COMPUTING DISCRETE LOGARITHMS WITH THE PARALLELIZED KANGAROO METHOD

EDLYN TESKE

Abstract. The Pollard kangaroo method computes discrete logarithms in arbitrary cyclic groups. It is applied if the discrete logarithm is known to lie in a certain interval, say $[a, b]$, and then has expected running time $O(\sqrt{b-a})$ group operations. In its serial version it uses very little storage. It can be parallelized with linear speed-up, and in its parallelized version its storage requirements can be efficiently monitored. This makes the kangaroo method the most powerful method to solve the discrete logarithm problem in this situation. In this paper, we discuss various experimental and theoretical aspects of the method that are important for its most effective application.

1. Introduction

The security of several important public-key cryptographic systems relies on the difficulty of the discrete logarithm problem (DLP). Important examples are the Digital Signature Algorithm (DSA), which is based on the DLP in multiplicative subgroups of finite fields, or its elliptic curve analogon ECDSA, which is based on the DLP in groups of points of elliptic curves over finite fields (see [MvOV96]).

We define the DLP as follows: Given a generator g of a finite cyclic group G and a group element h, find an integer x such that $g^x = h$. Such a solution x is unique up to multiples of the element order of g; we call it the discrete logarithm of h to the base g, and we write $x = \log_g h$.

In discrete logarithm based signature schemes, the integer x is the secret key used for signature generation, and h is the public key used for signature verification. One way to break such signature schemes is to solve the discrete logarithm problem. In the settings of both DSA and ECDSA, subexponential-time algorithms such as the index-calculus method do not apply, and the best methods known to date to solve the underlying DLPs are the parallelized Pollard rho and kangaroo methods. These methods are generic methods in the sense that they do not require any specific knowledge about the group – we only assume that we can compute the product $u * v$ of any two group elements u and v, and that each group element can be uniquely represented as a binary string.

The rho method [Pol78] is applied when x can be any non-negative integer smaller than $\operatorname{ord} g$, where $\operatorname{ord} g$ denotes the element order of $\operatorname{ord} g$, i.e., the least positive integer n such that $g^n = 1$. Then the rho method can be implemented such that it requires an expected number of $\sqrt{\pi(\operatorname{ord} g)/2} + O(\log(\operatorname{ord} g))$ multiplications to solve the DLP.

In this paper, we are interested in the situation that an interval $[a, b] \subset [0, \operatorname{ord} g]$ is given such that $x \in [a, b]$. For example, this scenario is relevant in signature schemes: computing a signature involves exponentiation of a group element by the secret key x, which can be done the faster, the smaller x is. Thus, to speed up the signature generation, one might want to choose the secret key x from an interval $[0, b]$ that is much smaller than the whole range $[0, \operatorname{ord} g]$. However,

Date: February 26, 2001.
1991 *Mathematics Subject Classification.* 11Y16, 94A60.

this reduces the security of the scheme, because with Pollard's kangaroo method [Pol78] one can compute discrete logarithms $x \in [a,b]$ in expected running time $2\sqrt{b-a}+O(\log(\operatorname{ord} g))$ rather than $O(\sqrt{\operatorname{ord} g})$ operations. Just as the rho method, the kangaroo method needs to store only a small, constant number of group elements, and it can be parallelized with linear speed-up [vOW99]. In the parallelized case we have increased off-line space requirements but they can be efficiently monitored.

We remark that also with the baby-step giant-step method one can fully exploit the knowledge that $x \in [a,b]$ to compute the discrete logarithm in at most $2\sqrt{b-a}$ group operations ($3\sqrt{b-a}/2$ on average). However, this method is not practical for large intervals since it has to store $\lceil\sqrt{b-a}\rceil$ group elements, and it cannot be efficiently parallelized.

In the following, we discuss the kangaroo method in more detail. After a description of its serial version (Section 2), we address the intrinsic differences between the kangaroo and the rho methods (Section 3). We then describe the distinguished point method (Section 4) and discuss an appropriate choice of its parameters. This method is crucial for the parallelization of the kangaroo method, both in the variants by van Oorschot and Wiener [vOW99] and in the variant by Pollard [Pol00]. We discuss both variants in Section 5, where we also address which sets of jumps and which spacings to choose. In Section 6 we focus on the analysis of the running time. Here we examine the distribution of the running time, the travel distances of the kangaroos, the probability that a kangaroo ends up in a cycle, and the case that the number of processors is not known *a-priori.* We also discuss the underlying heuristic assumptions for the running time analysis. In Section 7, where we deal with the issue of useless collisions, a tricky phenomenon that occurs in the van Oorschot-Wiener parallelization. We give a selection of experimental results in Section 8 that show the performance of the method in practice and illustrate some of the material in the preceding sections. Finally (Section 9) we discuss further applications of the (parallelized) kangaroo method.

Notation: If a and b are non-negative integers, $a < b$, and x is an integer chosen uniformly at random from the interval $[a,b]$, we write $x \in_R [a,b]$. For a group element g, we write $\langle g\rangle$ to denote the set $\{1, g, g^2, g^3, \ldots\}$, which is finite if and only if g is of finite order.

Acknowledgments. The author wishes to thank John Pollard for many interesting and helpful comments, which particularly improved Section 6. The author is grateful to the Mathematical Sciences Research Institute (MSRI) in Berkeley, California for its hospitality for a month in the Fall 2000 during its program on Algorithmic Number Theory.

2. Catching Kangaroos

Let g and h be group elements with $g^x = h$, where $x \in [a, b]$ but unknown.

We present the kangaroo method not in its original version [Pol78] but in the version by van Oorschot and Wiener [vOW99], which is faster than the original version if one allows slightly more storage.

We have two actors in the kangaroo method, a tame and a wild kangaroo. Their positions are represented by group elements, and they travel in the cyclic group $G = \langle g \rangle$. The tame kangaroo is set off at the group element $t_0 = g^{(a+b)/2}$, and the wild kangaroo is set off at the group element $w_0 = h$. Both starting points uniquely correspond to locations in the interval $[a, b]$: the tame kangaroo starts at the middle of the interval, and the wild kangaroo starts at $x = \log_g h$. Since we do not know x, we do not know the exact location of the wild kangaroo, and that is why it is called wild. The purpose is to provoke a collision between the tame and the wild kangaroo, from which we can deduce the wild kangaroo's starting point.

For this, we define a set of jump distances

$$S = \{s_1, \dots, s_r\}$$

with $s_i > 0, s_i = O(\sqrt{b-a})$ and $r = O(1)$, and a set of jumps

$$J = \{g^{s_1}, \dots, g^{s_r}\} \ .$$

Here, the exact choice of the s_i and r will be discussed further below and in Section 5.1. The kangaroos' travels consists of jumps, where each jump is a multiplication of the current position by some $g^{s_i} \in J$.

We choose a hash function $G \to \{1, \dots, r\}$ that divides G into r pairwise disjoint sets $M_1, \dots, M_r$, which gives a rule for the kangaroos' jumps. For example, if $t_0 \in M_3$, then the first jump of the tame kangaroo gives $t_1 = t_0 * g^{s_3}$. In general, for $k = 0, 1, 2, \dots$ we compute

$$\begin{aligned} t_{k+1} &= t_k * g^{s_i} \qquad \text{where } t_k \in M_i \ , \\ w_{k+1} &= w_k * g^{s_j} \qquad \text{where } w_k \in M_j \ , \end{aligned}$$

and thus obtain two sequences (t_k) and (w_k) in $\langle g \rangle$. At the same time, we keep track of the distances the kangaroos travel by setting $\delta_{0,\text{tame}} = \delta_{0,\text{wild}} = 0$ and

$$\begin{aligned} \delta_{k+1,\text{tame}} &= \delta_{k,\text{tame}} + s_i \qquad \text{where } t_k \in M_i \ , \\ \delta_{k+1,\text{wild}} &= \delta_{k,\text{wild}} + s_j \qquad \text{where } w_k \in M_j \ , \end{aligned}$$

which defines two strictly increasing sequences of integers.

