SPEEDING UP POLLARD'S RHO METHOD FOR COMPUTING DISCRETE LOGARITHMS

EDLYN TESKE

ABSTRACT. In Pollard's rho method, an iterating function f is used to define a sequence (y_i) by $y_{i+1} = f(y_i)$ for $i=0,1,2,\ldots$, with some starting value y_0 . In this paper, we define and discuss new iterating functions for computing discrete logarithms with the rho method. We compare their performances in experiments with elliptic curve groups. Our experiments show that one of our newly defined functions is expected to reduce the number of steps by a factor of approximately 0.8, in comparison with Pollard's originally used function, and we show that this holds independently of the size of the group order. For group orders large enough such that the run time for precomputation can be neglected, this means a real-time speed-up of more than 1.2.

1. Introduction

Let G be a finite cyclic group, written multiplicatively, and generated by the group element g. Given an element h in G, we wish to find the least non-negative number x such that $g^x = h$. This problem is the discrete logarithm problem (DLP) in G, and x is the discrete logarithm of h to the base g. We write $x = \log_g h$. We write $\langle g \rangle$ to denote the cyclic group generated by g, and $|\langle g \rangle|$, or |G|, to denote the order of g, which is the least positive number x such that $g^x = 1$.

An efficient method to solve the DLP is based on the rho method: Assume we are given a function $f:G\to G$. We select a starting value $y_0\in G$ and then compute the sequence (y_i) formed by the rule $y_{i+1}=f(y_i)$, for $i=0,1,2,\ldots$ Since G is finite, this sequence is ultimately periodic so that there exist two uniquely determined smallest integers $\mu\geq 0$ and $\lambda\geq 1$ such that $y_i=y_{i+\lambda}$ for all $i\geq \mu$. We call μ the preperiod and λ the period of (y_i) and any pair (y_i,y_j) with $y_i=y_j$ and $i\neq j$ a match. If the function f, which we refer to as the iterating function, is a random function in the sense that each of the $|G|^{|G|}$ functions $f:G\to G$ is equally probable, the expected value for $\lambda+\mu$ is close to $\sqrt{\pi|G|/2}=1.253\sqrt{|G|}$. Pollard [12] showed how this theory can be applied to solve the DLP in expected run time $O(\sqrt{|G|})$ multiplications in G.

Pollard's algorithm [12] is generic in the sense that it can be applied to any group for which the following is satisfied.

- Given any two group elements g and h we can compute the product g * h.
- Given any two group elements g and h we can check whether g = h.

Key words and phrases. Pollard's rho method, discrete logarithm, random walks in groups.

1

• Given a small integer r (in Pollard's case: r=3), we can divide the group into r disjoint sets T_1, \ldots, T_r of roughly equal size, and given any group element g we can check to which of these sets it belongs.

The space requirements of algorithms using the rho method are negligible. Therefore, to solve the DLP in groups of large group orders, this method is superior to Shanks' baby step-giant step method [14] that has roughly the same run time but space requirements $O(\sqrt{|G|})$.

Pollard's original algorithm for discrete logarithm computation [12] could be used on a programmable calculator, and Pollard applied it to residue class groups $(\mathbb{Z}/p\mathbb{Z})^*$ (p prime) with group orders up to 10^6 . Nowadays, algorithms based on the rho method run on powerful workstations to solve the DLP in various finite abelian groups of considerably larger group orders. Some work has been done to speed-up the rho method. There are better methods to find matches, e.g. by Brent [2], and van Oorschot and Wiener [18] have developed a method for efficient parallelization of the rho method. We now suggest to choose a more efficient iterating function to obtain further speed-up.

Recently, the author has elaborated a generic algorithm [17] that uses the rho method to compute the structure of a finite abelian group. This algorithm uses a type of iterating functions specially designed to meet the requirements for the group structure computation. Experiments show that the sequences produced by such functions have average values for $\lambda + \mu$ very close to the value $\sqrt{\pi |G|/2}$ expected for the random function case. The question naturally arises whether, for solving the DLP with the rho method, these functions lead to better performances than the function Pollard [12] used.

In this paper, we compare the performances of totally four types of iterating functions, among which we find the functions successfully used for group structure computation as well as Pollard's original function. In comprehensive experiments, we apply them to solve the DLP in groups of points of elliptic curves over prime fields (ECDLP). We chose this particular application since the ECDLP is one of the most fashionable examples for which the rho method is the best algorithm known up to date. This makes these groups particularly interesting for cryptographic applications, and much work has been done in this area since elliptic curve cryptosystems have first been proposed by Miller [10] and Koblitz [7]. However, since we do not exploit any special properties of the groups, the results we obtain are very likely to hold in any other finite abelian group. We also use a result about random walks on the integers $\operatorname{mod} p$ to show that the property of being-close-to-random of the functions used for group structure computation is independent of the size of the group orders, thus answering an open question in Teske [17] in the case of prime group orders.

Due to the method of Pohlig and Hellman [11], both in our experiments and in our theoretical considerations we restrict ourselves to groups of prime order. Then our main result consists of the following empirical estimates.

Let G be a group of prime group order p. Let g be a generator of G and $h \in G$. Let y_0 be a randomly chosen element of G and the sequence (y_i) be formed according

to the rule $y_{i+1} = f(y_i)$, i = 0, 1, 2, ..., with some function $f : G \to G$ that suits to find the discrete logarithm of h to the base g. Let $E(\lambda + \mu)$ denote the expected value of the sum of the preperiod and the period of (y_i) .

