

## IMPROVING THE PARALLELIZED POLLARD LAMBDA SEARCH ON ANOMALOUS BINARY CURVES

ROBERT GALLANT, ROBERT LAMBERT, AND SCOTT VANSTONE

ABSTRACT. The best algorithm known for finding logarithms on an elliptic curve ( $E$ ) is the (parallelized) Pollard lambda collision search. We show how to apply a Pollard lambda search on a set of equivalence classes derived from  $E$ , which requires fewer iterations than the standard approach. In the case of anomalous binary curves over  $F_{2^m}$ , the new approach speeds up the standard algorithm by a factor of  $\sqrt{2m}$ .

### 1. INTRODUCTION

Let  $E$  be an elliptic curve defined over a finite field  $F_q$ . Let  $P \in E$  be a point of prime order  $n$ , and let  $\langle P \rangle$  be the prime order subgroup of  $E$  generated by  $P$ . In elliptic curve cryptography, the major security consideration is the intractability of the elliptic logarithm problem. If  $Q \in \langle P \rangle$ , then  $Q = kP$  for some integer  $k$ ,  $0 \leq k < n$ , called the *logarithm* of  $Q$  to the base  $P$ , denoted  $\log_P Q$ . The problem of finding  $k$ , given  $P, Q$ , and the parameters of  $E$ , is known as the *discrete logarithm problem* on the elliptic curve. The best general algorithm known for this problem is the Pollard lambda method [5] as parallelized by van Oorschot and Wiener [4]. When  $M$  processors are used, the expected running time of this method is  $(\sqrt{\pi n/2})/M$  steps.

Anomalous binary curves were first suggested for use in cryptography by Koblitz [3]; see also Solinas [6]. In this paper, we show how the parallelized Pollard lambda method can be sped up by a factor of  $\sqrt{2m}$  for anomalous binary curves over  $F_{2^m}$ . For example, this reduces the expected time to solve the Certicom ECC2K-95 challenge [1] from about  $2^{48}$  steps to about  $2^{44}$  steps, and the expected time to solve the the Certicom ECC2K-108 challenge from about  $2^{54}$  steps to about  $2^{50}$  steps. Indeed, the ECC2K-95 was solved in roughly  $2^{44}$  steps by R. Harley of INRIA and his collaborators (including British Telecom).

### 2. EQUIVALENCE CLASSES AND FUNCTIONS

Let  $n - 1 = (2a)b$ . Since  $Z_n^*$  is cyclic, there is an element  $\alpha \in Z_n^*$  of order  $a$ . We can define an equivalence relation on  $\langle P \rangle$  (indeed, on the whole curve) by defining  $S \sim T$  if  $S = \pm\alpha^i T$  for some  $i \in \{0, 1, \dots, a - 1\}$ . This relation partitions  $\langle P \rangle$  into equivalence classes. The point at infinity is in a class by itself, and the other classes contain  $2a$  points each (provided  $-1$  is not a power of  $\alpha$ ). We denote the set of equivalence classes by  $E/\sim$ , and let  $[R]$  denote the equivalence class containing  $R$ .

---

Received by the editor June 9, 1998 and, in revised form, October 15, 1998.  
1991 *Mathematics Subject Classification*. Primary 94A60, 14Q05, 14H52.

The defining equation for an *anomalous binary curve*  $E$  is  $y^2 + xy = x^3 + w_2x^2 + w_6$ , where  $w_2, w_6 \in F_2$ . The *Frobenius map* is the endomorphism  $\phi : E \rightarrow E$  defined by  $\phi : (x, y) \mapsto (x^2, y^2)$  and  $\phi : \mathcal{O} \mapsto \mathcal{O}$ .

We are interested in the  $F_{2^m}$ -rational points of  $E$ . For such points, the Frobenius map is a multiplication map, mapping a point  $R$  to the point  $\lambda R$ , where  $\lambda$  is one of the roots of the characteristic equation of  $\phi$ . As  $\phi^m$  is the identity map, we see that  $\lambda^m \equiv 1 \pmod{n}$ . We assume  $m$  is odd, so that  $-1$  is not a power of  $\lambda$ .<sup>1</sup>

Hence, we are in the situation above with  $\alpha = \lambda$  and  $a = m$ . Thus  $\phi$  induces equivalence classes on  $\langle P \rangle$ , one containing the point at infinity,  $\mathcal{O}$ , and  $(n-1)/2m$  classes of size  $2m$ .

