptic curve of genus 4 over $\mathbb{F}_q$ with L-polynomial $q^4t^8 + (h - q^4 - 1)t^4 + 1$ or failure.

Procedure:

(1) If E is not an elliptic curve, output failure.

(2) Otherwise, calculate the polynomial $P \in \mathbb{F}_q[w_0]$ as

$$-\frac{1}{8}(((t-1)^2 + (t+1)^2 w_0^2)(w_0^2(t+1)^2 + 3t^2 - 2w - 1)$$
$$(-2(t-1)^2 - (t-1)(t+3)w_0 - 2(t+1)^2 w_0^2 + (t+1)^2 w_0^3)$$
$$(t(t+2)(w_0^2 - 1) + w_0^2 + 3))$$

(3) Output the hyperelliptic curve $C : y^2 = P$.

Remark 3. This algorithm only fails when r_1, r_2, and $\pm w$ are not distinct. Like the previous algorithm, the formula can be computed in $O(\log(q))$ steps. The only thing that needs to be checked is that the intermediate curve C_1' over $\mathbb{F}_{q^2}$ is of the form required by Theorem 1. This amounts to checking long equations and using elementary algebra, so we omit the proof.

4. The index calculus attack

In Section 2, we mentioned a map δ that could define a homomorphism from the Jacobian of C to the Jacobian of C'. Formally, the description of δ is given in Theorem 2. In this section, we will sketch an algorithm to compute it. For efficiency reasons, we will use Mumford's representation of divisors (for background, see [4]). For simplicity, we will only describe the algorithm in the case that there is one rational point P_∞ at infinity. For a rational point $P = (x_0, y_0)$ on C, recall that $P - P_\infty$ corresponds to the Mumford divisor $[z - x_0, y_0]$ where $k(z)$ is the function field of the projective line covered by C. When we use polynomial interpolation in the following algorithm, we will always mean the polynomial of minimal degree passing through the specified number of points.

Algorithm 4.

Input:

(1) a curve C over $\mathbb{F}_{q^2}$ of the form (3.1)

(2) a divisor $D = [z - x_0, y_0]$ on C

(3) the curve C' from Algorithm 2.

Output:

(1) a divisor D' in Mumford form on C' such that $\delta(D) = D'$.

(1) Define $a_0(z) = (z - x_0)(z - x_0^q)$ and define $a(t) = a_0(x)(t^2 - 1)^2$ where x is as in Corollary 1.

(2) if $y_0 = 0$, then return $[1, 0]$. //this is a ramified point

(3) if $x_0 \in k'$, then //this is a rational point that needs to be counted twice

(a) let $a_1(t)$ and $a_2(t)$ be the irreducible factors of $a(t)$ (they need not be distinct).

(b) if a_1 and a_2 are unique, replace a_i with a_i^2.

(c) Let α_i be the unique root of a_i, set $e_i = (\alpha_i^2 - 1)(\alpha_i + 1)$.

(d) Let T be the tuple $(e_1\mathrm{Tr}(y_0), e_2\mathrm{Tr}(y_0))$

(e) Let G be the polynomial $y^2 = G(x)$ defining C'.

(f) Set $b_i = (T[i]^2 + G)/(2T[i])$.

(g) if $G \bmod a_1 = b_i^2 \bmod a_1$ then return $[a_1, b_i \bmod a_1]$.

(h) if $G \bmod a_2 = b_i^2 \bmod a_2$ then return $[a_2, b_i \bmod a_2]$.

(4) let R be the list of roots of $a(t)$ over a splitting field (let R_i denote the ith coordinate).

(5) let $d_0 = \sqrt{f(x_0^q)(x_0^q - w)}$ over this extension (ignore the sign).

(6) let $e(t) = (t^2 - 1)(t + 1)$.

(7) for $i = 1$ to $\#R$, test if $x(R_i) = x_0$ (where $x(t)$ is the expression in Corollary 1).

(a) if the test passes, append $[(y_0 - d_0)e(R_i), (y_0 + d_0)e(R_i), R_i)]$ to matrix L.
(8) if #Rows(L) < 2 or $x_0 \in k'$, then
(a) let b be the polynomial interpolating the points $(L_{1,3}, L_{1,1})$, $(L^p_{1,3}, L^p_{1,2})$.
(b) replace $a(t)$ with its irreducible factor and return $2[a(t), b]$
(9) else let A interpolate $(L_{1,3}, L_{1,1})$, $(L_{2,3}, L_{2,2})$, $(L^p_{1,3}, L^p_{1,1})$, $(L^p_{2,3}, L^p_{2,2})$.
(10) and let B interpolate $(L_{1,3}, L_{1,2})$, $(L_{2,3}, L_{2,1})$, $(L^p_{1,3}, L^p_{1,2})$, $(L^p_{2,3}, L^p_{2,1})$.
(11) if $G \bmod a(t) = A^2 \bmod a(t)$, return $[a(t), A]$.
(12) else return $[a(t), B]$.