If we use the function f_P suggested by Pollard [12] to define the sequence (y_i) , we have

$$E(\lambda + \mu) \ge 1.59\sqrt{p}$$
.

With the same function f_T as used for group structure computation [17], we have

$$E(\lambda + \mu) \leq 1.30\sqrt{p}$$
.

In the following, we show how we solve the DLP using the Pohlig-Hellman and the rho methods. Then, in Section 3, we define our new iterating functions and discuss some important features of the sequences generated by them. In Section 4, we describe our experiments in detail. We give a representative selection of our experimental results and show how they lead to our main result.

2. Pollard's Rho Method for Discrete Logarithm Computation

Let g be a generator of G and $h \in G$. We outline how we use the methods of Pohlig-Hellman and Pollard to compute $\log_q h$.

First we use the Pohlig-Hellman method [11] to reduce the DLP in G to the DLP in groups of prime group order p, with p dividing |G|. Let the prime factorization of |G| be given as $|G| = \prod_p p^{t(p)}$. For each p dividing |G|, we compute $x = \log_g h$ modulo $p^{t(p)}$, as follows. Let t = t(p). We write $x \mod p^t$ in its base p expansion as $x = \sum_{j=0}^{t-1} x_j p^j$. Then the coefficients x_j are computed by solving the equations

$$\left(h * g^{-\sum_{i=0}^{j-1} x_i p^i}\right)^{|G|/p^{j+1}} = \left(g^{|G|/p}\right)^{x_j}, \qquad j = 0, 1, 2, \dots, t-1.$$

Each of these equations represents a DLP in the group of order p generated by $g^{|G|/p}$. Having computed $x \mod p^{t(p)}$ for all prime factors p of |G|, we use the Chinese Remainder Theorem to determine $\log_q h$.

So for the following we assume that G has group order |G| = p with p prime. To describe how to find $\log_g h$ using the rho method, we exemplarily use Pollard's iterating function, to which we refer as *Pollard's original walk*. So we divide G into three pairwise disjoint sets T_1 , T_2 and T_3 of roughly equal size. Let $f_P: G \to G$,

$$f_{P}(y) = \begin{cases} g * y , & y \in T_{1} , \\ y^{2} , & y \in T_{2} , \\ h * y , & y \in T_{3} . \end{cases}$$

We choose a random number α in the range $\{1, \ldots, |G|\}$, compute a starting element $y_0 = g^{\alpha}$, and put $y_{i+1} = f_{\mathbf{P}}(y_i)$, for $i = 0, 1, 2, \ldots$ This induces two integer

sequences (α_i) and (β_i) with the property that $y_i = g^{\alpha_i} * h^{\beta_i}$ for $i = 0, 1, 2, \ldots$. These sequences are given by $\alpha_0 = \alpha$ and $\beta_0 = 0$, and

$$\alpha_{i+1} = \alpha_i + 1$$
, $\alpha_{i+1} \equiv 2\alpha_i \mod |G|$, or $\alpha_{i+1} = \alpha_i$, $\beta_{i+1} = \beta_i$, $\beta_{i+1} \equiv 2\beta_i \mod |G|$, or $\beta_{i+1} = \beta_i + 1$,

according to the three cases above.

While computing the terms (y_i, α_i, β_i) , we try to find a match (y_j, y_i) for some j < i. We use the same method as in Teske [17], which is based on a method of Schnorr and Lenstra [13] but with optimized parameters. This means that we work with a chain of 8 cells, which in each stage of the algorithm store altogether 8 triplets $(y_{\sigma_d}, \alpha_{\sigma_d}, \beta_{\sigma_d})$, $d = 1, \ldots, 8$. In the beginning we put $\sigma_d = 0$ for all d, thus storing (y_0, α_0, β_0) in each cell. After the computation of each new triplet (y_i, α_i, β_i) , we check whether y_i matches one of the stored terms y_{σ_d} . If this is the case for some d, we return the corresponding triplet $(y_{\sigma_d}, \alpha_{\sigma_d}, \beta_{\sigma_d})$ and stop. Otherwise, we check whether $i \geq 3\sigma_1$. If this is the case, we put $\sigma_d = \sigma_{d+1}$ for $d = 1, \ldots, 7$, thus shifting the contents of the 8 cells to the left; we put $\sigma_8 = i$ and store (y_i, α_i, β_i) in the last cell. With μ denoting the preperiod and λ denoting the period of the sequence (y_i) , we have shown [17] that this method finds a match $(y_{\sigma}, y_{\sigma+\lambda})$ with

(2.1)
$$\sigma + \lambda \le 1.25 \cdot \max(\lambda/2, \mu) + \lambda.$$

Having found a match (y_i, y_i) , we have

$$g^{\alpha_j - \alpha_i} = h^{\beta_i - \beta_j} ,$$

from which we can compute $x = \log_q h$ by solving the equation

$$\alpha_j - \alpha_i \equiv (\beta_i - \beta_j)x \bmod p$$
,

provided that $gcd(\beta_i - \beta_j, p) = 1$. If $gcd(\beta_i - \beta_j, p) \neq 1$, we repeat the whole computation with another starting value y_0 ; but this case is very rare for large group orders |G| = p.