We require a (well-defined) *labelling function*  $\mathcal{L}$ , from the equivalence classes in  $E/\sim$  to some set of representatives  $\mathcal{R}$ . In this paper we require  $\mathcal{L}$  to be one-to-one. Furthermore, we would like this function to be easily computable, given a representative of an equivalence class.

In the case where  $E$  is an anomalous binary curve, a suitable labelling function for our purposes is to take the lexicographically least  $x$ -coordinate of the elements of the equivalence class—where the coordinates are written using a normal basis representation. This function is invariant under the Frobenius map and point negation. It is efficiently computed, since the  $x$ -coordinates of the points in the equivalence class are cyclic rotations of the  $x$ -coordinate of a representative point. Although this labelling function is defined on equivalence classes, it is easily computed, given a representative.

Finally, we require a random map  $\psi$  on  $E/\sim$ . As usual, we settle for a map that “appears” random in practice. Since we will usually only have a representative of a given equivalence class, we will have to ensure that the map is well-defined. Experiments show that the map

$$\psi : E/\sim \rightarrow E/\sim$$

defined by

$$\psi : R \mapsto R + \phi^l(R), \quad \text{where } l = \text{hash}_m(\mathcal{L}(R)),$$

and  $\text{hash}_m$  is a conventional hash function (in the computer science sense) having range  $[0, m-1]$ , acts like a random map on  $E/\sim$ , provided  $m \geq 3$ . Note the key point that this map is defined on points of  $E$ , but is a well-defined map on  $E/\sim$ , as  $\psi([R]) = [\psi(R)]$  for  $R \in E$ . Computing an image under this map is about the same work as a point add, since computing  $\phi^i(R)$  is reasonably easy. An efficient hash function mapping  $\mathcal{L}(R)$  to an integer in the interval  $[0, m-1]$  can be built by XOR-ing words to an appropriate size and taking the result modulo  $m$ . Note that  $\psi(R) = 2R$  is possible, and occurs when  $l = 0$ .

### 3. THE ALGORITHM

We now wish to apply a parallelized Pollard lambda type algorithm using the action of  $\psi$  on  $E/\sim$  to search for collisions. In particular, suppose we want to find  $k = \log_P Q$ , for  $Q \in \langle P \rangle$ . As usual, the idea is to iterate  $\psi$  starting at various classes in  $E/\sim$ . When we detect a collision we can (usually) determine  $k$ .

The algorithm is an analogue of the Pollard lambda algorithm, as parallelized by Wiener and van Oorschot [4]. It is best described in the setting of multiple

<sup>1</sup>If  $m$  is composite, the sets of  $K$ -rational points on  $E$ , where  $K$  is a subfield of  $F_{2^m}$ , form subgroups of the group of points on  $E$ . Thus such curves are normally not used in cryptography.

processors, although it can of course be simulated on a single processor. Suppose we have  $M$  processors. On machine  $i$ , start iterating  $\psi$  on the point  $R_i$ , where  $R_i = u_iP + v_iQ$ , with  $u_i, v_i$  chosen randomly from  $[0, n - 1]$ . One iteration of the algorithm amounts to machine  $i$  updating tuple  $[R_i^s, r_i^s]$  to tuple  $[R_i^{s+1}, r_i^{s+1}]$ , where

$$R_i^{s+1} = \psi(R_i^s) = R_i^s + \phi^l(R_i^s) \quad \text{for } l = \text{hash}_m(\mathcal{L}(R_i^s))$$

and

$$r_i^{s+1} = (1 + \lambda^l)r_i^s \pmod n.$$

Machine  $i$  starts the algorithm with the tuple  $[R_i^0, r_i^0] = [R_i, 1]$ . Note that  $R_i^s = r_i^s R_i = \psi^s(R_i)$ .