With a high probability there will be indices k and k' such that $t_k = w_{k'}$ (the case that this does not happen is addressed in Section 6.3). Notice that usually $k \neq k'$, so that we only have a collision between the two kangaroos' paths rather than a collision between the kangaroos themselves (so no one is getting hurt). Such a match $t_k = w_{k'}$, which can be detected using the distinguished point method (see Section 4), corresponds to the equation

$$g^{\frac{a+b}{2}} * g^{\delta_{k,\text{tame}}} = h * g^{\delta_{k',\text{wild}}} \ .$$

Since $h = g^x$, this translates into

$$x \equiv \frac{a+b}{2} + \delta_{k,\text{tame}} - \delta_{k',\text{wild}} \pmod{\operatorname{ord} g} \ . \tag{2.1}$$

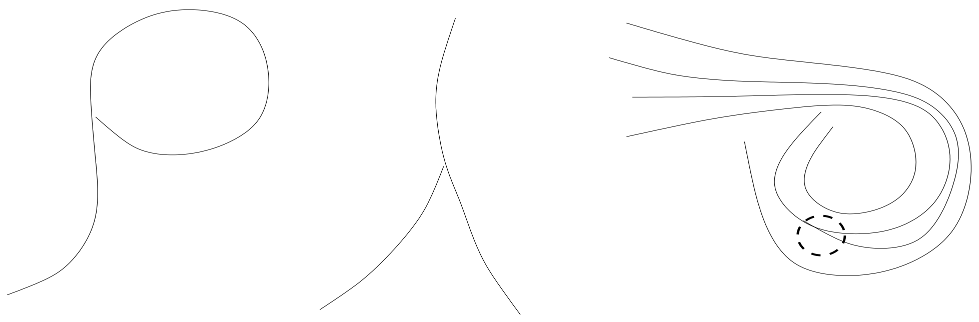

FIGURE 1.

We even have that (2.1) is most likely to hold over the integers (see Section 6.2):

$$x = \frac{a+b}{2} + \delta_{k,\text{tame}} - \delta_{k',\text{wild}} \ ,$$

so that we do not need to know $\operatorname{ord} g$ to find $x = \log_g h$.

Van Oorschot and Wiener [vOW99] have analyzed this method and have found that the expected running time is minimized if the jump distances s_i are chosen such that their mean value is approximately $\sqrt{b-a}\,/2$. Then one obtains that the expected number of jumps of both kangaroos until their paths collide is altogether $2\sqrt{b-a}$.

3. Kangaroo Method versus Rho Method – Or: What is the Lambda Method?

The kangaroo method is also known as the lambda method, but since the parallelization of the rho method has become popular, the rho method is sometimes also referred to as the lambda method. So both methods occasionally get mixed up. However, there is an intrinsic difference between them, which we address in this section.

Where does the confusion come from?

In Pollard's rho method for discrete logarithm computation a sequence (y_k) in G is defined by choosing an initial term $y_0 \in G$ and then following the rule $y_{k+1} = F(y_k)$, $k \in \mathbb{N}$, where $F : G \to G$ is a pseudo-random mapping. Such a sequence (y_k) is ultimately periodic. If its terms are drawn on a piece of paper starting at the bottom and ending in a cycle, the figure one obtains has the shape of the Greek letter rho (Figure 1, left).

In the kangaroo method, if the terms of the sequences of both kangaroos are drawn on a piece of paper, starting for (t_k) at the bottom left and for (w_k) at the bottom right and merging when the collision of the paths occurs, the figure one obtains has the shape of the Greek letter lambda (Figure 1, middle).

In the parallelized rho method, one works with a collection of sequences, one for each processor, and the goal is that two such sequences collide. If we draw again the terms of all sequences, the big picture shows a bundled rho, while if we zoom into the area of collision, we see a lambda (Figure 1, right).

Thus, the pictures we obtain from drawing the sequences do not suit well to distinguish between the two methods. The real difference lies in the sequences themselves. As before, let $G = \langle g \rangle$ and $h \in G$, and we consider the problem of finding $x = \log_g h$.

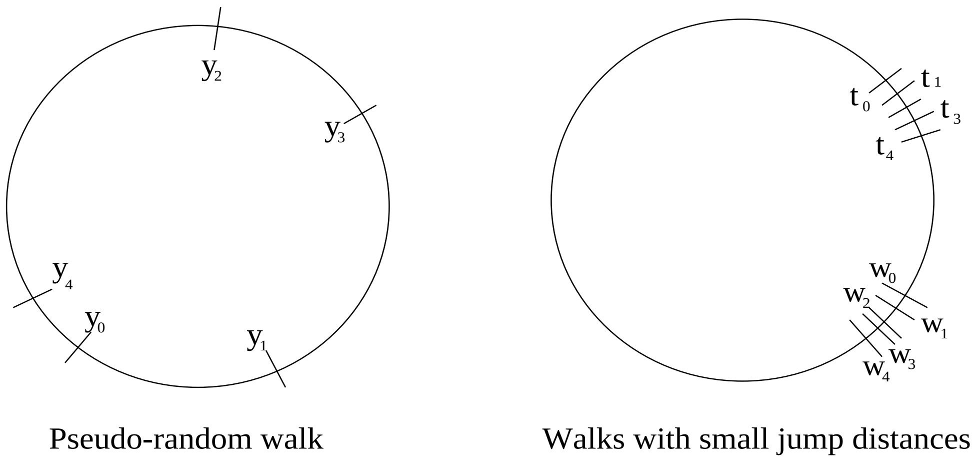

Figure 2. The walks in the Pollard rho and kangaroo methods

Both the kangaroo method and the rho method in its improved version [Tes00] work with sequences following the rule

$$z_{k+1} = z_k * m_i \ , \tag{3.1}$$

with $m_i \in \{m_1, \dots, m_r\}$, $r = O(1)$, and $i = i(z_k)$ given by a hash function mapping G to the integers $1, \dots, r$.

The characterizing feature of the rho method is that a *random walk* in G is simulated. For this, one works with $m_i = g^{u_i}$ that are *random powers* of the generator g, i.e., the u_i are thought to be integers randomly chosen from $[1, \operatorname{ord} g]$, and their mean value is about $(\operatorname{ord} g)/2$.

In the kangaroo method we work with $m_i = g^{s_i}$ where the s_i are viewed as *small distances*, in the order of $\sqrt{b-a}$ (with $a \le x = \log_g h \le b$). So we do not attempt to simulate a random walk in the whole group, but we think of the jump distances as comparatively small integers whose mean value is about $\sqrt{b-a}/2$. Thus, even if $a = 0$ and $b = \operatorname{ord} g$, that is, no information about the location of the discrete logarithm is available, the jumps in the kangaroo method are much smaller-sized than in the rho method. Note that if $a = 0$ and $b = \operatorname{ord} g$, the kangaroo method takes about 1.6 times longer than the rho method. The point of turnover is when $b - a < \pi/8 \cdot \operatorname{ord} g$.

Figure 2 illustrates both kinds of walks. Here, we view the elements of G as lying on a circle, placed equidistant starting with $g^0 = 1$ and such that clockwise, g^{i+1} comes right after g^i, and such that with $g^{\operatorname{ord} g} = 1$ we close the cycle.

Thus, from the walk (3.1) we can easily distinguish between the two methods: we speak of the rho method whenever random walks are used, and of the kangaroo method whenever walks with small jump distances are used.

4. The Distinguished Point Method

The idea of the distinguished point method is to search for a match $t_k = w_{k'}$ not among all terms of the sequences of the tame and the wild kangaroo, but only among a small subset of terms that satisfy a certain distinguishing property. It is due to van Oorschot and Wiener [vOW99].