All of the computations preformed in the above procedure are just arithmetic manipulation of small degree polynomials and a few factorizations. The complexity of this algorithm is clearly $O(\log(q))$ (see [31]). The proof of this algorithm is a consequence of the construction given in Theorem 2. To see this, note that at any step in the procedure, at least one root of $a(t)$ lies over the value of x_0 in the covering map $\widetilde{C'} \to C$. This means that $a(t)$ is a polynomial that has roots over the locus of points over x_0 in $\mathbb{P}^1_k$; there are at most 4 such points. What the algorithm is doing is testing all possible cases in this locus using the value y_0. Since δ is a homomorphism, there is only one unique solution (otherwise δ would map a point into a locus). The remaining details are minor. For example, if $x_0 \in k'$, then we can improve the algorithm by splitting up $a(t)$ and using the trace to avoid taking square roots.

The final step is to outline a full attack. First, we sample at random an elliptic curve in the class given by (3.2). We repeat until we find a $\#E$ with a large prime factor, and if so, we take a random discrete logarithm instance D, D' on E. We use Algorithm 3 for the choice of w_0 used to sample E to construct the hyperelliptic genus 4 curve C' over $\mathbb{F}_q$. Now we use the previous algorithm on D and D' twice to get to two divisors D_2, D'_2 on C' (here we may be forced to deal with points at infinity in a different way). A solution to the discrete logarithm problem on D_2 and D'_2 is only off by a factor of 2 when we apply it to the original D and D'.

For the hyperelliptic curve C' given in Algorithm 3, index calculus works by first finding rational points on C' to form a strictly ordered set $S \subset C'(k')$ of some large size. Once such a set has been fixed, one takes a discrete logarithm instance (D, D') on C' and calculates, for each step i, random numbers (α_i, β_i) of size less than the order of the Jacobian of C'. One then computes $D_i = \alpha_i D + \beta_i D'$ and converts this to Mumford notation $[a(t), b(t)]$. If the support of D_i is in S, then one collects (α_i, β_i) and the points in this support. This is the case if $a(t)$ splits and its roots r_i form points $(r_i, b(r_i))$ in S. For each i where this occurs, we form a (usually sparse) matrix M such that each row i has a number representing the multiplicity of these points (with respect to the given strict ordering on S). Once this linear system is large enough, we search through it to find a non-trivial vector in its kernel. Such a vector gives us the relation $R = \sum a_i M_i = 0$, where M_i is the ith row of M. The expression R can be rewritten as $R = \sum a_i(\alpha_i D + \beta_i D') = 0$, and from here we have $\sum a_i \alpha_i D = \sum -a_i \beta_i D'$. If we let $\alpha = \sum a_i \alpha_i$ and $\beta = \sum a_i \beta_i$, we have that $D = -\beta/\alpha D'$.

In our setting, we have a discrete logarithm problem (P, P') on E that is mapped to a discrete logarithm problem (D_2, D'_2) on a genus 4 curve C'. By adapting the index calculus method to mimic the number field sieve with the use of large primes, the amount of work required to solve a discrete logarithm problem on a genus 4 curve is $O(q^{14/9})$ (see [35]). In contrast, the amount of time running Pollard-Rho on the elliptic curve E we have constructed is $O(q^2)$. We will leave it to future work to determine exactly how much of a security parameter increase is required in practice.

We can also offer a few improvements that may help in practice. Since we know $\#C'(k') = q + 1$, we can order $k' \times k'$ in some natural way, and we can let S be the factor basis of points up to some maximal number. This saves us from having to store S. The second improvement is that we can compute $\alpha_i P_i + \beta P'_i$ on E and map it to a divisor on C'. This saves us from having to use the slower algorithms in [4] for Jacobian arithmetic on a hyperelliptic curve. Another option is to instead compute S and map each divisor in S back to E and ignore C' completely. We will leave further discussion to future work.