Remark 2.1. Under the assumption that f is a random mapping and that |G| is large enough such that a continuous approximation is valid, the expected value of the right-hand side of (2.1) is approximately $1.229\sqrt{\pi|G|/2}$. See [16] for details.

3. The Iterating Functions

We next define some new iterating functions. For $r \in \mathbb{N}$, let T_1, \ldots, T_r be a partition of G into r pairwise disjoint and roughly equally large sets. The set $\{1, 2, \ldots, |G|\}$ is denoted by [1, |G|]. To indicate that an element m is randomly chosen from the set M, according to the uniform distribution, we write $m \in M$.

As in the case of Pollard's original walk, together with the sequence (y_i) we always compute two sequences (α_i) and (β_i) which keep track of the exponents of g and h in the representation $y_i = g^{\alpha_i} * h^{\beta_i}$. Therefore, together with the definition of each iterating function f we indicate how the application of f to y_i effects on α_i and β_i . As before, we have $y_0 = g^{\alpha}$ with $\alpha \in_{\mathbb{R}} [1, |G|]$, and $\alpha_0 = \alpha$, $\beta_0 = 0$.

1. Pollard's walk, modified. As in Pollard's original walk, we use a partition of G into 3 sets. But now we let $m, n \in_R [1, |G|]$ and put $M = g^m$, $N = h^n$. Then we define $f_{Pm}: G \to G$,

$$f_{\rm Pm}(y) = \begin{cases} M * y , & y \in T_1 , \\ y^2 , & y \in T_2 , \\ N * y , & y \in T_3 . \end{cases}$$

For the sequences (α_i) and (β_i) this means

$$\alpha_{i+1} \equiv \alpha_i + m$$
, $\alpha_{i+1} \equiv 2\alpha_i$, or $\alpha_{i+1} = \alpha_i \pmod{|G|}$, $\beta_{i+1} \equiv \beta_i$, $\beta_{i+1} \equiv 2\beta_i$, or $\beta_{i+1} \equiv \beta_i + n \pmod{|G|}$,

according to the three cases above.

2. Linear walk, with 20 multipliers. (This is the walk used in [17].) We use a partition of G into 20 sets T_1, \ldots, T_{20} . Let $m_1, n_1, \ldots, m_{20}, n_{20} \in_R [1, |G|]$ and put

$$M_s = g^{m_s} * h^{n_s}$$
, $s = 1, \dots, 20$.

Define $f_{\mathbf{T}}: G \to G$,

$$f_{\mathrm{T}}(y) = M_s * y$$
, with $s = s(y)$ such that $y \in T_s$.

For the sequences (α_i) and (β_i) this means

$$\alpha_{i+1} \equiv \alpha_i + m_s$$
 and $\beta_{i+1} \equiv \beta_i + n_s \pmod{|G|}$,

where s such that $y_i \in T_s$.

3. Combined walk, with 16 multipliers and 4 squarings. Again, we use a partition of G into 20 sets. Choose 4 pairwise distinct numbers u_1, \ldots, u_4 between 1 and 20, and let $m_1, n_1, \ldots, m_{20}, n_{20} \in_R [\![1, |G|]\!]$. Put

$$M_s = g^{m_s} * h^{n_s}$$
, $s \in \{1, \dots, 20\} \setminus \{u_1, u_2, u_3, u_4\}$.

Define $f_C: G \to G$

$$f_{\mathrm{C}}(y) = \left\{ \begin{array}{cc} M_s * y \ , & \quad \text{if } s \notin \{u_1, u_2, u_3, u_4\} \text{ and with } s \text{ s.th. } y \in T_s \ , \\ y^2 \ , & \quad \text{otherwise } . \end{array} \right.$$

For the sequences (α_i) and (β_i) this means

$$\alpha_{i+1} \equiv \alpha_i + m_s$$
 or $\alpha_{i+1} \equiv 2 * \alpha_i \pmod{|G|}$,
 $\beta_{i+1} \equiv \beta_i + n_s$ or $\beta_{i+1} \equiv 2 * \beta_i \pmod{|G|}$,

according to the cases above.

Note that only when using $f_{\rm T}$ as iterating function we do not need to know the group order. An upper bound of |G| is already suitable for defining the multipliers M_1, \ldots, M_{20} appropriately, and we have shown [17] how one can do without knowing anything about |G|. On the contrary, for $f_{\rm P}$, $f_{\rm Pm}$ and $f_{\rm C}$ the knowledge of |G|, or at least of a multiple of it, is indispensable because of the otherwise exponential growth of (α_i) and (β_i) .

Since $f_{\rm Pm}$ and $f_{\rm C}$ can be viewed as variations of $f_{\rm P}$ and $f_{\rm T}$, we restrict our further discussion of the iterating functions to $f_{\rm P}$ and $f_{\rm T}$.

Let $x = \log_g h$. Then the terms y_i can be written as

$$y_i = g^{e_i}$$
, where $e_i \equiv \alpha_i + \beta_i x \mod p$, $i = 0, 1, 2, \dots$

This means there is a one-to-one correspondence between the walks (y_i) in G and the walks (e_i) on the integers mod p. A truly random walk in G, for instance, corresponds to the walk

$$e_0 \in_R [1, p]$$
, $e_{i+1} \equiv e_i + d \mod p$, with $d \in_R [1, p]$.