Let $\mathcal{D}$ be a subset of G. A group element z is called a *distinguished point* if $z \in \mathcal{D}$. For our purpose, we fix an integer f and let

$$\mathcal{D} = \mathcal{D}_f = \begin{array}{l}\{z \in G \ : \ \text{the } f \text{ least significant bits} \\ \text{in the representation of } z \text{ as a binary string are zero}\} \ .\end{array}$$

If $z \in \mathcal{D}_f$, we call z a *distinguished point of degree f*. This definition has the advantage that the check whether a group element is a distinguished can be done very fast, and that we can monitor the size of $\mathcal{D}_f$ by the parameter f.

We now explain how we use distinguished points to detect collisions in the kangaroo method. We compute the sequences (t_k) and (w_k) of the tame and the wild kangaroos, where each t_k can be written as $t_k = g^{(a+b)/2+\delta_{k,\text{tame}}}$ and each w_k as $w_k = h * g^{\delta_{k,\text{wild}}}$. After each jump of a kangaroo, we check for the current term whether it belongs to $\mathcal{D}_f$. If a term is a distinguished point, the pair $(t_k, \delta_{k,\text{tame}})$ resp. $(w_k, \delta_{k,\text{wild}})$ is stored. Then we go on. The check whether a collision has occurred can be done whenever a new distinguished point is being stored (for example, we could use a hash table for this), or we could store the distinguished points in a separate file and search it for a pair $t_k = w_{k'}$ by a separate routine.

To analyze the distinguished point method, let Θ denote the proportion of group elements that are distinguished. For $\mathcal{D} = \mathcal{D}_f$, we have $\Theta = 2^{-f}$. The average running time for each kangaroo to find a distinguished point is $1/\Theta = 2^f$ jumps. That means that after the paths of the two kangaroos have collided, an expected number of $2/\Theta$ jumps is performed until that collision is detected, which together with the aforementioned van Oorschot-Wiener analysis gives an expected running time of altogether $T := 2\sqrt{b-a} + 2^{f+1}$ jumps. The expected storage requirements amount to $M := 2^{-f+1}\sqrt{b-a} + 2$ pairs $(z, l) \in G \times \mathbb{N}$ where $l = O(b-a)$ (cf. Section 6.4). Thus, if we put

$$0 \ \leq \ f = \left\lceil \log_2 \sqrt{b-a} \right\rceil - C \ , \tag{4.1}$$

for some $C \in \mathbb{N}_0$, then $\Theta \approx 2^C/\sqrt{b-a}$,

$$2\sqrt{b-a}\,(1 + 1/2^C) \ \leq \ T \ \leq \ 2\sqrt{b-a}\,(1 + 1/2^{C-1})$$

and

$$2^C + 2 \ \leq \ M \ \leq \ 2^{C+1} + 2 \ .$$

So, just as in the baby-step giant-step method we find some time-memory trade-off. The great advantage of the distinguished point method is that we need the memory only "off-line", and that we have the parameter C with which we can efficiently monitor this trade-off.

Remark 1: The above inequalities suggest that putting, for example, $C = 10$ might be a good choice to get very small storage requirements and a minor increase of the O-constant. But sometimes a larger choice for C might be necessary. For example, when distinguished points serve as checkpoints, the expected number of $\sqrt{b-a}\,/2^C$ jumps that it takes to find one distinguished point can be too large.

Remark 2: On analyzing the asymptotic performance of the distinguished point method in the setting of the rho method, Schulte-Geers [SG00] finds that the distinguished point set must be at least of "critical size" $\alpha\sqrt{A}$, where A is the cardinality of the corresponding cyclic group and α should not be too small (in fact, the larger α, the better). Applied to the kangaroo method, where the computation happens in a set of $const \cdot (b-a)$ elements (cf. Section 6.4), this means that we need $\Omega(\sqrt{b-a})$ distinguished points on the kangaroos' paths. This is met by $\Theta = \Omega(1/\sqrt{b-a})$.

5. Parallelization of the kangaroo method

Van Oorschot and Wiener [vOW99] have shown how the kangaroo method can be parallelized with linear speed-up: Assume we have m processors, m even. Then, instead of one tame and one wild kangaroo, we work with two herds of kangaroos, one herd of $m/2$ tame kangaroos, and one herd of $m/2$ wild kangaroos, with one kangaroo on each processor. The distinguished point method is used to detect a collision.

On each of the m processors, we use the same set of jumps J, where the jump distances s_i are chosen such that their mean value, say β, satisfies

$$\beta_{\text{min}} = m\sqrt{b-a}\,/4\ . \tag{5.1}$$

Let ν be an integer that indicates the spacing between members of the same herd, for example $\nu \approx \frac{1}{m/2}\beta$ (see Section 5.2 for a discussion of ν). Then, $m/2$ tame and wild kangaroos $T_1, \dots, T_{m/2}$ and $W_1, \dots, W_{m/2}$ are set off at

$$t_0(T_i) = g^{(a+b)/2+(i-1)\nu} \qquad \text{and} \qquad w_0(W_i) = h * g^{(i-1)\nu}\ , \qquad i = 1, \dots, m/2\ ,$$

one on each processor. The initial travel distances are set to $\delta_{0,\text{tame}}(T_i) = \delta_{0,\text{wild}}(W_i) = (i-1)\nu$ and each kangaroo is provided with a tag indicating whether it is tame or wild. Then they jump, and after each jump it is checked whether the new kangaroo spot is a distinguished point. If this is the case, that distinguished point together with the corresponding travel distance and the tame/wild tag is sent to a central server where it is stored, and where the check for a collision between a tame and a wild kangaroo is done. Such a collision is expected to occur after $2\sqrt{b-a}\,/m$ jumps of each kangaroo (see [vOW99] and Section 6), so that the expected running time on each processor amounts to $2\sqrt{b-a}\,/m + 1/\Theta$ group operations, where Θ as in the previous section denotes the proportion of distinguished points in $\langle g \rangle$.

Remark: If m is odd or indefinite, we can simulate $m' = 2m$ virtual processors by having one pair of wild and tame kangaroos on each processor, and letting them jump alternately. Then the same analysis as above carries over with m replaced by m', and the work on each of the m processors is just twice the work on each of the m' virtual processors.

Of course, in the above version of parallelization collisions between kangaroos of the same herd may occur. We call such collisions *useless*, and we call a collision between a tame and a wild kangaroo *useful*. A useless collision does not give any information about the discrete logarithm, and since after a useless collision the colliding kangaroos continue with exactly the same path, computing time is wasted. This effect will be studied in Section 7.

Pollard [Pol00] has developed a version of parallelization where useless collisions *cannot* occur. His idea is the following: One works with u tame and v wild kangaroos, where u and v are coprime and such that $u \approx v \approx m/2$ and $u + v \leq m$. The r jump distances in the set of jumps are multiples of uv, say $s_i = q_i uv$ for positive integers q_i, $i = 1, \dots, r$. The set-off points of the tame and wild kangaroos are $g^{(a+b)/2+iv}$ ($i = 0, \dots, u-1$) and $h * g^{ju}$ ($j = 0, \dots, v-1$), respectively. This implies that any two tame kangaroos (and also any two wild kangaroos) travel with travel distances that are in distinct residue classes modulo uv. Also, since the equation

$$\frac{a+b}{2} + iv \equiv x + ju \pmod{uv}$$

has a unique solution in i and j, there is exactly one pair of tame and wild kangaroos that travel in the same residue class modulo uv. Hence, no collisions between members of the same herd can occur, and exactly one useful collision can occur. The analysis of this variant (see [Pol00] and

Section 6) shows that the mean value of the q_i should be close to $\sqrt{(b-a)/(uv)}/2$, i.e. the mean value of the jump distances s_i should be close to

$$\beta_{\min} = \sqrt{uv(b-a)}\,/2 \tag{5.2}$$

for optimal results. Then the expected running time is $\sqrt{(b-a)/(uv)} + 1/\Theta$ jumps on each processor. If u and v are close to $m/2$, this yields approximately the same value for $\beta_{\min}$ and also approximately the same expected running time as for the van Oorschot-Wiener parallelization.