5. Implementations and Examples

Although little is gained by writing out long equations to give examples, this section looks at the distribution problem. As mentioned in the introduction, very little is understood about the distribution of randomly selected elliptic curves. A fundamental result of M. Deuring in [6] gives us a complete description of what numbers can occur as the number of points on an elliptic curve (see also [36]). However, if we sample random elliptic curves,

then the distribution of their orders among the possible orders is still very mysterious (for conjectures and further discussion, see [13]). Depending on the application, control over the order is often desirable, and it is in these cases that one often assumes that the distribution has regularly occurring values of some desired type. For example, in elliptic curve public key cryptography, one often assumes that via a random search of elliptic curves, one will find prime order groups for a non-negligible portion of all elliptic curves (as the base field grows). In practice, this does seem to occur. The lack of proofs in this area requires us to give examples if we want to make claims about the types of curves we can generate. These examples give evidence that the desired orders can be found via a random search. In another type of distribution problem, it is very easy to single out supersingular elliptic curves from the class of all elliptic curves in characteristic $p \equiv 3 \bmod 4$. We will show how this can be used to construct supersingular hyperelliptic curves where there is a reduction of their discrete logarithm problems to the same problems in a finite field. At the end of this section, we will provide links to source code for all of our algorithms so that they can be tested against various distributions.

5.1. **Weak Elliptic Curves and Points Counts.** Algorithm 2 seems to be able to generate prime order Jacobians of genus 2 curves at a very fast rate. For one such example at 160 bits, we have the hyperelliptic curve

$$y^2 = 67353434412343231568661t^6 + 26676464356409951735460 1t^5 + 24402107149721427771223 0t^4 + 31101789997423268994912 2t^3 + 55713488716871372483478 5t^2 + 53729091200242790650125 6t + 24656406246248887968931 9$$

over the prime field GF$(1207034207473389622886491)$. The Jacobian of this curve has the 160 bit (probable) prime order

$$1456931578010913785349746263294406745202104698519.$$

We had to sample 41 elliptic curves of the form specified in Theorem 1 before it found a prime order elliptic curve; Algorithm 2 then gave us the equation seen above. In other situations, we have found them in as little as 3 random searches, and at other times it has taken around 100.

From a cryptographic point of view, one can conclude that the hyperelliptic curve above has the same level of security as the prime order elliptic curve it was sampled from. To test the speed at which a standard public key exchange could be executed on these curves, we have implemented Cantor's algorithms (see [4]) in a standard (and very basic) hyperelliptic curve package in C++ using the NTL/GMP libraries (see below for links to this source code). We have found that the speed to be competitive on limited hardware, but we will leave a full analysis for future work.

The next example shows two things. On the one hand, it shows us that we can also sample almost of prime order genus 4 curves using Algorithm 3. On the other, it shows us that we can sample elliptic curves of almost prime oder that can be attacked with index calculus. For an example of Algorithm 3 in action, let $w_0^2 = 12499454285990$ over $GF(17592186044423)$ and consider the elliptic curve defined by

$$E : y^2 = x^3 + (17592186044422w_1 + (12188339541165w_0 + 12188339541164))x^2 + ((5403846503258w_0 + 5403846503259)w_1 + (16901862777098w_0 + 4980906183389))x + (690323267325w_0 + 12611279861034)w_1$$

over $GF(175921860444234)$. This curve has the 171 bit prime factor

$$19954369021723020864120283435088194583135361731 09093.$$

One can test that E is ordinary and has no special features that would rule it out as undesirable from a cryptographic point of view. Algorithm 3 gives us the following genus 4 hyperelliptic curve.

$$C : y^2 = 362258632462x^{10} + 8876727378392x^9 + 1189989348357x^8 + 1604531604165x^7 + 8121691981760x^6 + 10448625566159x^5 + 16506034668669x^4 + 15707410711712x^3 + 7302308550432x^2 + 1913995000352x + 6027788515385$$

over $GF(17592186044423)$. Algorithm 4 gives us the transfer from E to Jac(C') (adapting it to work with double points at infinity where needed).

We will leave to future work exactly how much of a security parameter increase is required to protect these vulnerable elliptic curves. In practice, we can sample these weak curves quickly. Moreover, one can also readily produce examples of elliptic curves not in the class specified by Theorem 1 with the same number of points as one that is in this class. Since two elliptic curves with the same number of points are isogenous, we can attack curves not directly in the class of Theorem 1 once we have computed this isogeny. More computational experiments are necessary to determine if there is any hope of writing down exactly which types of elliptic curves must be avoided if they are to be used in public key cryptography.