As with the parallelized Pollard lambda method, each machine  $i$  occasionally sends distinguished tuples  $[\mathcal{L}(R_i^s), r_i^s, i]$  to a central processor, to be added to a database. The tuples sent and subsequently stored in the database are distinguished in that  $\mathcal{L}(R_i^s)$  satisfies some special property. A standard method is to consider a tuple to be distinguished if  $\mathcal{L}(R_i^s)$ , considered as a bit string, has  $t$  leading zeros. Here  $t$  is chosen as a tradeoff between memory space on the central processor and the expected number of iterations before finding such distinguished tuples.

The central processor also stores the initial values  $R_i, u_i, v_i$ . Note that the distinguished tuples stored in the database take roughly half as much space as algorithms where complete points are stored, or roughly two-thirds of the space in algorithms storing only the  $x$ -coordinates of points (or points in compressed form).

A *collision* occurs when two tuples in the database have the same first component. As with the parallelized Pollard lambda method, it is possible for the function induced by  $\psi$  to have a small cycle containing no distinguished points (and thus going undetected). By judiciously choosing the distinguishing characteristic and by restarting the algorithm with a new point (on a given machine) if no distinguished points are found after a preset number of iterations, we can avoid machines getting stuck in these small cycles. The more straightforward method of adapting Pollard lambda to take advantage of Frobenius and negation mappings requires more effort to deal with small cycles. A discussion of these techniques is delayed until that method is introduced in §6. For the moment, assume that a machine caught in a small cycle will be restarted at a new initial point.

Eventually, a collision in the database will be detected by the central processor. There are two possibilities.

The first (*fruitless*) possibility is that the collision resulted because  $\mathcal{L}(R_i^s) = \mathcal{L}(R_i^t)$  for  $s \neq t$ . In this case the induced equations give no information on the value of  $k$ , and machine  $i$  should be restarted on a new point.

The second (*fruitful*, and the more likely possibility if  $M \geq 3$ ) is that the collision results because  $\mathcal{L}(R_i^s) = \mathcal{L}(R_j^t)$  for  $i \neq j$ , with associated tuples  $[\mathcal{L}(R_i^s), r_i^s, i]$ ,  $[\mathcal{L}(R_j^t), r_j^t, j]$ . Given that a collision has occurred, the probability of a fruitful collision is roughly  $(M - 1)/M$ , and a fruitless collision  $1/M$ . Compute  $S = R_i^s = r_i^s R_i$  and  $T = R_j^t = r_j^t R_j$ . Since  $\mathcal{L}(S) = \mathcal{L}(T)$ , we have  $S = \pm \lambda^l T$  for some  $l$ . A short search ( $m$  applications of the Frobenius map) can be applied to determine the value of  $l$  and thereafter the correct sign, and thus  $c$  such that  $S = cT$ . As  $S = r_i^s u_i P + r_i^s v_i Q$  and  $T = r_j^t u_j P + r_j^t v_j Q$ , it follows that

$$r_i^s u_i P + r_i^s v_i Q = c(r_j^t u_j P + r_j^t v_j Q),$$

whence

$$(r_i^s u_i + r_i^s v_i k)P = (cr_j^t u_j + cr_j^t v_j k)P,$$

and thus

$$(r_i^s u_i + r_i^s v_i k) \equiv (cr_j^t u_j + cr_j^t v_j k) \pmod{n}.$$

With vanishing probability  $r_i^s v_i - cr_j^t v_j \pmod{n}$  is nonzero, so this relation can be solved for  $k$ .

#### 4. RUNNING TIME ESTIMATES

We briefly analyze the running time.

Note that while the algorithm consists of iterating the function  $\psi$  on points of  $E$ ,  $\psi$  is a well-defined function on  $E/\sim$  and furthermore (empirically) behaves like a random map on it. (It is worth mentioning that some of the random behavior of  $\psi$  is probably due to  $\mathcal{L}$ , which  $\psi$  depends on, and also upon  $\text{hash}_m$ .)