Notice that Pollard's variant of parallelization only works when the number of processors is fixed and known in advance, and all processors take part in the computation until the very end. Since from the very beginning of the computation it is determined which pair of kangaroos is the one to collide, a failure of one of the two corresponding processors would be fatal: the computation would not finish.

For both versions of parallelization, the linear speed-up has been confirmed in practice, and their performance is quite similar (see Section 8).

5.1. **The set of jumps.** The jump distances s_i must be chosen with care to obtain the theoretically predicted performance. Let $P = 1$ for the variant of van Oorschot and Wiener, and $P = uv$ for Pollard's variant, with u and v as above. Then the jump distances are of the form $s_i = q_i P$. We discuss the choice of the q_i and give two concrete choices that work well in practice.

Apart from the aforementioned condition on the mean value of the q_i, we need $\gcd\{q_1, \dots, q_k\} = 1$. This is because if the q_i have a common factor, say M, then each kangaroo travels with a travel distance that is in a fixed residue class modulo M. Thus, a tame and a wild kangaroo can meet only if they travel in the same residue class modulo M, which may cause the algorithm to take longer, or even to fail.

5.1.1. *Powers of two.* A first good choice is letting the q_i be the powers of two starting with $q_1 = 1$ and up to $q_r = 2^{r-1}$, where r is such that the mean value $(2^{r-1}-1)/r$ of the q_i is close to the optimal value β from (5.1), respectively (5.2). Notice that since β varies with the number of processors and the interval length, the number of jumps r varies as well.

Pollard has suggested powers of two as jump distances already in his 1978 paper [Pol78]. In [Pol00] he proves that in the serial case and also in his variant of parallelization this choice indeed yields the desired result. See also Section 6.6.

Remark: In practice, we choose the largest jump distance differently in order to better meet the optimal mean value. We let r be the largest integer such that $R := \lfloor (2^{r-1}-1)/r \rceil < \beta/P$ and put $q_r = r(\lfloor \beta/P \rceil - R)$. Experiments show that this leads to a slightly better performance than using exclusively powers of two.

5.1.2. *Random distances.* A second good choice consists of 20 integers $\{q_1, \dots, q_{20}\}$ randomly chosen from the interval $[1, 2\beta/P]$, subject to the conditions that they are pairwise distinct and that $\gcd\{q_1, \dots, q_{20}\} = 1$.

That we work with a set of size 20 stems from our work on the rho method [Tes00] where we showed that with 20 jump distances chosen randomly from $[1, \operatorname{ord} g]$ we have enough randomness to simulate a random walk in the cyclic group $\langle g \rangle$. In the kangaroo method, we want to simulate jumps with distances randomly chosen between 1 and $2\beta/P$. This analogy, supported by comprehensive testing in practice, suggests that 20 random jump distances from $[1, 2\beta/P]$ yield sufficiently random kangaroo paths.

5.2. **The spacings.** In the variant of parallelization by van Oorschot and Wiener we have to decide how far apart we set off kangaroos of the same herd. Here, it is simplest to choose equidistant spacings, that is, we let the tame kangaroos start at group elements $g^{(a+b)/2+i\nu}$ $(i = 0, 1, \dots, m/2-1)$ for some integer ν, and correspondingly for the wild kangaroos. On the one hand, we do not want ν to be too large. More precisely, we want that $\nu \leq \beta/(m/2)$. Then the maximum distance between any two set-off points of kangaroos of the same herd is bounded by β, which means that with one jump the hindmost kangaroo is expected to catch up with the leading kangaroo of the same herd. This enables us to view the herds as travelling clusters rather than loose collections of individual kangaroos. On the other hand, we do not want ν be too small, especially when the set of jumps contains many jumps with small distances, as it is the case if the jump distances are the powers of two. In this case, it might easily happen that small jump distances add up to make two members of the same herd collide. Indeed, this effect is quite remarkable, as our experimental results show. It therefore seems reasonable to choose ν close to $\beta/(m/2)$.

6. Analysis

We first analyze the parallelized kangaroo method in the version of van Oorschot and Wiener [vOW99], with m processors and one kangaroo on each processor, half of them tame and half of them wild. We make the (admittedly, idealistic in a widely distributed attack) assumption that all processors begin with their computation at the same time and operate with the same speed. We count the running time in terms of *iterations*, where one iteration comprises one kangaroo jump on each processor. Then one iteration requires m group operations altogether. We do not consider any ordering of these m operations.

We view the travels of the kangaroo as movements (to the right) on a line $\mathcal{L}$ where the group elements g^i are placed equidistantly and ordered by increasing exponent i.

The running time splits up into three parts: the time while the two herds of kangaroos travel in separate regions of $\mathcal{L}$, and the time when they travel in a common region, and the time from when a useful collision has occurred until it is being detected by the distinguished point method. With $Z_{\mathrm{S}}, Z_{\mathrm{C}}$ and Z_{D} we denote the respective running times on *one* processor in terms of group operations.

We assume that the spacing ν between members of the same herd is chosen small enough that the kangaroos $T_1, \dots, T_{m/2}$ and $W_1, \dots, W_{m/2}$ can be viewed as clusters on $\mathcal{L}$. Then let

$$d = \left| \frac{a+b}{2} - x \right|$$

be the distance on $\mathcal{L}$ between the two herds. Since we do not know which of the two herds is further to the right on $\mathcal{L}$, we simply speak of the leading and the following herds; it does not matter which is tame and which is wild. As before, let

$$\beta = \frac{1}{r} \sum_{i=1}^{r} s_i$$

be the mean value of the exponents s_i in the set S. Then, for the following herd, it takes on average d/β iterations to cover the distance to the set-off points of the leading herd. Hence, $Z_{\mathrm{S}} = d/\beta$. Only after that, when their paths overlap, collisions between tame and wild kangaroos can occur. Of course, at any time in the algorithm, a useless collision can occur. Here we do not consider the computing time wasted due to this. It is small, in general, and analyzed in Section 7.

To proceed, we make the simplistic assumption that in the path of each kangaroo, each jump can be viewed as a jump with a jump distance chosen randomly from the interval $[1, 2\beta]$. Then the following holds: 1.) For each leading kangaroo, in every interval on $\mathcal{L}$ of length 2β, we expect two spots of its path. 2.) Every jump of a following kangaroo hits one of these two spots with probability $2/(2\beta)$. Thus, each jump of a (fixed) following kangaroo hits the path of a (fixed) leading kangaroo with probability $1/\beta$. Assuming independence among the $m/2$ members of each herd, we conclude that for each iteration, a useful collision occurs with probability $\left(\frac{m}{2}\right)^2/\beta$. Hence, the expected running time in the second stage where the paths of both herds overlap amounts to $4\beta/m^2$ iterations. That is, $Z_{\mathrm{C}} = 4\beta/m^2$. Taking derivatives, we find that the expected running time $Z_{\mathrm{S}} + Z_{\mathrm{C}}$ is minimal when $\beta_{\min} = m\sqrt{d}/2$. But we do not know d – otherwise we could have immediately solved the DLP. However, in applications we might know the expected value of d, which we denote by $\widehat{d}$. We then choose

$$\beta_{\min} = \frac{m\sqrt{\widehat{d}}}{2} \ . \tag{6.1}$$

Then $Z_{\mathrm{S}} = 2d/(m\sqrt{\widehat{d}})$ and $Z_{\mathrm{C}} = (2\sqrt{\widehat{d}})/m$. Averaging over all integers $d \in [0, (b-a)/2]$, we obtain

$$Z_{\mathrm{S}} = Z_{\mathrm{C}} = \frac{2\sqrt{\widehat{d}}}{m} \tag{6.2}$$

and a total expected running time of $Z_{\mathrm{S}} + Z_{\mathrm{C}} = (4\sqrt{\widehat{d}})/m$ operations on each processor until a useful collision occurs. In particular, if the solution x is uniformly distributed in the interval $[a, b]$, we have $\widehat{d} = (b-a)/4$, which gives $\beta_{\min} = m\sqrt{b-a}/4$ (just as in (5.1)) and

$$Z_{\mathrm{S}} = \frac{4d}{m\sqrt{b-a}} \ , \qquad Z_{\mathrm{C}} = \frac{\sqrt{b-a}}{m} \ .$$