In this case, the numbers e_i are uniformly distributed on the integers mod p. Since we want to produce sequences (y_i) with expected preperiods and periods as close as possible to the case of the random walk, it is desirable that the distributions of the e_i generated by our iterating functions get as close as possible to uniformly distributed on the integers mod p, and this should happen after as few steps as possible. In the cases of both f_P and f_T , we have $e_0 \in_R [1, p]$. In the case of f_P , we then have

$$(3.1) \qquad e_{i+1}=e_i+1\ , \qquad e_{i+1}\equiv 2*e_i\ , \quad \text{ or } \quad e_{i+1}\equiv e_i+x \pmod p\ ,$$
 whereas $f_{\rm T}$ produces

(3.2)
$$e_{i+1} \equiv e_i + (m_s + n_s x) \mod p$$
, with $s = s(y_i)$ such that $y_i \in T_s$.

At first glance at these equations, it is not clear how to predict anything about how fast and how close the sequences (e_i) get to uniformly distributed mod p. Maybe the fact that one third of the steps in (3.1) has step-width one and one third has step-width x slows down the process of coming close to uniformly distributed? Maybe this effect varies for different sizes of group orders? One may also have some concerns that in the case of (3.2), the process of getting close to uniformly distributed slows down with increasing group orders, due to the fact that the number of multipliers in the definition of f_T remains constant (= 20).

We are not aware of any theoretical result about the behaviour of the sequence given through (3.1). In the case of the sequence given through (3.2), we can make use of a result of Hildebrand [4], which we present in the following.

Let n be a prime. Let $k \geq 2$ and p_1, \ldots, p_k such that $p_j > 0$ for all j and $\sum_{j=1}^k p_j = 1$. For $a_1, \ldots, a_k \in [1, n]$, let $\widetilde{a} = (a_1, \ldots, a_k)$ and

$$P_{\tilde{a}}(a) = \begin{cases} p_j & \text{if } a = a_j \text{ for some } j, \\ 0 & \text{otherwise }. \end{cases}$$

Then for $m \in \mathbb{N}$, let $P_{\widetilde{a}}^{(m)}$ be the probability distribution of the sum of m independent random variables distributed as $P_{\widetilde{a}}$. We consider the random walk (e_i) on the integers mod n defined by

(3.3)
$$e_0 = 0$$
, $e_{i+1} = e_i + a$, $i = 0, 1, 2, \dots$

where each a is randomly chosen from the set a_1, \ldots, a_k , according to the probability distribution $P_{\tilde{a}}$. Then $P_{\tilde{a}}^{(m)}$ gives the probability distribution of the position of the random walk after m steps. Then, with the distance of a probability distribution P on a finite group G from the uniform distribution U being defined

as

$$||P - U|| := \frac{1}{2} \sum_{v \in G} \left| P(v) - \frac{1}{|G|} \right| = \max_{A \subseteq G} |P(A) - U(A)|,$$

we have the following theorem.

Theorem 3.1 ([4]). Let p_j (j = 1, ..., k), \tilde{a} and $P_{\tilde{a}}$ be as above. Given $\varepsilon > 0$, then for sufficiently large primes n there exists some constant $\gamma > 0$, which may depend on k and on the values for p_j but not on n, such that for $m = \lfloor \gamma n^{2/(k-1)} \rfloor$ we have

$$E(||P_{\widetilde{a}}^{(m)} - U||) < \varepsilon ,$$

where the expectation is taken over a uniform choice of all possible \tilde{a} such that $a_1, \ldots, a_k \in [\![1, n]\!]$ and such that all values of a_1, \ldots, a_k are pairwise distinct.

It is worth noting that Greenhalgh [3] has shown the following lower bound, which nicely complements Theorem 3.1.

Theorem 3.2. Let p_j (j = 1, ..., k), \widetilde{a} and $P_{\widetilde{a}}$ be as above. Then there exists a value $\beta = \beta(p_1, ..., p_k) > 0$ and $n_0 = n_0(p_1, ..., n_k)$ such that for all choices of \widetilde{a} , $m = \lfloor \beta n^{2/(k-1)} \rfloor$ and $n > n_0$,

$$||P_{\widetilde{a}}^{(m)} - U|| \ge \frac{1}{4}.$$

Comparing the number $m = \lfloor \gamma n^{2/(k-1)} \rfloor$ of steps after which the random walk (3.3) is expected to be close to uniformly distributed with the expected number $E(\lambda + \mu) \approx \sqrt{\pi n/2}$ of steps until the first match occurs in the rho method, we see that $\lim_{n\to\infty} m/E(\lambda+\mu)=0$ for $k\geq 6$. Let us now go back to the walk (e_i) defined through (3.2). Let $a_s=m_s+n_sx,\ s=1,\ldots,20$. Let X denote the number of pairwise distinct numbers in the set $\{a_1,\ldots,a_{20}\}$. We have $P(X=20)=\prod_{l=0}^{19}(1-l/n)>0.99998$ for $n\geq 10^7$, so that in the very most cases we work with k=20 pairwise distinct numbers a_s . Apart from this, the situation of (3.2) differs from the situation of Theorem 3.1 only by the fact that $e_{i+1}=e_i+a_s$ with s such that $y_i\in T_s$ rather than s randomly chosen. These differences do not change the following conclusion from Theorems 3.1 and 3.2, which answers an open question in Teske [17].