Our collision detection actually detects when we have repeated an equivalence class. Therefore, we are essentially applying the standard Pollard lambda search on  $E/\sim$ . As  $E/\sim$  has size  $n/(2m)$ , the standard analysis shows that we can expect a collision in an expected  $\sqrt{(\pi n/2)/(2m)}$  total iterations. Of course, the fact that the fruitless collisions do not give us any information changes the analysis slightly, since in this case we do not end the search. However, if  $M$  is of reasonable size (say  $M > 100$ , as would be expected in any cryptographically significant application of the method), then we can expect to find a fruitful collision, and hence the desired logarithm, in time roughly

$$\frac{1}{M} \sqrt{\frac{\pi}{2} \cdot \frac{n}{2m}}$$

(using  $M$  processors). This decreases the running time of the standard algorithm by a factor of  $\sqrt{2m}$ .

Of course each iteration requires slightly more work than the standard algorithm, since we must evaluate  $\mathcal{L}$  at each step, which is a little more complicated than the analogous iteration function in the standard algorithm. The computation of  $\mathcal{L}$  is not unreasonable, perhaps adding only 20% to the cost of an iteration. Hence, a significant time savings is still realized.

The multiple  $r_i^s$  of  $R_i$  also needs to be updated at each iteration, which seems to require a multiplication by  $1 + \lambda^l$  modulo  $n$ . However,  $r_i^s$  is always of the form  $\prod_{i=0}^{m-1} (1 + \lambda^i)^{e_i}$ . As iterations proceed, the exponents  $e_i$  are incremented, and only when a distinguished point is encountered will the  $m$  exponentiations be performed, and the resulting information sent to the central processor. At this point, the product is remembered, and the exponents  $e_i$  are reset to 0. A precomputed table of some of the powers of  $1 + \lambda^j$  modulo  $n$  will be convenient for the exponentiations. The amortized cost to update the multiples  $r_i^s$  is therefore negligible.

We mention that these ideas can be applied to any elliptic curve, but where we use  $\alpha$  (a generator of a subgroup of order  $m$  in  $Z_n^*$ ) to define the equivalence class instead of  $\lambda$ . However, an efficient labelling function  $\mathcal{L}$  must be found if such a method is to improve on the standard algorithm. The determination of such efficient functions appears difficult, especially if the order  $a$  of  $\alpha$  is large. This is work in progress.

## 5. ALTERNATIVES

Above, we have described a mapping that, although operating on representatives, is well-defined on equivalence classes. An alternative, and more straightforward, method to obtain a mapping that is well-defined on equivalence classes is to build a mapping with domain and range restricted to canonical representatives of the classes.

In typical applications of the Pollard lambda collision search, the “random” function  $\psi$  maps the current point  $R$  to  $R$  plus a linear combination of  $P$  and  $Q$  (where  $P$  is the base of logarithms and  $Q$  is the point for which a logarithm is desired). Usually the function is piecewise defined, using an assortment of linear combinations, one of which is selected depending upon the current point  $R$ . For example, if we have  $N$  linear combinations to choose from, we might regard the  $x$ -coordinate as an integer in  $[0 \dots 2^m]$ , and select the  $i$ th linear combination if this  $x$ -coordinate satisfies  $(i - 1)/N \leq x/2^m < i/N$ . This is suggested in [5], with  $N = 3$ .

If  $R$  is canonical, then the point resulting from the addition of the selected linear combination is not, of course, guaranteed to be a canonical representative of the equivalence class. To force  $\psi$  to be well-defined on equivalence classes, the result of the addition can be normalized to a canonical representative. The normalization process might select the point in the equivalence class of the result having the lexicographically least representation. In the case of anomalous binary curves, with the equivalence relation given earlier, we would select from all Frobenius mappings of both the result and its negative.