Again, by taking averages over all d, we find $Z_{\mathrm{S}} = Z_{\mathrm{C}} = \sqrt{b-a}/m$ and $Z_{\mathrm{S}} + Z_{\mathrm{C}} = 2\sqrt{b-a}/m$. Observe that with β as in (5.1) and for any $x \in [a, b]$ (independent of its distribution in $[a, b]$) the expected running time $Z_{\mathrm{S}} + Z_{\mathrm{C}}$ satisfies the inequalities

$$\frac{\sqrt{b-a}}{m} \ \le\ Z_{\mathrm{S}} + Z_{\mathrm{C}} \ \le\ \frac{3\sqrt{b-a}}{m} \ .$$

To obtain the corresponding results for the total expected running time, we just have to add the $Z_{\mathrm{D}} = 1/\Theta$ iterations needed to detect a collision once it has occured.

For a corresponding analysis of Pollard's variant of parallelization, we simply put $m = 2$ and replace the interval length $b - a$ by $(b-a)/(uv)$ everywhere in the above analysis. This works because we only have to consider the expected number of jumps of the two kangaroos that are destined to collide, and they travel in a fixed residue class modulo uv.

6.1. **Distribution of the running time.**

For our further considerations, we need to know about the distribution of the running time until a useful collision occurs. We here restrict ourselves to the case that only one wild and one tame kangaroo are involved. Under the assumption that when travelling in the common area, each jump of the following kangaroo hits the path of the leading kangaroo with probability $1/\beta$, the probability that *no* collision has occured after $k\beta$ iterations in the common area is

$$\mathrm{Prob}(Z_{\mathrm{C}} > k\beta) = (1 - 1/\beta)^{k\beta} \approx e^{-k} \ . \tag{6.3}$$

Now let τ denote the number of iterations until a useful collision occurs, i.e., $\tau = Z_{\mathrm{S}} + Z_{\mathrm{C}}$. Then

$$\mathrm{Prob}(\tau < k\beta) \approx 1 - e^{-(k-t)} \ ,$$

where $t\beta = Z_\mathrm{S}$ is the time spent in separate areas. If the discrete logarithm is uniformly distributed over the interval $[a, b]$, then t is uniformly distributed in $[0, 2]$. In this case,

$$\mathrm{Prob}(\tau < k\beta) \approx \begin{cases} \dfrac{1}{2}\displaystyle\int_0^k (1 - e^{-(k-t)})\,dt & \text{if } k \le 2\ , \\ \dfrac{1}{2}\displaystyle\int_0^2 \left[1 - e^{-(2-t)} + \int_2^k e^{-(x-t)}\,dx\right] dt = \dfrac{1}{2}\displaystyle\int_0^2 (1 - e^{-(k-t)})\,dt & \text{if } k > 2\ . \end{cases}$$

Integrating, we obtain

$$\mathrm{Prob}(\tau < k\beta) \approx \begin{cases} \frac{k}{2} - \frac{1}{2}(1 - e^{-k}) & \text{if } k \le 2\ , \\ 1 - \frac{1}{2}e^{-(k-2)}(1 - e^{-2}) & \text{if } k > 2\ . \end{cases} \tag{6.4}$$

From this we see that since the expected running time is 2β, the probability P_K that it takes less than K times as many iterations until a collision as expected satisfies

$$P_K \approx K - \frac{1}{2}(1 - e^{-2K}) \qquad (K \le 1)\ ,$$

while for the probability Q_K that it takes more than K times as many iterations until a collision as expected we have

$$Q_K \approx 1 - \frac{1}{2}e^{-2(K-1)}(1 - e^{-2}) \qquad (K \ge 1)\ .$$

For example, $P_{1/2} \approx 0.18$, $P_{1/10} \approx 0.94 \cdot 10^{-2}$, $P_1 \approx 0.57$, $Q_1 = 1 - P_1 \approx 0.43$, $Q_2 \approx 0.58 \cdot 10^{-1}$, and $Q_4 \approx 0.11 \cdot 10^{-2}$.

To obtain probability estimates that are independent of the distribution of the discrete logarithm in $[a, b]$ we use the fact that always $t \le 2$, and thus

$$\mathrm{Prob}(\tau < k\beta) \ge 1 - e^{-(k-2)}\ . \tag{6.5}$$

6.2. **Doing without knowledge of the group order.** Now we show that (2.1) is indeed very likely to hold over the integers. For this, assume first that only one tame and one wild kangaroo are involved. Accordingly, let $\beta = \sqrt{b-a}/2$. Let $n = \mathrm{ord}\, g$. The concern is that at the point the collision is detected one kangaroo might have travelled a total distance that is by a multiple of n larger than the other kangaroo's travel distance. Let δ_S, δ_C, δ_D denote the respective travel distances covered by the kangaroos while they travel in separate regions, while they travel in a common region but on different paths, and while they travel on the same path until a distinguished point on that path is found (i.e., $\delta_\mathrm{S} = \beta Z_\mathrm{S}$, etc). Then the total travel distance of the tame kangaroo is

$$\delta_\mathrm{tame} = \delta_\mathrm{S,tame} + \delta_\mathrm{C,tame} + \delta_\mathrm{D,tame}\ , \tag{6.6}$$

and correspondingly for the wild kangaroo. We have $\delta_\mathrm{S,tame} - \delta_\mathrm{S,wild} = x - (a+b)/2$ and $\delta_\mathrm{D,tame} = \delta_\mathrm{D,wild}$. Plugging this into (2.1), we get

$$\delta_\mathrm{C,tame} - \delta_\mathrm{C,wild} = q \cdot n\ , \qquad \text{for some } q \in \mathbb{Z}\ .$$

We need to show that $q = 0$. For this, it is sufficient to show that $\max\{\delta_\mathrm{C,tame}, \delta_\mathrm{C,wild}\} < n$. Since the expected number of jumps of each kangaroo in the common area is given by β and the average travel distance for each jump is β, it is immediate that the *expected* value of δ_C equals $\beta^2 = (b-a)/4$ (which is less than n for $b - a < 4n$).

From (6.3) we obtain that $\mathrm{Prob}(\delta_\mathrm{C} > k\beta^2) \approx e^{-k}$. Now let φ be such that $b - a \le n/\varphi$. Then $\beta^2 \le n/4\varphi$ and $\mathrm{Prob}(\delta_\mathrm{C} > kn/(4\varphi)) \le e^{-k}$. Thus, if $\varphi = 1$, then $\mathrm{Prob}(\delta_\mathrm{C} < n) \ge 1 - e^{-4} \ge 0.98$. If $\varphi = 2$, then $\mathrm{Prob}(\delta_\mathrm{C} < n) \ge 1 - 4 \cdot 10^{-3}$. Since $e^{-k} < 10^{-6}$ for $k > 13.82$, we finally have that if $\varphi = 7/2$, that is $b - a \le (\mathrm{ord}\, g)/(7/2)$, then $\delta_\mathrm{C} < \mathrm{ord}\, g$ with probability at least $1 - 10^{-6}$.

