MATHEMATICS OF COMPUTATION
Volume 70, Number 234, Pages 809–825
S 0025-5718(00)01213-8
Article electronically published on February 18, 2000

ON RANDOM WALKS FOR POLLARD'S RHO METHOD

EDLYN TESKE

Abstract. We consider Pollard's rho method for discrete logarithm computation. Usually, in the analysis of its running time the assumption is made that a random walk in the underlying group is simulated. We show that this assumption does not hold for the walk originally suggested by Pollard: its performance is worse than in the random case. We study alternative walks that can be efficiently applied to compute discrete logarithms. We introduce a class of walks that lead to the same performance as expected in the random case. We show that this holds for arbitrarily large prime group orders, thus making Pollard's rho method for prime group orders about 20% faster than before.

1. Introduction

Let G be a finite cyclic group, written multiplicatively, and generated by the group element g. We define the *discrete logarithm problem (DLP)* as follows: given a group element h, find the least non-negative integer x such that $h = g^x$. We write $x = \log_g h$ and call it the *discrete logarithm of h to the base g*. Besides the integer factorization problem, the DLP is the currently most popular computational problem to serve as a base for secure and efficient public-key cryptosystems. Such cryptosystems are, for example, the Diffie-Hellman key exchange protocol, the ElGamal encryption and signature schemes, and the Digital Signature Algorithm (DSA) (cf. [MvOV96]). Originally, they worked with multiplicative groups of finite prime fields. Once elliptic curve cryptosystems were proposed by Koblitz and Miller, analogous practical systems based on the DLP in groups of points of elliptic curves over finite fields were designed (cf. [MvOV96]). While for solving the DLP in $(\mathbb{Z}/p\mathbb{Z})^*$ there exists the index calculus method, which is a subexponential-time algorithm, for the elliptic curve DLP the best algorithms currently known have exponential run time. Among these algorithms we find algorithms based on *Pollard's rho method* [Pol78]. They take expected time $O(\sqrt{n})$ group operations to compute $\log_g h$, where n denotes the order of g. Their space requirements are negligible, and van Oorschot and Wiener [vOW99] showed that they can be efficiently parallelized, which makes the rho method the most powerful method to attack the elliptic curve DLP known to date.

In the rho method, an *iterating function* $F : G \to G$ 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. The sequence $y_0, y_1, y_2, \ldots$ represents a walk in the group G. The basic assumption for the analysis of the expected run time of the rho method is that the walk (y_i) behaves

Received by the editor February 23, 1999 and, in revised form, May 24, 1999.
2000 *Mathematics Subject Classification.* Primary 11Y16; Secondary 65C05, 94A60.
Key words and phrases. Pollard's rho method, discrete logarithm, random walks in groups.

as a *random random walk*. By this we mean that the initial value y_0 is a randomly chosen group element according to the uniform probability distribution, which we denote by $y_0 \in_R G$, and that the iterating function f is a *random mapping* in the sense that it is equally probable among all functions $F : G \to G$.

The problem of efficient simulation of a random random walk in Pollard's rho method is the central topic of this paper. Here, "efficient" means that to evaluate the corresponding iterating function should require essentially no more than one group operation and use only constant or polynomial storage. About the finite abelian group G we only want to assume that its elements are uniquely represented, and that we have an algorithm to perform the group operation.

In Section 2, we give the basic facts and definitions needed throughout the paper, and we describe the set-up for our experiments. Then, in Section 3, we study the function originally suggested by Pollard [Pol78] for discrete logarithm computation in $(\mathbb{Z}/p\mathbb{Z})^*$ (p prime), where we observe that its average performance is worse than expected for a random mapping. We also study the obvious generalization of Pollard's function for arbitrary groups (of prime order), for which we get that its average performance is even worse than in the case of $(\mathbb{Z}/p\mathbb{Z})^*$. In Section 4, we define new iterating functions. These functions extend Pollard's functions in the sense that they perform essentially the same group operations (i.e., multiplication with a fixed element, or squaring), but they have more iterating function rules: up to $r = 100$ rules instead of $r = 3$ as in Pollard's case. We show that for an appropriate choice of the parameters (for example $r \geq 16$, but fixed) they yield an average performance that is hardly distinguishable from the performance of a random mapping. In the case of prime group orders, this can be proved by probability theoretic results on random walks in the additive group $\mathbb{Z}/p\mathbb{Z}$. This also answers Teske's open question [Tes98b], and is the content of Section 5.

Remark 1.1. We would like to mention a related approach by Horwitz and Venkatesan [HV], who considered rapidly-mixing random walks in Cayley graphs over abelian groups. They developed an algorithm, which, if the Cayley graph is generated by $r = O(\log |G|)$ generators, finds a discrete logarithm in (provably) expected running time $O(\log |G| \sqrt{|G|})$.

Acknowledgment

The author wishes to thank John M. Pollard for a very interesting correspondence on the contents of Sections 3 and 4.

2. Preliminaries

Given a group element g, we write $\langle g \rangle$ to denote the cyclic group generated by g. We write $\operatorname{ord} g$ to denote the *order* of g, which is the least positive number x such that $g^x = 1$. The order of the finite abelian group G, which is the number of elements of G, is denoted by $|G|$. If W is a set and w is a randomly chosen element of W (according to the uniform distribution), we write $w \in_R W$. When we work with a finite abelian group G, we assume that the following hold:

1. Given any two group elements g and h, we can compute the product $g * h$.
2. Each group element can be represented as a unique binary string.

Note that the second property implies that we can perform fast equality checks.

2.1. **Pollard's rho method.** If W is any finite set and $F : W \to W$ is a mapping and the sequence $(w_k)_{k\in\mathbb{N}_0}$ in W is formed by the rule

$$w_0 \in W\ , \quad w_{k+1} = F(w_k) \quad k \in \mathbb{N}_0\ , \tag{2.1}$$

this sequence is ultimately periodic. Hence, there exist integers $\lambda \geq 1$ and $\mu \geq 0$ such that $w_0, \ldots, w_{\mu+\lambda-1}$ are pairwise distinct and $w_k = w_{k+\lambda}$, $k \geq \mu$. We call λ the *period* and μ the *preperiod* of the sequence (w_k). Under the assumption that $w_0 \in_R W$ and F is a random mapping, the expected values for μ and λ are close to $\sqrt{\pi|W|/8} = 0.626...\sqrt{|W|}$ ([Har60]). A pair (w_i, w_j) of two terms of the sequence is called a *match* if $w_i = w_j$ and $i < j$. Because of the picture one obtains when drawing the terms of (w_k), starting at the bottom and ending in a cycle, the method of solving computational problems by using sequences as in (2.1) is called the *rho method*. Pollard [Pol75] first applied this result to obtain an efficient and simple algorithm for factoring. Then in [Pol78] he found an algorithm that uses the rho method to compute discrete logarithms in the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ (p prime) in the expected run time of $O(\sqrt{p}\,)$ group operations. This algorithm can easily be generalized to compute discrete logarithms in arbitrary finite abelian groups. More recently, a rho-method based algorithm for group structure computation has been found (see [Tes98b]).