To contrast this method from the method described in §3, we examine the form of the iteration used in the random mappings. The method of §3 employs an iterate of the form  $R_{i+1} \leftarrow \mu([R_i]) \cdot R_i$ , where  $\mu([R_i])$  is an integer multiple determined by the equivalence class of  $R_i$ . We call this the multiplicative method. The alternative method of this section more closely resembles the iterate originally employed by Pollard:  $R_{i+1} \leftarrow [R_i + \mu_P(R_i) \cdot P + \mu_Q(R_i) \cdot Q]$ , where  $\mu_P(R_i)$  determines the multiple of  $P$  to add, and similarly  $\mu_Q(R_i)$  the multiple of  $Q$ . We call this the additive method. As will be seen in the next section, this straightforward adaptation of Pollard’s iteration (restricting the domain and range to canonical representatives of the equivalence classes) can suffer from short cycles which yield no information on the logarithm of  $Q$ , and requires correctives to avoid such cycles. We prefer the multiplicative method of §3, since it does not suffer from this defect.

We note that the alternative method described in this section has been independently discovered by Wiener and Zuccherato [8].

## 6. DEALING WITH CYCLES

It is apparent that fruitless cycles can be produced by the additive method described in §5. Assume that the normalization function of §5 finds the point in the equivalence class with the lexicographically least  $(x, y)$ -coordinate pair. Now suppose the current (canonical) point  $R_s$  causes the multiple  $k_s P$  to be added, and that the normalization of  $R_s + k_s P$  is  $-(R_s + k_s P)$ . If the iteration function for  $-(R_s + k_s P)$  adds  $k_s P$  once more, the normalization of the result,  $-R_i$ , would be the original  $R_i$  for the normalization given above (other normalizations can have different behaviour). This small cycle has been produced by cancellation.

Other small cycles can be produced. For example, the characteristic equation  $\lambda^2 - \lambda + 2 = 0$  (if  $\text{trace}(w_2) = 1$ ) is involved in the fruitless cycle:

$$\begin{array}{llll}
 \text{initial point} & & R & \\
 \text{add } P, \text{ then normalize by } -\lambda & \rightarrow & -\lambda(R + P) & = -\lambda R - \lambda P \\
 \text{add } P & \rightarrow & -\lambda R - \lambda P + P & \\
 \text{add } P, \text{ then normalize by } \lambda^{-2} & \rightarrow & \lambda^{-2}(-\lambda R - \lambda P + P + P) & \\
 \text{(recall } -\lambda P + 2P = -\lambda^2 P) & & & = -\lambda^{-1}R - P \\
 \text{add } P, \text{ then normalize by } -\lambda & \rightarrow & -\lambda(-\lambda^{-1}R - P + P) & = R,
 \end{array}$$

which can be produced by the additive algorithm.

Not having the need for normalization, the standard Pollard technique is not likely to produce such cycles. This is because the fruitless cycles in that case occur only if the multiples of  $P$  and  $Q$  are identical (modulo  $n$ ), and this occurs only with vanishing probability for the additive method described in §5 if normalization is not applied. The multiplicative method also preserves this positive feature of the original Pollard method; that is, fruitless cycles occur with small probability, and only when the multiple of the initial point is identical. The additive method with normalization can be biased to avoid fruitless cycles by “widening” the iteration function. To clarify, we call an iteration function having more possibilities at each step than some other function a *wider* iteration function.<sup>2</sup> A wider iteration function will have a larger choice of update operations at each step.

Many iterations are expected to compute a cryptographically significant logarithm. Hence widening the iteration function will only lessen the problem of fruitless cycles—they will still occur and must be detected.

When parallelizing the additive algorithm by using distinguished points, it is possible that a fruitful cycle will go undetected in a cycle containing no distinguished point. For the multiplicative method, such a cycle is always fruitless, but still must be detected and dealt with. In what follows, a *label* refers to a canonical point in the case of the additive algorithm, and an equivalence class label in the case of the multiplicative algorithm.

To detect cycles on a given machine, we propose to intermittently save labels and detect repetitions by comparing new labels against these stored ones. To accomplish this, let each machine maintain an interval value which is the number of iterations a machine will perform before saving the current label (this save is local, and has nothing to do with the label being distinguished).