If $m > 2$ and the herds of tame and wild kangaroos are viewed as clusters, each kangaroo's travel distance δ_C in the common area also has expected value $(b-a)/4$, if $\beta = m\sqrt{b-a}/4$. The initial travel distances $\delta_0 = j\nu$, $j \in \{0, \dots, m/2-1\}$ by which the members of the same herd are spaced out do not cause any problem if $\nu \le \beta/(m/2)$ as suggested in Section 5.2. Finally, on replacing β by $\beta/(m/2)^2$ in (6.3) we find that

$$e^{-k} \approx \mathrm{Prob}(Z_C > k\beta/(m/2)^2) = \mathrm{Prob}(\delta_C > k(b-a)/4) \ , \tag{6.7}$$

and we can proceed exactly as in the case $m = 2$. This shows that (2.1) is most likely to hold over the integers if $b - a \le (\mathrm{ord}\, g)/(7/2)$, and if we only assume that $b - a \le \mathrm{ord}\, g$, then (2.1) holds over the integers with probability at least $1 - e^{-4} > 0.98$.

Recall that if $b - a \ge n\pi/8 = n \cdot 0.392\dots$, i.e., $\varphi \le 2.54$, the rho method is faster than the kangaroo method and should be used if $n = \mathrm{ord}\, g$ is known. The above discussion for such small values of φ is interesting for the case that, for example, we are only given an approximate value for $\mathrm{ord}\, g$. In this case the rho method requires us to compute the order first, while the kangaroo method could be applied to find $x = \log_g h$ directly. Notice that even if (2.1) does not hold over the integers, the kangaroo method still yields a value x such that $g^x = h$.

6.3. **Kangaroos running in cycles.**

During a kangaroo's travel, there is a possibility that the sequence of its spots becomes periodic. While cycles are the ultimate goals in the rho method, they do not reveal any information about x in the kangaroo method. Kangaroos running in cycles do not find new distinguished points, they slow down the algorithm and even might cause it to fail. We show that if $b - a \le (\mathrm{ord}\, g)/4$, this is very unlikely to happen. As before, we first assume that we have only one tame and one wild kangaroo and that we work with jump distances of mean value $\beta = \sqrt{b-a}/2$.

Let $n = \mathrm{ord}\, g$. A necessary condition for a kangaroo to end up in a cycle is that it has to travel at least a distance n in order to go around the cyclic group $\langle g \rangle$. That takes n/β iterations. From (6.5), we have $\mathrm{Prob}(\tau > n/\beta) = \mathrm{Prob}(\tau > \frac{4n}{b-a}\beta) \le e^{-(4n/(b-a)-2)}$, which is an upper bound on the probability $\mathcal{P}$ that a kangaroo ends up in a cycle before a useful collision has occured. Now let φ be such that $b - a \le n/\varphi$. Then $\mathcal{P} < e^{-(4\varphi-2)}$. If $\varphi = 1$, then $\mathcal{P} < 0.14$. If $\varphi = 2$, then $\mathcal{P} < 2.5 \cdot 10^{-3}$, and if $\varphi = 4$, then $\mathcal{P} < 10^{-6}$. This shows that a kangaroo is very unlikely to end up in a cycle if $b - a < (\mathrm{ord}\, g)/4$.

If we assume that the discrete logarithm is uniformly distributed in the interval $[a, b]$, we can use (6.4). Then we find $\mathrm{Prob}(\tau > n/\beta) = \frac{1}{2}e^{-4n/(b-a)-2}(1 - e^{-2})$ if $b - a \le 2n$, which results in bounds on $\mathcal{P}$ that are by a factor $(1 - e^{-2})/2 \approx 2.3$ smaller than above.

If $m > 2$ kangaroos are involved, then the mean jump distance is larger and kangaroos go around the cyclic group $\langle g \rangle$ faster, so that cycles might be more likely. More precisely, with $\beta = m\sqrt{b-a}/4$, we only have $\mathcal{P} < \mathrm{Prob}(\tau > n/\beta) = \mathrm{Prob}(\tau > \frac{16n}{m^2(b-a)}\beta)$. With $b - a \le n/\varphi$, and using (6.5), we obtain $\mathcal{P} < e^{-16\varphi/m^2-2}$. This shows that if, for example, we want to use $m = 1000$ processors, we need that $b - a$ be about 10^6 times smaller than $n = \mathrm{ord}\, g$ in order to exclude that kangaroos end up in cycles. Notice that if a kangaroo ends up in a cycle, this can be easily detected from the distinguished points it submits. However, to halt it requires a message from the central server to the corresponding processor, which we want to avoid in an open parallelized computation.

6.4. **Bounding the travel distances.**

It is now easy to show that if β is as in (5.1), the travel distances of all kangaroos can be bounded in terms of $b - a$, as asserted in our discussion of the storage requirements in Section 4. We write the total travel distance of a tame kangaroo as in (6.6), and correspondingly for a wild kangaroo. Notice that $\delta_{\text{tame}} = \max_{k\in\mathbb{N}}\{\delta_{k,\text{tame}}\}$ and $\delta_{\text{wild}} = \max_{k\in\mathbb{N}}\{\delta_{k,\text{wild}}\}$. Let $\delta = \max_{i=1,\dots,m/2}\{\delta_{\text{wild}}(W_i), \delta_{\text{tame}}(T_i)\}$.

For any kangaroo, $\delta_{\text{S}} \le (b-a)/2$, $\delta_{\text{D}} = m\sqrt{b-a}/(4\Theta)$, and δ_{C} follows the rule $\text{Prob}(\delta_{\text{C}} > k(b-a)/4) \approx e^{-k}$ (from (6.7)) with expected value $(b-a)/4$. Also taking into account the initial spacing ν between members of the same herd, we find that each kangaroo's expected travel distance is bounded by $\frac{3}{4}(b-a) + \frac{m\sqrt{b-a}}{4\Theta} + \left(\frac{m}{2}-1\right)\nu$. With $\Theta = 2^{-f}$ and f, C as in (4.1), we have $m\sqrt{b-a}\,/(4\Theta) \le m(b-a)/2^{C+1}$. Thus, we expect that

$$\delta \le (b-a)\left(\frac{3}{4} + \frac{m}{2^{C+1}}\right) + \left(\frac{m}{2} - 1\right)\nu \ .$$

When ν is chosen such that $\frac{m}{2}\nu \le \beta$ (see Section 5.2), the last summand can be bounded by $\beta = O(m\sqrt{b-a})$. Then, and with probability at least $1 - 10^{-6}$, we have

$$\delta < (4 + m/2^{C+1})(b-a) + O(m\sqrt{b-a}) \ .$$

6.5. **When the number of processors is not known.** The optimal choice for β depends on m, the number of processors involved. However, it is not always possible to exactly determine the number of processors that are going to participate. In this case, we need to work with an *a-priori* estimate of this number. Of course, we want that a slightly wrong such estimate does not dramatically affect the running time. We show now that this is indeed the case.