5.2. **Supersingular Hyperelliptic Curves.** Another application of our algorithms is to use them to construct supersingular curves (see [8] for background information). The existence of these curves is not resolved for all characteristics and for all genera (see [22]). Where they are known to exist, supersingular curves have an application to pairing based cryptography (see [3], [10]). From a discrete logarithm point of view, these curves should be considered very weak when compared to other elliptic curves; it is possible to reduce the discrete logarithm problem on these curves to a discrete logarithm problem in a finite field (see [24]). We will show how to use our algorithms to sample curves where the discrete logarithm is very weak in this sense, even though our curves will be of a higher genus. The fact that our approach generates hyperelliptic curves is interesting for two other reasons: the first is that supersingular hyperelliptic curves are interesting from the theory of moduli (see [22]), and the second is that they have an application to pairing and are more efficient to use than a general curve.

We will prove that every elliptic curve is of the form in Theorem 1 over at most a base extension. This should not be surprising from a geometric point of view: an elliptic curve is given by four distinct points and we can move any three of them by a fractional linear transformation of $\mathbb{P}^1$. In what follows, we will require that the characteristic of our base field is larger than 3.

Lemma 3. Every elliptic curve can be written in the form given in Theorem 1 over at most a finite base extension.

Proof. Over a base extension, E can be written as

$$E : y^2 = t(t-1)(t-\lambda)$$

over some k. For all $w \in k^*$, we can show that E is isomorphic over a finite extension of k to a curve of the form

$$E' : y^2 = (t^2 - \zeta)(t - w)$$

for $\zeta = \lambda^2 w^2/(\lambda-2)^2$ if $\lambda \neq -1$ and $\zeta = w^2/9$ otherwise. To see this, one just computes the j-invariants of E and E' and shows that they are the same. Thus, over a finite extension, E and E' are isomorphic. □

To illustrate how this result can be used to manufacture supersingular hyperelliptic curves, recall that the curve $y^2 = x^3 - x$ is supersingular over $\mathbb{F}_p$ for all $p \equiv 3 \bmod 4$ (this is a simple calculation of the Hasse-invariant, see [33]). Using Lemma 3, this curve can be made into $E : y^2 = (t^2 - w^2/9)(t - w)$ for w the square root of a non-square in $\mathbb{F}_p$. We can now send this to Algorithm 2 to produce an explicit supersingular genus 2 curve C' over $\mathbb{F}_p$. A simple algorithm to go from $\mathrm{Jac}(C')$ to E based on Theorem 2 would be the following.

Algorithm 5.

Input:
(1) $E : y^2 = f$
(2) C'
(3) A Mumford divisor $D = [a(t), b(t)]$, where $\deg(a(t)) \leq 2$.

Output:
(1) a divisor D' on E such that $\delta'(D) = D'$.

Procedure:
(1) Factor $a(x)$ into roots α_1, α_2.
(2) Evaluate $\beta_i = x(\alpha_i)$ for x as in Corollary 1.
(3) Collect the points $P_i = (\beta_i, \pm\sqrt{f(\beta_i)})$, output the P_i that creates D in Algorithm 4.

All divisors on a genus 2 curve can be represented by $[a(t), b(t)]$ for $\deg(a(t)) \leq 2$ (see [4]). If we now apply the reduction in [24] to E, we have reduced the discrete logarithm problem on the divisors of C' to that of a finite field.

Remark 4. The source code for all of these algorithms can be found at http://www.ottojohnston.com/dlp