Corollary 3.1. If for the sequences (y_i) defined by f_T we observe the same average performance for some range of prime group orders, this performance does not considerably change when passing over to much larger group orders.

Remark 3.1. On the other hand, it is very likely that if a certain stable performance for any of the iterating functions $f_{\rm P}$, $f_{\rm Pm}$, $f_{\rm T}$, $f_{\rm C}$ is observed over a sufficiently large range of group orders, it will not considerably improve when passing over to much larger group orders.

4. Experimental Results

Using the computer algebra system LiDIA [9], we implemented the Pohlig-Hellman and the rho methods and conducted experiments to compare the performances of the iterating functions $f_{\rm P}$, $f_{\rm Pm}$, $f_{\rm T}$ and $f_{\rm C}$ to solve the DLP in elliptic curve groups over prime fields of characteristic $\neq 2, 3$. In this section, we describe these experiments and give a representative selection of our experimental results.

Let us first introduce elliptic curve groups over prime fields and the notation we use in the following. We refer to Koblitz [6] for an elementary introduction to elliptic curves, and to Silverman [15] for more details. So let q be a prime $\neq 2, 3$, and let \mathbb{F}_q denote the field $\mathbb{Z}/q\mathbb{Z}$ of integers modulo q. Let $a,b\in\mathbb{F}_q$ such that $4a^3+27b^2\neq 0$. Then the elliptic curve $E_{a,b}$ over \mathbb{F}_q is defined through the equation

$$E_{a,b}: y^2 = x^3 + ax + b$$
.

The set of all solutions $(X,Y) \in \mathbb{F}_q \times \mathbb{F}_q$ of this equation, together with the element \mathcal{O} called the "point at infinity", forms a finite abelian group which we denote by $E_{a,b}(\mathbb{F}_q)$. Usually, this group is written additively. But we remain in the multiplicative setting of the previous sections and therefore write it multiplicatively, which is just a matter of notation.

For $r \in \{3, 20\}$, we define the partition of $E_{a,b}(\mathbb{F}_q)$ into r sets T_1, \ldots, T_r as follows. First we compute a rational approximation A of the golden mean $(\sqrt{5}-1)/2$, with a precision of $2 + \lfloor \log_{10}(qr) \rfloor$ decimal places. Let

$$u^*: E_{a,b}(\mathbb{F}_q) \to [0,1) , \qquad P = P(X,Y) \mapsto \left\{ \begin{array}{cc} (AY) \bmod 1 & & \text{if } P \neq \mathcal{O} \\ 0 & & \text{if } P = \mathcal{O} \end{array} \right.$$

where $c \mod 1$ denotes the (non-negative) fractional part of c, namely $c - \lfloor c \rfloor$. Then let

$$u: E_{a,b}(\mathbb{F}_q) \to \{1,\ldots,r\}, \qquad u(P) = \lfloor u^*(P) \cdot r \rfloor + 1$$

and

$$T_s = \{ P \in E_{a,b}(\mathbb{F}_q) : u(P) = s \} .$$

From the theory of multiplicative hash functions we know [5] that among all numbers between 0 and 1, choosing A as a rational approximation of $(\sqrt{5}-1)/2$ with a sufficiently large precision (that is, in comparison with the input size) leads to the most uniformly distributed hash values, even for non-random inputs. The precision indicated above is large enough such that all decimal places of the golden mean that are significant for the value of u(P) appear in A.

The purpose of our experiments is to produce data on which we can base reliable statements about the expected number of steps until a match is found. These statements are needed for all four iterating functions defined in Sections 2 and 3, and they have to be made in terms of the square root of the orders of the groups in which we use the rho method. For our experiments, this means that we actually do not need to perform all steps of the discrete logarithm computation (as presented in Section 2) in order to get the data we want: Given a discrete logarithm problem, we restrict ourselves to solving it in the subgroup whose order p is the largest prime

factor of |G|, thus producing data relevant for groups of group order p. When using the rho method to compute this discrete logarithm, we count the number of steps we perform until we find a match. Then we determine the ratio R of the number of steps and \sqrt{p} . We do this a couple of times for each iterating function, for a couple of DLPs, in a couple of groups of some group order p between 10^2 and 10^{12} .

Let us describe this explicitly. First we produce a data file containing approximately 2000 6-tuples (q, a, b, n, p, k) with the following properties: $q > 10^2$ and prime, $E_{a,b}$ is an elliptic curve over \mathbb{F}_q , the corresponding elliptic curve group $E_{a,b}(\mathbb{F}_q)$ has group order n, and p is the largest prime factor of n and has k digits, $k \geq 3$. To compute a 6-tuple we select a number l, $2 \leq l \leq 20$, randomly choose a prime q between 10^l and 10^{l+1} , then randomly chose $a,b \in (\mathbb{F}_q)^*$ and check whether $4a^3 + 27b^2 \neq 0 \mod q$. If this is the case, we use our implementation for the group structure computation [17] or, for primes $q > 10^9$, the implementation [8] of an algorithm of Atkin [1], to compute the order n of $E_{a,b}(\mathbb{F}_q)$. Finally we factor n to find p and k. Having built up this file, for $k = 3, 4, \ldots, 13$ we go through the following algorithm:

- 1. Read 6-tuple (q, a, b, n, p, k) from file.
- 2. Use the algorithm for group structure computation to find a group element g such that $g^{n/p}$ has group order p and therefore is a generator of the subgroup $G(p) = \{P^{n/p} : P \in E_{a,b}(\mathbb{F}_q)\}.$
- 3. Randomly choose $h \in E_{a,b}(\mathbb{F}_q)$ and compute $h^{n/p}$.
- 4. Put $g' = g^{n/p}$ and $h' = h^{n/p}$ and G = G(p).
- 5. Use the rho-method as described in Section 2 to compute $\log_g h' = \log_g h$ mod p. For each of the four iterating functions f_P , f_{Pm} , f_T , f_C , do this st times, where st = st(k) and between 100 and 1.
- 6. For each of the four iterating functions, keep track of the average run times and of the average number of steps computed until a match has been found.
- 7. Go back to 1. until m 6-tuples have been used, where m=m(k) and between 100 and 30.

Our results are listed in Tables 1-3.

In Table 1, each row shows the averages taken over the m ratios

```
(average number of steps until match is found (average taken over the st computations for the same DLP) \sqrt{p}
```

where p denotes the order of the group in which the respective computation took place. In the last row we list the averages taken over all ratios in the rows above, where we weighted each ratio by the number of examples having contributed to it. We see a clear difference between the average performances of $f_{\rm P}$ and $f_{\rm Pm}$ on the one hand and of $f_{\rm T}$ and $f_{\rm C}$ on the other hand. The performances are convincingly stable, which gives us a good basis for drawing conclusions in the sense of Corollary 3.1 and Remark 3.1.

Note that the higher oscillation in the first row is mostly due to the fact that as soon as the partition $\{T_1, T_2, T_3\}$ and the group elements g and h are defined, the functional graph associated with the map f_P is completely determined. Therefore,

| Number of | average (number of steps/ \sqrt{p}) with | | | | Number of |
|---------------|---|---------------------------|------------------------------|-----------------|-----------------|
| digits in | Pollard's | Pollard's | linear | combined walk | examples |
| largest prime | original | walk, | walk, with 20 | 16 multipliers, | computed |
| factor p | walk | $\operatorname{modified}$ | $\operatorname{multipliers}$ | 4 squarings | $(m \cdot st)$ |
| 3 | 1.891 | 1.871 | 1.454 | 1.463 | $100 \cdot 100$ |
| 4 | 1.776 | 1.844 | 1.453 | 1.477 | $100 \cdot 100$ |
| 5 | 1.773 | 1.832 | 1.453 | 1.461 | $100 \cdot 100$ |
| 6 | 1.800 | 1.837 | 1.462 | 1.469 | $100 \cdot 100$ |
| 7 | 1.825 | 1.820 | 1.445 | 1.469 | $100 \cdot 100$ |
| 8 | 1.703 | 1.832 | 1.443 | 1.459 | $80 \cdot 40$ |
| 9 | 1.773 | 1.842 | 1.440 | 1.461 | $40 \cdot 30$ |
| 10 | 1.804 | 1.817 | 1.441 | 1.474 | $30 \cdot 30$ |
| 11 | 1.948 | 1.872 | 1.428 | 1.489 | $25 \cdot 20$ |
| 12 | 1.856 | 1.801 | 1.431 | 1.481 | $30 \cdot 5$ |
| 13 | 1.895 | 1.785 | 1.319 | 1.313 | $40 \cdot 1$ |
| ave | 1.807 | 1.841 | 1.452 | 1.467 | |

Table 1. DL-computation in groups of prime order, average number of steps

| | Pollard's | Pollard's | lin. walk, | comb. walk | Number of | |
|--|-----------|---------------------------|------------|--------------|------------------|--|
| | original | walk, | with 20 | 16 multipl., | $_{ m examples}$ | |
| | walk | $\operatorname{modified}$ | multipl. | 4 squarings | $(m \cdot st)$ | |
| $q = 422827, \ a = 334851, \ b = 138169, \ n = 422613, \ p = 46957,$ | | | | | | |
| g = (29541, 46435), h = (105820, 396164), x = 7855 | | | | | | |
| av.no.steps/ \sqrt{p} | 3.193 | 1.892 | 1.527 | 1.481 | $1 \cdot 100$ | |
| av.(10 smallest) | 528 | 87 | 73 | 124 | | |
| av. (10 largest) | 938 | 769 | 766 | 617 | | |
| q = 34158689, a = 5903203, b = 12110056, n = 34152717, p = 81901, | | | | | | |
| g = (1663637, 28574918), h = (27578155, 12646030), x = 48707 | | | | | | |
| av.no.steps/ \sqrt{p} | 0.974 | 1.77 | 1.415 | 1.488 | $1 \cdot 100$ | |
| av.(10 smallest) | 115 | 86 | 95 | 109 | | |
| av. (10 largest) | 497 | 986 | 864 | 898 | | |

Table 2. Two examples with m=1

when using Pollard's original walk the only variation between the st different runs for the same DLP comes through the different starting points, but important properties such as the number of components of the graph or the cycle lengths in the components remain the same. We illustrate this phenomenon in Table 2. For both examples given in this table, we ran our algorithm st=100 times. We see that while the average ratios for $f_{\rm Pm}$, $f_{\rm T}$ and $f_{\rm C}$ differ only slightly from the values in Table 1, the ratios for $f_{\rm P}$ differ considerably. We also show the average values taken over the 10 smallest numbers of steps and the 10 largest numbers of steps. These two examples are extreme but typical cases. In all cases where we took averages over 100 computations, the ratios for $f_{\rm Pm}$ varied between 1.6 and 2.0, for $f_{\rm T}$ and $f_{\rm C}$ between 1.2 and 1.7, but for $f_{\rm P}$ we often found similar deviations as shown in

Table 2. In this respect, the experiments with the modified walk $f_{\rm Pm}$ can be viewed as control experiments for $f_{\rm P}$.