Let m be the estimate for the number of processors in a parallelized kangaroo attack. Let $\beta = m\sqrt{\hat{d}}/2$ the correspondingly optimized value for the mean jump distance. Now assume that γm machines, $\gamma > 0$, join the computation. Let $Z(\beta,\gamma)$ denote the expected running time per processor until a useful collision occurs, averaged over all $d \in [0,(b-a)/2]$. Then $Z(\beta,1) = 4\sqrt{\hat{d}}/m$ (from (6.2)). To deal with the case $\gamma \ne 1$, let as before Z_{S} and Z_{C} denote the respective expected running times per processor for the first and second stages of the kangaroos' travels. Then $Z_{\text{S}} = d/\beta$ and $Z_{\text{C}} = 4\beta/(\gamma m)^2$ and

$$Z(\beta,\gamma) = 2\sqrt{\hat{d}}\,(1 + 1/\gamma^2)/m \ .$$

We now determine the running time for the case β had been chosen to optimize the γm-processor setting. In this case, we would have worked with $\beta_{\min} = \gamma m\sqrt{\hat{d}}/2$, and then

$$Z(\beta_{\min},\gamma) = 4\sqrt{\hat{d}}/(\gamma m) \ .$$

Taking the quotient $Z(\beta,\gamma)/Z(\beta_{\min},\gamma)$, we see that by being mistaken about the number of processors by a factor γ, we end up with a running time that is by a factor

$$\sigma(\gamma) := \frac{1}{2}\left(1 + \frac{1}{\gamma}\right) \qquad (> 1 \text{ for } \gamma \ne 1)$$

longer than what corresponds to linear speed-up. For example, if we err in our estimate by a factor of 2, we experience a running time that is by a factor of $\sigma(2) = \sigma(1/2) = 5/4$ longer than in the optimal case. In other words, if we unexpectedly have twice as many processors for a computation, this still gives a speed-up of a factor 1.6 compared to our originally estimated time. For smaller errors, the loss in speed-up is even smaller. However, if we err considerably, for example, by a factor of 10, then $\sigma(10) = \sigma(1/10) = 13/5$, i.e. the running time is more than five times as long as what we could achieve with an optimized choice of β! In this case, one might want to consider a restart of the computation (especially if $\gamma \ll 1$), although such an action might result in a loss of contributing processors. The best strategy here depends on the application.

6.6. **Heuristic assumptions in the analysis.** The above analysis holds under some heuristic assumptions, which we summarize in this section.

Throughout the analysis, we always need to assume that the hash function $\langle g \rangle \to \{1, \ldots, r\}$ that picks the jumps from the set J behaves like a random function. This is reasonable given that $|\langle g \rangle| \gg r$. This is the only assumption underlying our analysis for the expected running time Z_{S}.

In our analysis of the second stage when the kangaroos' trails overlap, we work under the assumption that in each iteration, a collision between two fixed kangaroos occurs with probability $1/\beta$. This holds, for example, under the assumption that each jump is with a jump distance randomly chosen from $[1, 2\beta]$. Based on related work with the rho method [Tes00], it is reasonable to assume that this can be simulated with 20 jump distances randomly chosen from $[1, 2\beta]$.

Pollard [Pol00] gives a running time analysis that can do without this assumption: for the setting with two kangaroos, one of each kind, he proves that the expected running time of this second stage is $2\beta A(S)$ multiplications where $A(S)$ is a constant that depends on the set S of jump distances. He gives an explicit formula for $A(S)$ that can be evaluated in special cases. For example, if the jump distances are powers of 2 as in Section 5.1.1 and $6 \leq |S| \leq 20$, then $|A(S) - 1| < 0.12$, and Pollard conjectures that $A(S) \to 1$ for $|S| \to \infty$. If $A(S)$ is close to 1, then Z_{C} is close to 2β, just as what we get from (6.1) and (6.2). Unfortunately, we are not able to determine $A(S)$ if the jump distances are 20 random integers as in 5.1.2.

The analysis of the running time of the parallelized version of van Oorschot and Wiener needs that all kangaroos of each herd are mutually independent. Pollard's variant of parallelization, on the contrary, does not require any additional assumptions: Since only two processors contribute anything to the solution, the parallelized case is analyzed just as the serial case but with the interval length reduced by a factor $1/(uv)$.

7. Useless Collisions

Recall that a useless collision is a collision between kangaroos of the same herd. They can only occur in the parallelization variant by van Oorschot and Wiener.

The following simple argument shows that the expected number of useless collisions is bounded by two. We assume that $\beta = \beta_{\min}$ as in (6.1) which gives the expected running times as in (6.2). In particular, then the two herds are expected to spend as many iterations travelling in separate regions as they are expected to spend travelling in a common region. That is, useless collisions can occur during twice as many iterations as useful collisions can occur. Altogether, there are $\frac{m}{2}(\frac{m}{2} - 1)$ possible pairs for useless collisions among the $m/2$ tame kangaroos and among the $m/2$ wild kangaroos. On the other hand, there are $(m/2)^2$ possible pairs for useful collisions. Thus, for each individual iteration during the travel in the common region, a useless collision is slightly less likely to occur than a useful collision. While the herds travel in separate regions, a useless collision is as likely to occur as it is while they travel in a common region. We have one useful collision during the travel in the common region, and therefore we expect at most two useless collisions throughout the whole computation.

We now estimate the impact of useless collisions on the running time. Let $m > 4$. The first useless collision reduces the number of possible useful collisions to $\frac{m}{2}(\frac{m}{2} - 1)$, and after the second useless collision this number is $(\frac{m}{2} - 1)^2$ or $\frac{m}{2}(\frac{m}{2} - 2)$, depending on whether the collisions happened in different herds or in the same herd. Thus, if only one or two useless collisions occur, this leads to an increase in the running time by a factor of at most $\frac{m}{2}/(\frac{m}{2} - 2)$. Hence, if m is large, two useless collisions only marginally affect the running time.

We need to emphasize, however, that this reasoning relies on the randomness assumptions under which the analysis of this parallelized version has been conducted. Our experimental results confirm that on average we have less than two useless collisions. But there are unfortunate choices of the sets of jumps and the spacings where we find considerably higher numbers of useless collisions. We think, nevertheless, that these choices could be identified. See Section 8 for details.

If $m = 4$, we do expect an increase of the running time due to useless collisions. This is because after the first useless collision, the number of possible useful collisions decreases by a factor of two, from 4 to 2. If a useless collision happens before the kangaroos' regions overlap, we hence expect that in the region of overlap the expected running time is doubled. Indeed, our experimental results show average running times that are noticeably larger. We remark that also the Pollard variant of parallelization does not give fully linear speed-up if $m = 4$. There, we work with $u = 1$ tame and $v = 3$ wild kangaroos, which results in an expected running time that is by a factor $2/\sqrt{3} = 1.15\dots$ larger than what we would get from linear speed-up. Thus, if one has only 4 processors and wants to optimize the performance, one should work with more than one kangaroo on each processor. For example, simulating 16 processors by having four kangaroos jumping alternately on each machine should do it, as our experimental results suggest.

8. Experimental Results

We work with the elliptic curve E over $\mathbb{F}_p$, $p = 10^{15} + 37$ given by the equation

$$y^2 = x^3 + 5x + 19 \qquad (\mathrm{mod}\ 10^{15} + 37)\ . \tag{8.1}$$

We conduct several series of experiments, where we solve elliptic curve DLPs in the cyclic group $E_{\mathbb{F}_p}$ with the kangaroo method. We use the Computer Algebra System LiDIA [LiD] on a SunUltra Enterprise 450 under Solaris 2.6. If not indicated otherwise, we work with distinguished points of degree zero. Then $Z_{\mathrm{D}} = 0$. When measuring the running times, we also ignore the time for the precomputation of the sets of jumps, which is $O(\log(b - a))$. We work with the two sets of jumps we introduced in Section 5.1.

Our first experiment deals with the serial case, i.e. $m = 1$, $m' = 2$. We randomly choose a generator P of the curve group and let $Q = xP$ with x chosen uniformly at random from the interval $[0, 10^8]$. We use the kangaroo method with one tame and one wild kangaroo jumping alternately on the same processor to compute x. We do this 10000 times, for both sets of jumps; here the set of jumps with powers of two as jump distances has 17 elements. We find that on average, it takes

$$\begin{array}{ll} 2.07 \cdot 10^4 & \text{operations when using powers of two as jump distances ,} \\ 2.08 \cdot 10^4 & \text{operations when using random jump distances} \end{array}$$

to find the discrete log. This closely matches the theoretically predicted times.