REFERENCES

[1] E. Arbarello. M. Cornalba. P. Griffiths. J. Harris. *Geometry of Algebraic Curves: Volume 1*, Springer-Verlag, 1985.
[2] E. Artin. *Quadratische Körper im Gebiete der höheren Kongruenzen*, I, II. Math. Z. 19, (1924). Collected Papers, 1-94.
[3] D. Boneh. M. Franklin. *Identity-Based Encryption from the Weil Pairing*, SIAM J. of Computing, 2003.
[4] D. Cantor. *Computing in the Jacobian of a hyperelliptic curve*, Mathematics of Computation 48, 1987.
[5] G. Cornelissen. Erratum to *Two-torsion in the Jacobian of hyperelliptic curves over finite fields*, Arch. Math. 77, 2001.
[6] M. Deuring. *Die Typen der Multiplikatorenringe elliptischer Funktionenkörper*, Abh. Math. Sem. Univ. Hamburg 14, 1941.
[7] C. Diem. *On the discrete logarithm problem in class groups of curves*, Math. Comp., to appear.
[8] T. Ekedahl. *On supersingular curves and abelian varieties*, Math. Scand. 60, 1987.
[9] A. Enge. *Computing Discrete Logarithms in High-Genus Hyperelliptic Jacobians in Provably Subexponential Time*, Math. Comp., 2002.
[10] S. Galbraith. *Supersingular curves in cryptography*, Asiacrypt 2001, Springer LNCS 2248, 2001.
[11] S. Galbraith. M. Harrison. D. Morales. *Efficient Hyperelliptic Arithmetic Using Balanced Representation for Divisors*, ANTS, 2008.
[12] S. Galbraith, F. Hess, N. P. Smart. *Extending the GHS Weil descent attack*, Eurocrypt 2002, Springer LNCS 2332, 2002.
[13] S. Galbraith. J. McKee. *The probability that the number of points on an elliptic curve over a finite field is prime*, Journal of the London Mathematical Society, 62, no. 3, 2000.
[14] P. Gaudry. *An Algorithm for Solving the Discrete Log Problem on Hyperelliptic Curves*, Eurocrypt 2000, Springer LNCS 1807, 2000.
[15] R. Hartshorne. *Algebraic Geometry*, Springer-Verlag., New York, 1977.
[16] F. Hess. *Computing Relations in Divisor Class Groups of Algebraic Curves Over Finite Fields*, preprint, 2004.
[17] M. Jacobson. N. Koblitz. J. Silverman. A. Stein. E. Teske. *Analysis of the Xedni calculus attack*, Design, Codes and Cryptography 20, 2000.
[18] E. Kani. M. Rosen. *Idempotent relations and factors of Jacobians*, Math. Ann. 284, 1989.
[19] K. S. Kedlaya. *Counting points on hyperelliptic curves using Monsky Washnitzer cohomology*, J. Ramanujan Math. Soc, 2003.
[20] N. Koblitz. *Algebraic aspects of cryptography*, Springer-Verlag, New York, 1998.
[21] N. Koblitz. *Hyperelliptic Cryptosystems*, Journal of Cryptology, Volume 1, Number 3, 1989.
[22] K. Li. F. Oort. *Moduli of supersingular abelian varieties*, Lecture notes in mathematics, Vol. 1680, Springer, 1998.
[23] Q. Liu. *Algebraic Geometry and Arithmetic Curves*, Oxford Grad. Texts Math., Oxford University Press Inc., New York, 2002.
[24] A. Menezes. T. Okamoto. S. Vanstone. *Reducing elliptic curve logarithms to logarithms in a finite field*, STOC '91, ACM, 1991.
[25] J. S. Milne. *Jacobian Varieties*, In *Arithmetic Geometry* (Storrs, Conn.), Springer-Verlag, 1984.
[26] J. S. Milne. *On the Arithmetic of Abelian Varieties*, Inventiones math. 17, 1972.
[27] D. Mumford. *Abelian Varities*, 2nd ed, Oxford Univ. Press, Oxford, 1974.
[28] D. Mumford. *The Red Book of Varieties and Schemes*, Lect. Notes Math., Springer, New York, 1988.
[29] J. Pila. *Frobenius maps of Abelian varieties and finding roots of unity in finite fields*, Math. Comput., 1990.
[30] T. Satoh. *Generating Genus Two Hyperelliptic Curves over Large Characteristic Finite Fields*, Eurocrypt 2009, Springer LNCS 5479, 2009.
[31] A. Schönhage. V. Strassen. *Schnelle Multiplikation großer Zahlen*, Computing 7, 1971.
[32] I. Shafarevich. *Basic Algebraic Geometry 1*, 2nd ed., Springer-Verlag, 1994.
[33] J. Silverman. *The Arithmetic of Elliptic Curves*, Springer-Verlag, 1985.
[34] J. Silverman. *The Xedni Calculus And The Elliptic Curve Discrete Logarithm Problem*, Designs, Codes and Cryptography 20, 1999.
[35] N. Thériault. *Index calculus attack for hyperelliptic curves of small genus*, Asiacrypt 2003, Springer LNCS 2894, 2003.
[36] W. C. Waterhouse. *Abelian varieties over finite fields*, Ann. Sci. École-Norm. Sup, 1969.

E-mail address: mail@ottojohnston.com