Now we explain how the rho method for computing discrete logarithms works. Let $g, h \in G$ and $\langle g\rangle = G$. Compute the walk (y_k) according to the rule $y_0 = g^{\alpha_0} * h^{\beta_0}$ ($\alpha_0, \beta_0 \in_R \{0, 1, \ldots, |G| - 1\}$) and $y_{k+1} = F(y_k)$ ($k \in \mathbb{N}_0$). Here, the iterating function F must be chosen such that the following holds:

> If we know α and β such that $y = g^\alpha * h^\beta$, we can
>
> efficiently compute α' and β' such that $F(y) = g^{\alpha'} * h^{\beta'}$.

This implies that while computing (y_k), we easily can keep track of the corresponding *sequences of exponents*, (α_k) and (β_k), such that $y_k = g^{\alpha_k} * h^{\beta_k}$ ($k \in \mathbb{N}_0$). Hence, as soon as we have a match (y_i, y_j), we have an equation of the form

$$g^{\alpha_i} * h^{\beta_i} = g^{\alpha_j} * h^{\beta_j}\ .$$

Since $h = g^x$, this gives

$$\alpha_i + \beta_i x \equiv \alpha_j + \beta_j x \mod |G|\ .$$

Now, if $\gcd(\beta_i - \beta_j, |G|) = 1$, we get that $x = (\alpha_j - \alpha_i)(\beta_i - \beta_j)^{-1} \bmod |G|$. Due to the method of Pohlig and Hellman [PH78], in applications the group order $|G|$ is prime, so that it is very unlikely that $\gcd(\beta_i - \beta_j, |G|) > 1$ if $|G|$ is large.

2.2. **Finding a match.** While computing the terms (y_i, α_i, β_i), we try to find a match (y_j, y_i) for some $j < i$. For this, we use the same method as in [Tes98b]. With μ denoting the preperiod and λ denoting the period of the sequence (y_i), this method finds a match $(y_j, y_{j+\lambda})$ with $j \leq 1.25 \cdot \max(\lambda/2, \mu)$. However, this bound is not sharp. Therefore, we define the *expected delay factor* δ as the ratio $E(l(\lambda, \mu))/(\lambda + \mu))$, where $l(\lambda, \mu)$ denotes the number of steps until a match is found. For our match-finding algorithm, we found experimentally that $\delta \approx 1.13$ (cf. [Tes98c]). This implies that if the iterating function behaves like a random mapping, we expect to find a match after approximately

$$1.13 \cdot \sqrt{\pi|G|/2} = 1.416\ldots\sqrt{|G|} \tag{2.2}$$

steps. Let $L_0 = 1.416$. Later, we compare this number with the experimentally determined average values for

$$L := \frac{\text{number of iterations performed until a match is found}}{\sqrt{|G|}} \ . \tag{2.3}$$

Remark 2.1. The method we use to find a match generalizes a method used by Schnorr and Lenstra [SL84] such that optimal average case performance (experimentally) is achieved. A family of match-finding algorithms with optimal worst case performance is discussed in [SSY82]. If storing a large number of terms is not a problem, distinguished point methods as described in [vOW99] can be more efficient to find matches.

2.3. **Partitioning the group.** Let $T_1, \ldots, T_r$ be a partition of G, such that the T_s are pairwise disjoint and of roughly equal size. The iterating functions considered in the following are always given in terms of r different rules, one for each part of the partition. It turns out that for our application it is sufficient to consider the case $3 \leq r \leq 100$. Such a partition can be produced by a hash function $v : G \to \{1, \ldots, r\}$, if we define $T_s = \{a \in G \ : \ v(a) = s\}$.

In our experiments we work with a multiplicative hash function defined as follows: Let A be a rational approximation of the golden mean $(\sqrt{5} - 1)/2$. Let

$$v^* : G \to [0, 1) \ , \qquad v^*(g) = (A \cdot b(g)) \bmod 1 \ ,$$

where $b : G \to \mathbb{R}$ (ideally injective), and $c \bmod 1$ denotes the (non-negative) fractional part of c, namely $c - \lfloor c \rfloor$. The mapping b can be based on the unique encoding of each group element as a binary string. Then let

$$v : G \to \{1, \ldots, r\} \ , \qquad v(g) = \lfloor v^*(g) \cdot r \rfloor + 1 \ . \tag{2.4}$$

From the theory of multiplicative hash functions we know [Knu73] that among all numbers between 0 and 1, choosing A as a rational approximation of $(\sqrt{5}-1)/2$ with a sufficiently large denominator (that is, in comparison with the input size) leads to the most uniformly distributed hash values, even for non-random inputs. This ensures that our results are not distorted by the properties of the hash functions. In practical applications, simpler hash functions that can be evaluated faster are certainly preferable.

2.4. **Reliability considerations.** Our aim is to check the performance of various iterating functions compared to the random mapping case in terms of the average number of steps needed to find a match, which is about 1.13 times the average value of $\lambda + \mu$. A problem in this context is the spread of $\lambda + \mu$ about $E(\lambda + \mu)$. Let $x = (\lambda + \mu)/\sqrt{|G|}$. In the case of a random random walk, the probability density function of x is given by $f(x) = xe^{-x^2/2}$ ([Har60]). This function belongs to a certain Weibull distribution; such distributions are extensively studied in reliability engineering (see, for example, [Kec93]). If we work with a good simulation of a random random walk, it is reasonable to assume that the spread of $(\lambda + \mu)/\sqrt{|G|}$ is similar to the spread of the corresponding Weibull distribution; this agrees with the experimental evidence that when an iterating function yields a mean value of $(\lambda + \mu)/\sqrt{|G|}$ close to the random case, then the variance is close to the variance of the random case, which is $2 - \pi/2$ (see [Tes98a]). Hence, we have to choose the size of the sample space very carefully. We can derive from [Kec93] that, for example, if we work with a sample space of size $N = 30$ and we get an average

value of $(\lambda+\mu)/\sqrt{|G|} = 1.26$, we can only have a 20% confidence that the correct mean value lies between 1.21 and 1.28. If we want to have higher confidence, say 90%, then we only can state that the correct mean value lies between 1.06 and 1.50, which is an interval far too large for our purpose. Experiments show that if we work with $N = 1000$, the average values computed from two such series of computations may differ up to 5%, while choosing $N = 10000$ produces fairly constant average values for $(\lambda+\mu)/\sqrt{|G|}$.