Several variations of this method are possible, where different numbers of past values are stored, and at different intervals. In practice, it will be sufficient to detect cycles of bounded length, so the save interval can be set to a small value (10 or 20 perhaps). Larger cycles are much less probable, and can be dealt with by the standard method of stopping the iteration after some period of time if a distinguished label has not been encountered.

Once these cycles have been detected, they must be dealt with. In the additive iteration method, such cycles are much more probable. In that case they must first be classified as fruitful or unfruitful. Thereafter the unfruitful cycles can be dealt with in several possible ways. One possibility is to produce a modified iteration that depends only on the labels in the cycle. For example, the cycle could be traversed, the lexicographically least label identified, and a modified iteration taking us out of the cycle could be applied at the point or equivalence class corresponding to this identified label. We call this idea *collapsing* the cycle, since it treats the cycle like

<sup>2</sup>Wider iteration functions have been studied by Teske [7].

a new point in the trajectory. This method ensures that two trajectories entering a cycle will merge. Alternatively, the machine could be restarted.

For the additive algorithm, the most likely fruitless cycle occurs by adding a point, negating as normalization, and adding the same point, as outlined above. We propose that special provision be made for such cycles, involving collapsing the cycles.

In the multiplicative method, such cycles, though unlikely, are always unfruitful. They might be dealt with merely by restarting the machine; this is likely the simplest and best policy. Alternatively, the cycle can be collapsed. The method of collapsing small cycles is much more appropriate for the additive iteration method.

## 7. CONCLUSION

In this paper we outlined an improvement on the standard parallelized Pollard lambda algorithm for finding discrete logs on an anomalous binary curve. The generalization to curves defined over subfields other than  $F_2$  is straightforward. The result is a speedup by a factor of  $\sqrt{2t}$  (for  $t$  odd) for elliptic curves over  $F_{2^{st}}$  that are defined over  $F_{2^s}$ . The algorithm described in §3 appears to be superior to the alternative algorithm described in §5, in that the need to deal with fruitless cycles is much reduced and less storage space is required for the central database.

Elliptic curves over  $F_{2^m}$  which have a defining equation whose coefficients are in a proper subfield of  $F_{2^m}$  thus offer slightly less security than elliptic curves over  $F_{2^m}$  which do not have such a defining equation.

## REFERENCES

- [1] The Certicom ECC Challenge, available from <http://www.certicom.com/chal/>, 1997.
- [2] D. Knuth, *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*, 2nd ed., Addison-Wesley, 1981. MR **83i**:68003
- [3] N. Koblitz, “CM-curves with good cryptographic properties”, *Advances in Cryptology—CRYPTO ’91*, Lecture Notes in Computer Science, **576** (1992), Springer-Verlag, 279–287. MR **94e**:11134
- [4] P. van Oorschot and M. Wiener, “Parallel collision search with cryptanalytic applications”, to appear in *Journal of Cryptology*.
- [5] J. Pollard, “Monte Carlo methods for index computation mod  $p$ ”, *Mathematics of Computation*, **32** (1978), 918–924. MR **58**:10684
- [6] J. Solinas, “An improved algorithm for arithmetic on a family of elliptic curves”, *Advances in Cryptology—CRYPTO ’97*, Lecture Notes in Computer Science, **1294** (1997), Springer-Verlag, 357–371.
- [7] E. Teske, “Speeding up Pollard’s rho method for computing discrete logarithms”, preprint, 1997, available from <http://www.informatik.th-darmstadt.de/TI/Veroeffentlichung/TR/>.
- [8] M. Wiener and R. Zuccherato, “Faster attacks on elliptic curve cryptosystems”, preprint, 1998, available from <http://grouper.ieee.org/groups/1363/contributions/attackEC.ps>.

CERTICOM CORP., 200 MATHESON BLVD. W., SUITE 103, MISSISSAUGA, ONTARIO, CANADA L5R 3L7

*E-mail address:* [rgallant@certicom.com](mailto:rgallant@certicom.com)

*E-mail address:* [rlambert@certicom.com](mailto:rlambert@certicom.com)

*E-mail address:* [svanstone@certicom.com](mailto:svanstone@certicom.com)