It is interesting to see what the different average ratios mean for the expected run times. Since we do not want do take average run times when different groups are involved, for $k=5,\ldots,13$ we select one elliptic curve group each from the previously computed examples such that the largest prime factor p of the group order has k digits and such that the average ratio (number of steps)/ \sqrt{p} is close to the corresponding value of Table 1. These ratios together with the average run times are listed in Table 3; all run times were taken on a SPARCstation ULTRA170. We see that for prime group orders up to seven digits, the smaller number of steps needed by $f_{\rm T}$ and $f_{\rm C}$ does not pay off in run time, whereas for prime group orders with 9 or more digits we notice a clear speed-up. This is due to the fact that for $f_{\rm T}$ and $f_{\rm C}$ we have to precompute the multipliers. Using the method of fast exponentiation for this, the precomputation requires $O(\log p)$ multiplications so that the run time for it becomes more and more negligible with increasing group orders. In our experiments, it never took more than two seconds to compute the multipliers.

| Number of | | Pollard's | Pollard's | lin. walk, | comb. walk |
|------------------|--------------------------|------------------------------|-------------------------------|-------------------------------|-------------------------------|
| digits in | | original | walk, | with 20 | 16 multipl., |
| largest $p, (p)$ | | walk | modified | $\operatorname{multipl}.$ | 4 squarings |
| 5 | no. of steps/ \sqrt{p} | 1.8 | 1.82 | 1.455 | 1.447 |
| (62219) | run time | $0.10\mathrm{s}$ | $0.11\mathrm{s}$ | $0.19\mathrm{s}$ | $0.17\mathrm{s}$ |
| 6 | no. of steps/ \sqrt{p} | 1.838 | 1.829 | 1.425 | 1.462 |
| (690611) | run time | $0.36\mathrm{s}$ | $0.37\mathrm{s}$ | $0.48\mathrm{s}$ | $0.46\mathrm{s}$ |
| 7 | no. of steps/ \sqrt{p} | 1.831 | 1.813 | 1.452 | 1.486 |
| (2994463) | run time | $0.67\mathrm{s}$ | $0.69\mathrm{s}$ | $0.68\mathrm{s}$ | $0.69\mathrm{s}$ |
| 8 | no. of steps/ \sqrt{p} | 1.711 | 1.883 | 1.447 | 1.465 |
| (29738497) | run time | $2.07\mathrm{s}$ | $2.31 \mathrm{\ s}$ | $1.90\mathrm{s}$ | $1.95\mathrm{s}$ |
| 9 | no. of steps/ \sqrt{p} | 1.727 | 1.846 | 1.44 | 1.472 |
| (102982171) | run time | $4.53\mathrm{s}$ | $4.72 \mathrm{\ s}$ | $3.55\mathrm{s}$ | $3.94\mathrm{s}$ |
| 10 | no. of steps/ \sqrt{p} | 1.818 | 1.857 | 1.467 | 1.469 |
| (1485244759) | run time | $19.84\mathrm{s}$ | $19.89\mathrm{s}$ | $15.22\mathrm{s}$ | $15.23\mathrm{s}$ |
| 11 | no. of steps/ \sqrt{p} | 1.833 | 1.862 | 1.476 | 1.449 |
| (2189335923) | run time | $1\mathrm{m}23.63\mathrm{s}$ | $1\mathrm{m}25.70\mathrm{s}$ | $1\mathrm{m}7.18\mathrm{s}$ | $1\mathrm{m}~7.09\mathrm{s}$ |
| 12 | no. of steps/ \sqrt{p} | 1.809 | 1.832 | 1.434 | 1.458 |
| (416701214639) | run time | $7\mathrm{m}41.84\mathrm{s}$ | $7\mathrm{m}49.9\mathrm{s}$ | $6\mathrm{m}1.22\mathrm{s}$ | $6\mathrm{m}4.18\mathrm{s}$ |
| 13 | no. of steps/ \sqrt{p} | 1.799 | 1.831 | 1.429 | 1.452 |
| (4105475030323) | run time | $27\mathrm{m}17.7\mathrm{s}$ | $27\mathrm{m}48.55\mathrm{s}$ | $22\mathrm{m}13.52\mathrm{s}$ | $22\mathrm{m}29.79\mathrm{s}$ |