2.5. **Set-up for our experiments.** For our experiments, we use the computer algebra system LiDIA [LiD97]. We work with the multiplicative groups of finite prime fields, with prime-order subgroups of $(\mathbb{Z}/p\mathbb{Z})^*$, and with prime-order subgroups of groups of points on elliptic curves over finite prime fields. For each type of group, we work with (sub)group orders between 10^{n-1} and 10^n, for $3 \le n \le 13$. We do not work with larger groups for practicality reasons, since then the running time becomes too long to work with any meaningful sample space. For fixed n, we repeatedly do the following: First, we find a (sub)group with order between 10^{n-1} and 10^n. Then we find a group element g that generates the (sub)group and and randomly choose another (sub)group element h. This constitutes one instance of DLP. We apply the rho method with various iterating functions, where we always determine the ratio L as defined in (2.3).

3. Findings about Pollard's walk

3.1. **Pollard's original application: $(\mathbb{Z}/p\mathbb{Z})^*$.** Let p be a prime, let h be any integer (mod$p$), and let g be a primitive root modulo p. To compute $\log_g h \pmod p$, Pollard [Pol78] used the iterating function $F : (\mathbb{Z}/p\mathbb{Z})^* \to (\mathbb{Z}/p\mathbb{Z})^*$ given by

$$F(y) = \begin{cases} gy & \text{if } 0 < y \le p/3 \ , \\ y^2 & \text{if } p/3 < y \le 2p/3 \ , \\ hy & \text{if } 2p/3 < y < p \ . \end{cases} \tag{3.1}$$

and defined a sequence (y_k) according to the rule $y_0 = 1$, $y_{k+1} \equiv F(y_k) (\mathrm{mod} p)$. There are sequences (α_k) and (β_k) such that $y_k = g^{\alpha_k} * h^{\beta_k}$ $(k \in \mathbb{N}_0)$, and these sequences follow the rules

$$\begin{aligned} &\alpha_0 = 0 \ , \quad \alpha_{k+1} \equiv \alpha_k + 1 \ , \ 2\alpha_k \ , \text{ or } \alpha_k \bmod (p-1) \ , \qquad k \in \mathbb{N}_0 \ , \\ &\beta_0 = 0 \ , \quad \beta_{k+1} \equiv \beta_k \ , \ 2\beta_k \ , \text{ or } \beta_k + 1 \bmod (p-1) \ , \qquad k \in \mathbb{N}_0 \ , \end{aligned}$$

according to the three cases above.

The experimental results for this walk are given in Table 1. Here, the third column shows the number of examples computed; the first factor indicates how many different instances of the DLP have been used, while the second factor indicates how many times every instance has been solved, each time with a new initial term $y_0 \in_R (\mathbb{Z}/p\mathbb{Z})^*$. In the second column, L is as defined in (2.3); the averages are taken over all examples computed for the respective n. The last row shows the average value for L taken over all 55980 ratios (2.3). We see that the walk (3.1) behaves not exactly like a random random walk but a bit worse.

3.2. **Pollard's original application, generalized.** Given an arbitrary finite abelian group $G = \langle g \rangle$ and a group element h for which we want to compute $\log_g h$,

TABLE 1. DL-computation in $(Z/pZ)^*$ using (3.1)

#digits of group order	average L	number of examples computed
3	1.570	$100 \cdot 100$
4	1.591	$100 \cdot 100$
5	1.553	$100 \cdot 100$
6	1.477	$100 \cdot 100$
7	1.539	$100 \cdot 100$
8	1.605	$80 \cdot 40$
9	1.511	$40 \cdot 30$
10	1.493	$30 \cdot 30$
11	1.690	$25 \cdot 20$
12	1.713	$30 \cdot 5$
13	1.667	$30 \cdot 1$
average	1.553	55980

we can generalize Pollard's rho method as follows: Let T_1, T_2, T_3 be a partition of G, let $Y_0 = 1$, and compute $Y_{k+1} = F(Y_k)$ using the iterating function

$$F(Y) = \begin{cases} Y * g & \text{if } Y \in T_1 \ , \\ Y^2 & \text{if } Y \in T_2 \ , \\ Y * h & \text{if } Y \in T_3 \ . \end{cases} \tag{3.2}$$

In typical applications, the group order $|G|$ is known and one applies the Pohlig-Hellman method [PH78], which reduces the DLP in G to the DLP in groups of prime group order p, with p dividing $|G|$. Then the expected run time of Pollard's rho method is $O(\max\{\sqrt{p} \ : \ p \,|\, |G|\})$ instead of $O(\sqrt{|G|})$, under the assumption that a random random walk in the corresponding subgroups of prime order is simulated.

Experimental results for prime-order subgroups of $(\mathbb{Z}/p\mathbb{Z})^*$ are given in Table 2a). The sample space sizes are the same as in Table 1. Again, the data suggest that the average performance is independent of the size of the group order. But the average values for L are larger than in Table 1. This is somehow surprising since the iterating functions (3.1) and (3.2) look so similar. We discuss this phenomenon below.

When conducting the analogous experiment in elliptic curve subgroups of prime order, with the same sample space sizes as before, we get very similar results. They are shown in Table 2b). We conclude that the canonical generalization of Pollard's rho method for groups of prime order does not produce random random walks: the average performance is by a factor of about 1.25 worse than expected in the random case.

3.3. **Comparison of the walks obtained from** (3.1) **and** (3.2)**.** To understand the difference between applying (3.1) for the DLP in $(\mathbb{Z}/p\mathbb{Z})^*$ and applying (3.2) for the DLP in a prime-order subgroup of some $(\mathbb{Z}/q\mathbb{Z})^*$, we look at an example. Let $G = (\mathbb{Z}/11\mathbb{Z})^*$, $g = 2$, and $h = 5$. Figure 1 shows the actions of the three mappings F_1, F_2, F_3 (according to the three cases in (3.1)). Now let G be the subgroup of order 11 of $(\mathbb{Z}/23\mathbb{Z})^*$ generated by $g = 2$, and let $h = 8$. The actions of the three

TABLE 2. DL-computation in a) subgroups of $(Z/pZ)^*$ of prime order, and b) elliptic curve subgroups of prime order, with iterating function (3.2)

a)

#digits of subgroup order p	average L
3	1.786
4	1.743
5	1.809
6	1.767
7	1.720
8	1.758
9	1.731
10	1.943
11	1.834
12	1.772
13	1.858
average	1.759

b)

#digits of subgroup order p	average L
3	1.891
4	1.776
5	1.773
6	1.800
7	1.825
8	1.703
9	1.773
10	1.804
11	1.948
12	1.856
13	1.924
average	1.807

mappings F_1, F_2, F_3 (according to the three cases in (3.2)) are shown in Figure 2. The remarkable difference between both applications is that the second mapping is a bijection in the latter case, but non-bijective in the first case. That this does have an effect on the performance of the algorithm can be seen if we look at which parts of the graphs above actually take part in the computation. This is indicated by the thick arrows in Figure 3. Hence, when applying (3.1) to $(\mathbb{Z}/p\mathbb{Z})^*$, there are $\lceil p/6 \rceil$ pairs of numbers Y and $-Y (\mathrm{mod} p)$ which are not distinguished by the algorithm. In other words, the algorithm "sees" only about $5/6 \cdot p$ pairwise distinct group elements, which reduces the expected run time by a factor of approximately $\sqrt{5/6} = 0.91$. Indeed, if we compare the average ratios given in Tables 1 and 2, we find that $1.553/1.759 = 0.88$.

Remark 3.1. A slightly different idea is used to speed up the rho method in the case of binary anomalous elliptic curves. There (see [GLV], [WZ98]), an equivalence relation on the group of points is established such that the iterating function "lives" on the equivalence classes rather than on the individual elements.

Remark 3.2. To explain the difference between the performance of (3.1) and (3.2), Pollard [Pol] suggests considering the variance, say V, of the in-degree in the graph corresponding to the iterating function. This method was successfully applied by Brent and Pollard [BP81] for the rho-method for factoring, and works with the conjecture that the expected number of iterations is given as $const/\sqrt{V}$. (For the case that the set of possible values for the in-degree includes zero and at least one integer greater than one, this conjecture was proved by Arney and Bender [AB82].) Here, we have $V = 1$ in the case of (3.1), and $V = 2/3$ in the case of (3.2), so that the in-degree method predicts that (3.1) requires $\sqrt{2/3} = 0.82$ times as many iterations as (3.2).

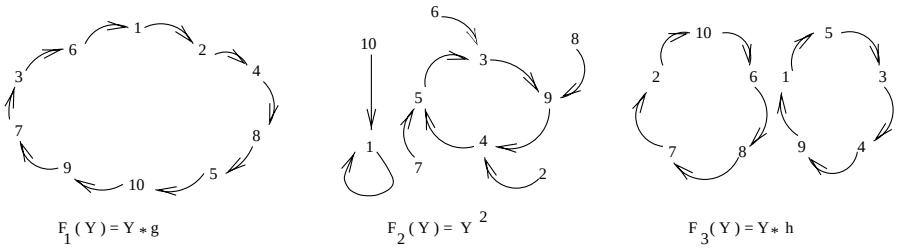

FIGURE 1. Actions of the functions in (3.1) for $G = (\mathbb{Z}/11\mathbb{Z})^*$, $g = 2$, $h = 5$.

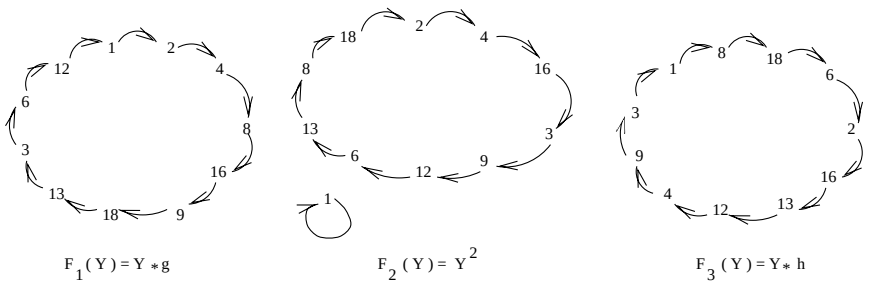

FIGURE 2. Actions of the functions in (3.2) for $G = \langle 2 \rangle \subset (\mathbb{Z}/23\mathbb{Z})^*$, $g = 2$, $h = 8$.

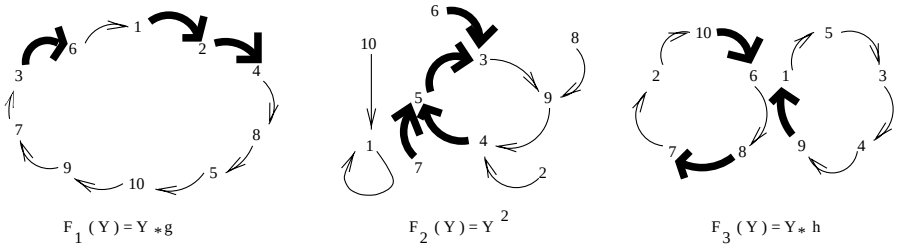

FIGURE 3. Function (3.1) for $G = (\mathbb{Z}/11\mathbb{Z})^*$, $g = 2$, $h = 5$.

4. New walks

In this section we consider alternative walks that, like the walk generated by (3.2), can be used to solve the DLP in any finite abelian group. The aim is to find a walk whose performance comes closer to the random case performance than Pollard's walks. We define and study *r-adding walks* and *$r + q$-mixed walks*. Both walks have the property that evaluating the corresponding iterating function requires only one group multiplication and an evaluation of a fixed hash function $v : G \to \{1, \ldots, r\}$ or $v : G \to \{1, \ldots, r + q\}$, respectively.