Table 3. Selected run times, on SPARCstation ULTRA170

Finally, we want to recover the expected values for $\lambda + \mu$ from our experimental data. For this, we need the expected "delay factor" $\delta = E(l(\lambda, \mu))/(\lambda + \mu)$), where $l(\lambda, \mu)$ denotes the number of steps until a match is found by our algorithm. For the case that the iterating function is a random function, an upper bound for δ is given in Remark 2.1: $\delta \leq 1.229$. A sharp value for δ can be found experimentally. For this, we run our algorithm for groups of small prime group orders, but we store the

| Number of | average(# steps to find match/ $(\lambda + \mu)$) with | | | | |
|---------------|---|---------------------------|------------|--------------|--|
| digits in | Pollard's | Pollard's | lin. walk, | comb. walk | |
| largest prime | original | $\operatorname{walk},$ | with 20 | 16 multipl., | |
| factor p | walk | $\operatorname{modified}$ | multipl. | 4 squarings | |
| 3 | 1.127 | 1.138 | 1.12 | 1.125 | |
| 4 | 1.138 | 1.135 | 1.126 | 1.126 | |
| 5 | 1.137 | 1.134 | 1.131 | 1.128 | |
| 6 | 1.127 | 1.129 | 1.118 | 1.132 | |
| average | 1.132 | 1.134 | 1.124 | 1.128 | |

Table 4. Delay factors

whole sequence. In each run, in addition to $l(\lambda, \mu)$ we determine $\lambda + \mu$ and compute the ratio of both numbers. Having done this 50 times, we take the average over these ratios. The results are shown in Table 4.

Dividing the average values of the last row of Table 1 by the corresponding average delay factors of Table 4, we obtain the following approximations of the expected values for $\lambda + \mu$:

- 1. Pollard's original walk (f_P) : $E(\lambda + \mu) \approx 1.596 \sqrt{|G|}$.
- 2. Pollard's walk, modified (f_{Pm}) : $E(\lambda + \mu) \approx 1.623\sqrt{|G|}$.
- 3. Linear walk, 20 multipliers (f_T) : $E(\lambda + \mu) \approx 1.292 \sqrt{|G|}$.
- 4. Combined walk, 20 mult., 4 squarings (f_C) : $E(\lambda + \mu) \approx 1.3\sqrt{|G|}$.

Comparing these values, we see that we obtain a speed-up of more than 1.2 if we use $f_{\rm T}$ instead of $f_{\rm P}$. Corollary 3.1 and Remark 3.1 ensure that this holds not only for group orders up to 10^{13} but also beyond this bound.

5. Acknowledgments

This paper was written during the author's stay at the Department of Computer Science of the University of Manitoba. The author wishes to thank Hugh Williams for giving her this wonderful opportunity. The author also wishes to thank Eric Bach for pointing out Hildebrand's work, and Martin Hildebrand for his helpful comments on his results.

REFERENCES

- [1] O. Atkin. The number of points on an elliptic curve modulo a prime. Manuscript.
- [2] R. P. Brent. An improved Monte Carlo factorization algorithm. BIT, 20:176-184, 1980.
- [3] A. Greenhalgh. Random walks on groups with subgroup invariance properties. PhD thesis, Department of Mathematics, Stanford University, 1989.
- [4] M. V. Hildebrand. Random walks supported on the random points of Z/nZ. Probability Theory and Related Fields, 100:191-203, 1994.
- [5] D. E. Knuth. The art of computer programming. Volume 3: Sorting and searching. Addison-Wesley, Reading, Massachusetts, 1973.
- [6] N. Koblitz. A Course in Number Theory and Cryptography. Springer-Verlag, New York, 1987.

- [7] N. Koblitz. Elliptic curve cryptosystems. Mathematics of Computation, 48:203-209, 1987.
- [8] F. Lehmann, M. Maurer, V. Müller, and V. Shoup. eco a tool for elliptic curve group order computations, 1997. TI, Technische Universität Darmstadt.
- [9] LiDIA Group, Technische Universität Darmstadt. LiDIA A library for computational number theory. Available from http://www.informatik.tu-darmstadt.de/TI/LiDIA.
- [10] V. Miller. Uses of elliptic curves in cryptography. In Advances in Cryptology CRYPTO '85, volume 218 of Lecture Notes in Computer Science, pages 417-426, 1986.
- [11] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. IEEE-Transactions on Information Theory, 24:106-110, 1978.
- [12] J. M. Pollard. Monte Carlo methods for index computation (mod p). Mathematics of Computation, 32(143):918-924, 1978.
- [13] C.P. Schnorr and H.W. Lenstra, Jr. A Monte Carlo factoring algorithm with linear storage. Mathematics of Computation, 43(167):289-311, 1984.
- [14] D. Shanks. Class number, a theory of factorization and genera. In Proc. Symp. Pure Math. 20, pages 415-440. AMS, Providence, R.I., 1971.
- [15] J. Silverman. The arithmetic of elliptic curves. Springer-Verlag, 1986.
- [16] E. Teske. New algorithms for finite abelian groups. PhD thesis, Technische Universität Darmstadt, 1998.
- [17] E. Teske. A space efficient algorithm for group structure computation. To appear in Mathematics of Computation, 1998.
- [18] P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. To appear in Journal of Cryptology.

TECHNISCHE UNIVERSITÄT DARMSTADT, INSTITUT FÜR THEORETISCHE INFORMATIK, ALEXANDERSTRASSE 10, 64283 DARMSTADT, GERMANY

 $E ext{-}mail\ address: teske@cdc.informatik.tu-darmstadt.de}$