4.1. **Adding walks.**

Definition 4.1. Let $r \in \mathbb{N}$ and $M_1, \ldots, M_r$ be randomly chosen elements of G. Let $v : G \to \{1, \ldots, r\}$ be a hash function. A walk (Y_k) in the finite abelian group G such that $Y_{k+1} = F(Y_k)$ for some iterating function $F : G \to G$ is called *r-adding* if F is of the form

$$F(Y) = Y * M_{v(Y)} \ . \tag{4.1}$$

The naming becomes clear when we look at the corresponding sequences of exponents. Assume that given $g, h \in G$, we want to compute $\log_g h$ by using an r-adding walk. We compute Y_0 by choosing $\alpha_0 \in_R \{1, \ldots, |G|\}$ and putting $Y_0 = g^{\alpha_0}$. We compute $M_1, \ldots, M_r$ according to the rule

$$M_s = g^{m_s} * h^{n_s} \ , \qquad s = 1, \ldots, r \ , \tag{4.2}$$

where

$$m_s, n_s \in_R \{1, \ldots, |G|\} \ . \tag{4.3}$$

Then each term Y_k can be represented in the form

$$Y_k = g^{\alpha_k} * h^{\beta_k} \ , \tag{4.4}$$

where $\beta_0 = 0$ and

$$\alpha_{k+1} = \alpha_k + m_{v(Y_k)} \qquad \text{and} \qquad \beta_{k+1} = \beta_k + n_{v(Y_k)} \ . \tag{4.5}$$

Note that these addings are computed modulo $|G|$. However, since the increase of the terms in (4.5) is only linear in the number of iterations, it is not even necessary to perform the reduction modulo $|G|$ while computing the terms of the sequences (α_k) and (β_k). This implies that r-adding walks can also be used if the group order is not known. Note that there is a canonical generalization of these walks for the purposes of element order computation and group structure computation (see [Tes98b]). Schnorr and Lenstra [SL84] used r-adding walks to compute the element order in class groups.

Now, the question is whether r-adding walks achieve the same performance as a random random walk would do, and, if this is the case, how should the parameter r be chosen. Experiments with elliptic curve subgroups of prime group orders up to 13 digits show that $r = 20$ is a good choice: In [Tes98c] we observed that the average values for L were convincingly stable for different sizes of group orders, and very close to the random case value $L_0 = 1.41$. We will see in Section 5 that 20-adding walks are suitable for simulating random random walks for any size of group orders.

We next report on further experiments to study the performance of r-adding walks. Again, we work with elliptic curve groups of prime group orders. For practicality reasons, we use only group orders up to 10 digits. For each range $[10^{n-1}, 10^n]$, we have conducted between 3000 (for the smallest group orders) and 200 (for the largest group orders) DL computations, with a total sum of 11000 computations. For $r \geq 4$ the average value for L (taken over the ratios for a certain range $[10^{n-1}, 10^n]$) appeared to be independent of the size of the group order. The results for $r \geq 4$, in terms of the average values for L taken over all 11000 ratios, are shown in Table 3 and Figure 4. There, we see that the average ratio L as a

TABLE 3. DL-computation in elliptic curve subgroups of prime order, r-adding walk

r	average L
4	1.842
5	1.660
6	1.568
7	1.524
8	1.508
9	1.488
10	1.475
11	1.456
12	1.455
13	1.451
14	1.454
15	1.447
16	1.430

r	average L
17	1.420
18	1.420
19	1.417
20	1.419
30	1.413
40	1.388
50	1.408
60	1.383
70	1.387
80	1.380
90	1.369
100	1.384

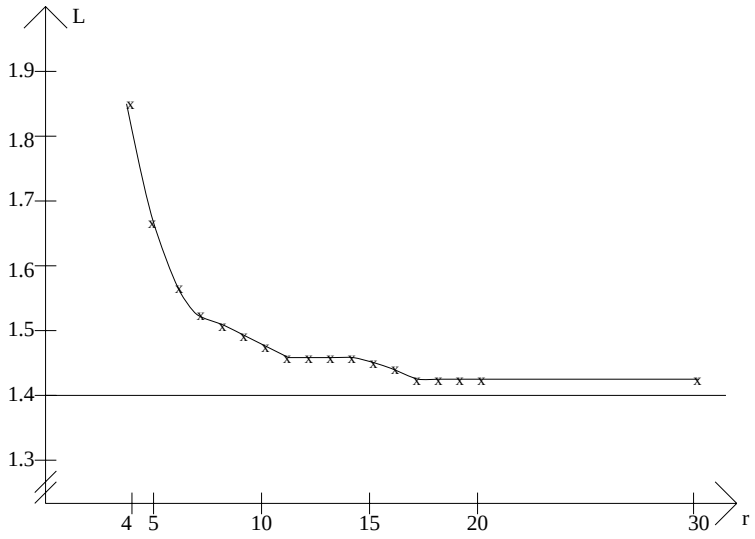

FIGURE 4. DL-computation in elliptic curve subgroups of prime order, r-adding walk

function of r decreases fast until a close-to-random performance is achieved, and then remains rather constant. [1]

On the other hand, for $r = 3$ we do not even obtain a constant average value for L for different ranges of group orders: As the results in Table 4 show, we have

[1] Blackburn and Murphy [BM] suggest under certain heuristic assumptions that the relationship between L and r follows the rule $L = c \cdot \sqrt{\frac{r}{r-1}}$. Their reasoning matches Brent and Pollard [BP81], who conjectured that what matters most for the performance of random walks is the variance of the in-degree in the graph corresponding to the iterating function.

TABLE 4. DL-computation in elliptic curve subgroups of prime order, 3-adding walk

#digits of p	average L
3	2.156
4	2.365
5	2.651
6	2.780
7	3.007
8	3.079
9	3.166
10	3.327
11	3.484

that L steadily increases when the group order gets larger. This behaviour of 3-adding walks is fundamentally different from the behaviour of the generalization of Pollard's original walk (Table 2). This is somewhat surprising since both walks work with a partition of G into 3 parts, and for both walks the mappings F_1, F_2 and F_3 that reflect the three different actions are bijective mappings. Obviously, the different performance for both kinds of walks is due to the squaring step in (3.2), which improves the performance.

4.2. **Mixed walks.** In view of the previous results it is now a natural question to ask: what kind of performance is achieved when we introduce squaring steps to r-adding walks ($r \geq 4$)?

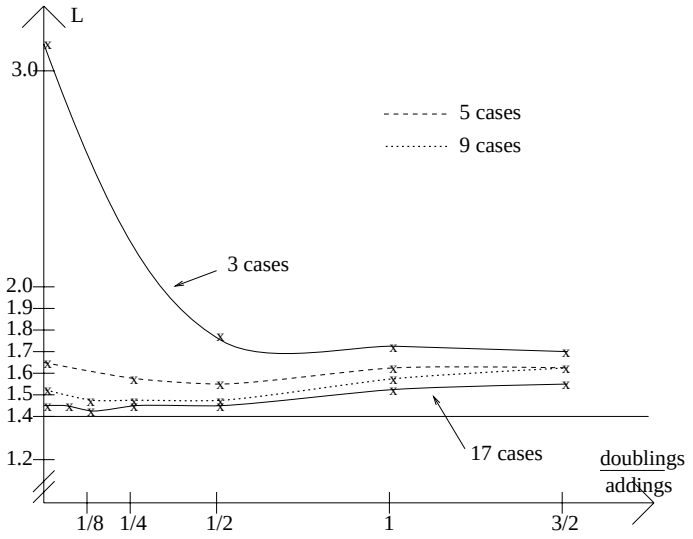

FIGURE 5. Mixed walks for elliptic curve subgroups of prime group order

TABLE 5. Mixed walks for elliptic curve subgroups of prime group order

r (#addings)	q (#doublings)	average L
3	0	3.122
2	1	1.776
2	2	1.770
2	3	1.769
5	0	1.644
4	1	1.589
4	2	1.569
4	4	1.627
4	6	1.618
9	0	1.513
8	1	1.473
8	2	1.496
8	4	1.487
8	8	1.576
8	12	1.623
17	0	1.451
16	1	1.453
16	2	1.427
16	4	1.435
16	8	1.455
16	16	1.502
16	24	1.546

Definition 4.2. Let $r, q \in \mathbb{N}$ and $M_1, \ldots, M_r \in G$. Let $v : G \to \{1, \ldots, r+q\}$ be a hash function. A walk (Y_k) in the finite abelian group G such that $Y_{k+1} = F(Y_k)$ for some iterating function $F : G \to G$ is called $r+q$-*mixed* if F is of the form

$$F(Y) = \begin{cases} Y * M_{v(Y)} & \text{if } v(Y) \in \{1, \ldots, r\} , \\ Y^2 & \text{if } v(Y) \in \{r+1, \ldots, r+q\} . \end{cases} \tag{4.6}$$

For example, the walk generated by the function (3.2) is a 2+1-mixed walk. Note that the squaring step means a doubling step for the corresponding sequences of exponents. Therefore, we also address the parameter q as the number of doublings in a certain walk.

To judge the performance of $r+q$-mixed walks, we have conducted experiments with elliptic curve subgroups of prime group orders in the range $[10^7, 10^8]$. Here, we worked with 200 different instances of the DLP and performed 10 DL computations for each instance. We did this for several combinations of r and q, as shown in Table 5. A graphic interpretation of these results is given in Figure 5. Here, the x-axis bears the ratios q/r; assuming that the sets $T_i = \{a \in G : v(a) = i\}$ $(i = 1, \ldots, r+q)$ are of roughly equal size, q/r is roughly the ratio of $|T_1 \cup \cdots \cup T_r|$ and $|T_{r+1} \cup \cdots \cup T_{r+q}|$. The number of cases in Figure 5 refers to the number of different actions taken by the iterating function (4.6); that is, we consider the doubling-squaring action as a single action regardless of the number q of parts causing this action.

Our experimental results suggest that for a fixed number of cases, the best performance is obtained if the ratio of doublings and addings is between 1/4 and 1/2, while the performance gets worse if this ratio gets much larger than 1.[2] On comparing the data with the corresponding results for r-adding walks, we see that apart from the case $r = 3$, the introduction of a doubling step does not lead to a significantly better performance.

5. Provably good random walks

The aim of this section is to show that the similarity between the performance of an r-adding walk ($r \geq 16$) and the random-case performance that we observed in our experiments has not just been a lucky coincidence. In fact, if we assume that the hash function v is independent, we can prove that this behaviour is typical of such walks.

5.1. **Reduction of the walk (Y_k) to walks on the integers** mod n**.** We consider an r-adding walk in the cyclic group $G = \langle g \rangle$ of group order n. Let x be the smallest integer such that $h = g^x$, and for $s = 1, \ldots, r$ let

$$t_s = m_s + x n_s \mod n \ ,$$

where m_s and n_s are as in (4.2) and (4.3). Then an r-adding walk (Y_k) in G is given by the recurrence

$$Y_{k+1} = Y_k * g^{t_{v(Y_k)}} \ , \qquad k \in \mathbb{N}_0 \ .$$

Further, define the sequence (ϑ_k) by putting

$$\vartheta_k = \alpha_k + x\beta_k \ , \qquad k \in \mathbb{N}_0 \ ,$$

where α_k and β_k are as in (4.4) and (4.5). Then each term Y_k can be represented in the form $Y_k = g^{\vartheta_k}$, and we have

$$\vartheta_{k+1} = \vartheta_k + t_{s_k} \ , \qquad k \in \mathbb{N}_0 \ , \tag{5.1}$$

where $s_k = v(Y_k)$. The bijection

$$\mathbb{Z}/n\mathbb{Z} \ni z \mapsto g^z \in G \ ,$$

establishes a one-to-one correspondence between the r-adding walks (Y_k) in G and the walks (ϑ_k) on the integers mod n. Therefore, instead of studying r-adding walks, we may restrict ourselves to walks of the form (5.1). If each $v(Y_k)$ in (5.1) is randomly chosen from the set $\{1, \ldots, r\}$, the corresponding walk is a random walk on the integers mod n. For such walks we find an elaborate theory in the literature.

5.2. **Random walks on the integers** mod n**.**

Definition 5.1. Let $n \in \mathbb{N}$, and let P be a probability distribution on $\mathbb{Z}/n\mathbb{Z}$. By a *random walk on the integers* mod n *determined by* P we mean a sequence (X_k) given by

$$X_0 = 0 \ , \qquad X_{k+1} = X_k + \xi_k \ , \quad k = 0, 1, 2, \ldots \ ,$$

where each ξ_k is randomly chosen from $\mathbb{Z}/n\mathbb{Z}$, according to the probability distribution P.

[2] It is easy to see that when number of addings is zero, the number of steps until the first match occurs is given by the order of 2 in the group $(\mathbb{Z}/q\mathbb{Z})^*$, where q denotes the order of the actual (sub)group under consideration.

For probability distributions Q and P on $\mathbb{Z}/nZ$, the *convolution product* P^*Q is defined by

$$P^*Q(a) = \sum_{b\in\mathbb{Z}/nZ} P(a-b)Q(b) \ , \qquad a \in \mathbb{Z}/n\mathbb{Z} \ .$$

If (X_k) is a random walk on $\mathbb{Z}/n\mathbb{Z}$ determined by P and $m \in \mathbb{N}$, then P^{*m} is the distribution of the walk after m steps, that is, P^{*m} is the distribution of X_m:

$$P^{*m}(a) = P(X_m = a) \ , \qquad a \in \mathbb{Z}/n\mathbb{Z} \ .$$

The distance we use between probability distributions on G is the variation distance.

Definition 5.2. Let P and Q be probability distributions on a finite group G. The *variation distance* between P and Q is

$$\|P - Q\| := \frac{1}{2}\sum_{a\in G} |P(a) - Q(a)| = \max_{A\subseteq G} |P(A) - Q(A)| \ .$$

It can be verified immediately that the variation distance defines a metric on the set of probability distributions on G.

Now let $r \in \mathbb{N}$ and $p_1, \ldots, p_r$, be such that

$$p_j > 0 \ , \qquad\qquad j = 1, \ldots, r,$$

and

$$\sum_{j=1}^{r} p_j = 1 \ .$$

For $n \in \mathbb{N}$ and any set of integers $\{a_1, \ldots, a_r\} \subseteq \mathbb{Z}/n\mathbb{Z}$ let

$$\widetilde{a} = (a_1, \ldots, a_r) \ .$$

For any such $\widetilde{a}$ we define the probability distribution $P_{\widetilde{a}}$ on the integers mod n by

$$P_{\widetilde{a}} : \mathbb{Z}/n\mathbb{Z} \to [0,1] \ , \qquad P_{\widetilde{a}}(a) = \begin{cases} p_j & \text{if } a = a_j \text{ for some } j \ , \\ 0 & \text{otherwise .} \end{cases}$$

If (X_k) is a random walk on the integers mod n determined by $P_{\widetilde{a}}$, we say that (X_k) is *supported by r points*. The walks (ϑ_k) in (5.1) would fit perfectly into this definition if only the s_k were randomly chosen from the set $\{1, \ldots, r\}$ according to some probability distribution $P_{\widetilde{a}}$ rather than determined by the hash function v. In this case, for any $m \in \mathbb{N}$, the distribution of ϑ_m would be given by the corresponding $P_{\widetilde{a}}^{*m}$. In the special case when $\widetilde{a} = (1, 2, \ldots, n)$ and all these coefficients are equally probable, the probability distribution $P_{\widetilde{a}}$ is the *uniform distribution*, denoted by U. The corresponding random walk (X_k) equals the walk that is obtained by $X_0 = 0$ and the iteration $X_{k+1} = F(X_k)$, $k = 0, 1, 2, \ldots$, with a random function $F : \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$. In other words, the walk corresponding to the uniform distribution is a random random walk.

For the case that n is a prime, Hildebrand [Hil94] has shown that random walks on $\mathbb{Z}/n\mathbb{Z}$ supported by r points get close to uniformly distributed after a constant multiple of $n^{2/(r-1)}$ steps:

Theorem 5.1. *Let $p_1, \dots, p_r$, $\widetilde{a}$ and $P_{\widetilde{a}}$ be as above. Given $\varepsilon > 0$, then for sufficiently large primes n there exists some constant $\gamma > 0$, which may depend on r and on the values for p_j but not on n, such that for $m = \lfloor \gamma n^{2/(r-1)} \rfloor$ we have*

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

where the expectation is taken over a uniform choice of all possible $\widetilde{a}$ such that $a_1, \dots, a_r$ are randomly chosen numbers in $\mathbb{Z}/n\mathbb{Z}$.

If n is composite and the r values satisfy certain divisibility restrictions, it will take only slightly more than $n^{2/(r-1)}$ steps for the position of the random walk to be close to uniformly distributed. To state this result explicitly, we define an *aperiodic set* $\{a_1, \dots, a_r\}$ of $\mathbb{Z}/n\mathbb{Z}$ to be a set such that $\{a_j - a_i \ : \ i, j = 1, \dots, r\}$ generates $\mathbb{Z}/n\mathbb{Z}$. It is easy to see that this is the case if and only if $\gcd(n, a_2 - a_1, \dots, a_r - a_1) = 1$.

Theorem 5.2 ([DH97])**.** *Let $r \geq 2$. Let $M := \{a_1, \dots, a_r\}$ be uniformly chosen from all aperiodic subsets of $\mathbb{Z}/n\mathbb{Z}$ with r pairwise distinct elements. Let $\widetilde{a} = (a_1, \dots, a_r)$, and let $p_1, \dots, p_r$ and $P_{\widetilde{a}}$ as before. Then*

$$E(\|P_{\widetilde{a}}^{(m)} - U\|) \longrightarrow 0 \qquad as \qquad n \to \infty \ ,$$

where

$$m := m(n) \geq \sigma(n) n^{2/(r-1)}$$

and $\sigma(n)$ is any function with $\sigma(n) \to \infty$ as $n \to \infty$. The expected value is over the choice of the set M.

Remark 5.1. Assume that $d \,|\, n$. Then $\gcd(n, a_2 - a_1, \dots, a_r - a_1) = d$ if and only if $a_1, \dots, a_r$ are in the same arithmetic progression modulo d. Hence,

$$P(\gcd(n, a_2 - a_1, \dots, a_r - a_1) = d) = \left(\frac{1}{d}\right)^{r-1} \ .$$

With $r = 20$ and $d \geq 2$ this means that $\{a_1, \dots, a_r\}$ is *not* aperiodic with probability smaller than $2 \cdot 10^{-6}$.

5.3. **Application to r-adding walks.** The basic difference between the model on which the preceding theorems are based and our situation is that we have $s_k = v(Y_{k-1})$, instead of s_k randomly chosen from $\{1, \dots, r\}$ according to some probabilities $p_1, \dots, p_r$. This difference can be removed if we assume that an independent hash function $v : G \to \{1, \dots, r\}$ is used in (4.1).

We now compare the number $m = \lfloor \gamma n^{2/(r-1)} \rfloor$ of steps after which the r-adding walk is expected to be close to uniformly distributed with the expected number $E(\lambda + \mu) = O(\sqrt{n}\,)$ of steps until the first match occurs in the rho method. We see that $\lim_{n\to\infty} m/E(\lambda + \mu) = 0$ for $r \geq 6$. This means that with increasing group order, the part of the walk that does not behave like a random random walk becomes negligible in comparison with the expected length of the walk until a match is found. Hence, we get the following theorem.

Theorem 5.3. *Let G be a finite abelian group of prime group order. Assume we use the rho method to compute discrete logarithms in G. Let $r \geq 6$ and let L be as defined in* (2.3)*. If we work with an r-adding walk in G together with an independent hash function, the average value for L does not get larger with increasing group order.*

In particular, we have

Corollary 5.1. *Assume the hash function* (2.4) *behaves as an independent hash function. Then for any* $r \geq 16$*, the average value for* L *(as in* (2.3)*) obtained with an* r*-adding walk in a group of prime order satisfies*

$$L_{\text{aver.}} \leq 1.45 \ ,$$

regardless of the size of the group order.

6. Conclusion

Pollard's rho method yields a space efficient algorithm to compute discrete logarithms. Its running time analysis has hitherto relied on the assumption that a random random walk in the corresponding cyclic group can be simulated. We have shown that the walk originally used by Pollard for $(\mathbb{Z}/p\mathbb{Z})^*$ does not behave like a random random walk, and the less does its canonical generalization to arbitrary finite abelian groups. On the other hand, we have shown experimentally that there are efficient walks that come very close to the performance of random random walks; examples for such walks are r-adding walks, with $r \geq 16$. We proved this under the assumption that an independent hash function $v : G \to \{1, \ldots, r\}$ is used. In particular, 16-adding walks yield a speed-up by a factor of at least 1.25 if used instead of Pollard's walks. Experimental results suggest that also $r + q$-mixed walks with $r \geq 16$ and $q/r \approx 1/4$ yield a performance that is similar to a random random walk performance.

References

[AB82] J. Arney and E. A. Bender, *Random mappings with constraints on coalescence and number of origins*, Pacific Journal of Mathematics **103** (1982), 269–294. MR **84h:**05110

[BM] S.R. Blackburn and S. Murphy, *The number of partitions in Pollard rho*, Private communication, May 1998.

[BP81] R.P. Brent and J.M. Pollard, *Factorization of the eighth Fermat number*, Mathematics of Computation **36** (1981), 627–630. MR **83h:**10014

[DH97] J. J. Dai and M. V. Hildebrand, *Random random walks on the integers* $\bmod n$, Statistics and Probability Letters **35** (1997), 371–379. MR **99f:**60128

[GLV] R. Gallant, R. Lambert, and S. Vanstone, *Improving the parallelized Pollard lambda search on binary anomalous curves*, to appear in *Mathematics of Computation*.

[Har60] B. Harris, *Probability distributions related to random mappings*, Annals of Math. Statistics **31** (1960), 1045–1062. MR **22:**9993

[Hil94] M. V. Hildebrand, *Random walks supported on the random points of* $\mathbb{Z}/n\mathbb{Z}$, Probability Theory and Related Fields **100** (1994), 191–203. MR **95j:**60015

[HV] J. Horwitz and R. Venkatesan, *Random Cayley graphs and the discrete log*, Preprint, 1998.

[Kec93] D. B. Kececioglu, *Reliability and life testing handbook*, vol. 1, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[Knu73] D. E. Knuth, *The art of computer programming. volume 3: Sorting and searching*, Addison-Wesley, Reading, Massachusetts, 1973. MR **56:**4281

[LiD97] LiDIA Group, Technische Universität Darmstadt, *LiDIA - a library for computational number theory, version 1.3*, 1997, Available from http://www.informatik.tu-darmstadt.de/TI/LiDIA.

[MvOV96] A. Menezes, P. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1996. MR **99g:**94015

[PH78] 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** (1978), 106–110. MR **58:**4617

[Pol] J. M. Pollard, Private communications, March 1998, March 1999.

[Pol75] J. M. Pollard, *A Monte Carlo method for factorization*, BIT **15** (1975), no. 3, 331–335. MR **52:**13611

[Pol78] J. M. Pollard, *Monte Carlo methods for index computation (mod p)*, Mathematics of Computation **32** (1978), no. 143, 918–924. MR **58:**10684

[SL84] C. P. Schnorr and H. W. Lenstra, Jr., *A Monte Carlo factoring algorithm with linear storage*, Mathematics of Computation **43** (1984), no. 167, 289–311. MR **85d:**11106

[SSY82] R. Sedgewick, T. G. Szymanski, and A. C. Yao, *The complexity of finding cycles in periodic functions*, SIAM J. Computing **11** (1982), no. 2, 376–390. MR **83f:**68045

[Tes98a] E. Teske, *New algorithms for finite abelian groups*, Ph.D. thesis, Technische Universität Darmstadt, Germany, 1998, Shaker Verlag, Aachen.

[Tes98b] E. Teske, *A space efficient algorithm for group structure computation*, Mathematics of Computation **67** (1998), 1637–1663. MR **99a:**11146

[Tes98c] E. Teske, *Speeding up Pollard's rho method for computing discrete logarithms*, Algorithmic Number Theory Seminar ANTS-III, Lecture Notes in Computer Science, vol. 1423, Springer-Verlag, 1998, pp. 541–554.

[vOW99] P. C. van Oorschot and M. J. Wiener, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12** (1999), 1–28. MR **99i:**49054

[WZ98] M. Wiener and R. Zuccerato, *Faster attacks on elliptic curve cryptosystems*, Proceedings of SAC, Workshop on Selected Areas in Cryptography, Lecture Notes in Computer Science, 1998.

University of Waterloo, Department of Combinatorics and Optimization, Waterloo, Ontario, Canada N2L 3G1

E-mail address: eteske@cacr.math.uwaterloo.ca