In our next series of experiments, we assume that the discrete logarithm lies in the interval $[0, b]$, $b = 10^8, 10^{10}, 10^{12}$, and we work with $m = 20$, $m = 100$ and $m = 1000$. For each combination of b and m we repeatedly do the following: We randomly choose a generator P and let $Q = xP$ with x chosen uniformly at random from $[0, b]$. Then we use the kangaroo method with $m = 20$, $m = 100$ and $m = 1000$ kangaroos to compute x. Instead of using m processors, we simulate the parallelization on a serial processor, by successively computing jumps for all kangaroos. We apply both the variant of van Oorschot and Wiener and the variant of Pollard, with both sets of jumps. For each combination, this is done 1000 times for the two smaller values of b, and 100 times for the largest value of b. The spacings in the van Oorschot-Wiener variant are $\nu = 5003$, $\nu = 50021$ and $\nu = 500009$ for $b = 10^8, 10^{10}, 10^{12}$, respectively, so that ν is always the next prime after $2\beta_{\min}/m = \sqrt{b - a}\,/2$. The numbers of tame kangaroos in the Pollard variant are $u = 9$,

		$x \in_R [0, 10^8]$		$x \in_R [0, 10^{10}]$		$x \in_R [0, 10^{12}]$	
		vOo/Wi	Pollard	vOo/Wi	Pollard	vOo/Wi	Pollard
20 kang.	powers of 2	2.23	2.21	2.12	1.97	2.18	1.97
	random	2.26	2.12	2.13	1.93	2.17	2.01
100 kang.	powers of 2	2.17	2.14	1.97	1.91	2.01	1.81
	random	2.07	2.09	2.01	2.00	2.06	2.12
1000 kang.	powers of 2	2.18	2.06	1.94	2.01	2.09	2.17
	random	2.06	1.94	1.94	1.96	2.18	2.12

Table 1. Average performance: $\frac{\#\text{ steps}}{\text{kangaroo}} \cdot \frac{\#\text{ kangaroos}}{\sqrt{b}}$ $(b = 10^8, 10^{10}, 10^{12})$.

		$x \in_R [0, 10^8]$					
	spacing:	$\nu = 13$	$\nu = 203$	$\nu = 503$	$\nu = 2503$	$\nu = 5003$	$\nu = 10007$
100 kang.	powers of 2	4.26	2.08	2.02	1.82	1.75	1.66
	random	1.81	1.91	1.91	1.87	1.86	1.88
1000 kang.	powers of 2	51.44	7.64	5.95	2.75	2.26	1.62
	random	1.79	1.71	1.84	1.76	1.87	1.89

Table 2. Useless collisions for various choices of spacings.

$u = 49$ and $u = 499$ for $m = 20$, $m = 100$ and $m = 1000$, respectively. For $b = 10^{12}$ we work with distinguished points of degree 3, such that it takes an expected number of 8 steps to detect a collision in this case. We count the number of steps for each kangaroo, and determine their average values taken over the 1000 resp. 100 runs of the algorithm. The theory predicts that these average values, multiplied by m and divided by $\sqrt{b}$ are about 2. We give the corresponding experimental results in Table 1, where in each pair of rows the upper values give the performance with power-of-2 jump distances, and the lower values show the performance with random jump distances. Note that the performance for individual runs of the computation is pretty variable so that even after taking averages over 1000 runs, we can reproduce the corresponding values given in Table 1 only within a margin of 0.05.

Now we turn to the impact of the spacing between kangaroos of the same herd in the van Oorschot-Wiener parallelization on the number of useless collisions. Using the same set-up as before, we work with $b = 10^8$ and various spacings: $\nu = 13, 203, 503, 2503, 5003, 10007$. We use both choices for the sets of jumps, and $m = 100$ and $m = 1000$ kangaroos. The average numbers of useless collisions (averages taken over 1000 runs) are shown in Table 2. For random jump distances, the number of useless collisions seems to be independent of the spacing, while if the jump distances are powers of two, the larger the spacings, the fewer useless collisions we observe. In both cases $\nu \approx \beta/(m/2)$ seems to work well. As for the average running times when powers of 2 are used: only for $(m, \nu) = (1000, 13)$ we find a slightly larger running time than what we expect from Table 1; here the corresponding ratio is 2.30.

We now closer investigate the case $m = 4$ kangaroos. We expect fewer useless collisions, but that they have some impact on the running time. This is indeed the case, as our results in Table 3 show. There we work with interval length $b = 10^8$ and spacing $\nu = 5003$; the data for other spacings $\nu = 13, 203, \ldots$ do not show remarkable variation. We also give the corresponding performance

		$x \in_R [0, 10^8]$		
		vOorschot/Wiener		Pollard
		useless coll.	perform.	perform.
4 kang.	powers of 2	0.75	2.89	2.40
	random	0.77	2.85	2.43
8 kang.	powers of 2	1.22	2.56	2.15
	random	1.35	2.73	2.14
16 kang.	powers of 2	1.59	2.29	2.17
	random	1.57	2.23	2.05

TABLE 3. Useless collisions and average performances for $m = 4$, $m = 8$ and $m = 16$ kangaroos.

ratio for Pollard's variant of parallelization, which corresponds roughly to what the theory predicts (see Section 7). We also give results for $m = 8$ and $m = 16$ kangaroos for both versions of parallelization. The better performance in these cases suggests that it is preferable to work with 16 simulated processors if one wants to apply the parallelized kangaroo method on 4 processors.

8.1. **Conclusion.** Our experimental results suggest that both versions of parallelization – van Oorschot and Wiener versus Pollard – and both choices of jumps distances – powers of two versus random distances – perform equally well. Pollard's variant of parallelization is easier to analyze because it reduces to the serial case, and easier to handle because we do not have to deal with useless collisions and a proper choice of spacing. The method is not suitable, however, in a distributed application where we cannot foresee how many processors will be contributing or where we cannot rely on that all processors are stable until the end. Then the method by van Oorschot and Wiener can be used, and works well in practice.

9. OTHER APPLICATIONS

9.1. **Computing logarithms in arithmetic progressions.** Here we consider the following problem: we want to compute the discrete logarithm $x = \log_g h$, where we are also given an integer q and a residue class a mod q such that

$$x \equiv a \mod q \ . \tag{9.1}$$

Without loss of generality, we may assume that $q < \operatorname{ord} g$. A typical application is that q is a power of two, in which case (9.1) means that a certain number of the lowest bits are known. Pollard [Pol00] has given a method for the case that x is known modulo a number of small primes that can be composed to two coprime products u and v of approximately the same size.

Let $n = \operatorname{ord} g$, and let $l \geq 0$ such that $x = a + lq$. Then $l \leq \lfloor (n-a)/q \rfloor$, and $(g^q)^l = h * g^{-a}$. With $N = \lfloor (n-a)/q \rfloor$, $g' = g^q$ and $h' = h * g^{-a}$ the DLP $g^x = h$ is equivalent to the problem of finding l in the equation

$$(g')^l = h' \ , \tag{9.2}$$

where l is known to lie in the interval $[0, N]$. This problem can now be solved by the parallelized kangaroo method just as described earlier in this work.

Observe that in terms of the original elements g and h, the kangaroos jumping to solve (9.2) do their jumps in the same arithmetic progression modulo q. Hence, we can view this situation as a very special case of Pollard's variant of parallelization [Pol00], namely the case that we already know from the very beginning which two kangaroos meet, so that we only let these two kangaroos do their jumps.

With the rho method, the knowledge of $x \bmod q$ can only be used to decrease the running time by a factor of $\sqrt{\gcd(n,q)}$. Thus, given a and q such that $x \equiv a \mod q$, the kangaroo method applied to g' and h' is preferable to the rho method if $N < \pi/8 \cdot n/\gcd(n,q)$. This is certainly the case if $q > 2.6 \gcd(n,q)$.

9.2. **Low Hamming weight DLP.** (This is an open problem.) If $\mu = \lceil \log_2(\operatorname{ord} g) \rceil$, then the binary representation of $x = \log_g h$ requires at most μ bits, and we can write $x = \sum_{i=0}^{\mu-1} x_i 2^i$, where $x_i \in \{0,1\}$ for $0 \le i \le \mu - 1$. The number of 1's in this representation is called the *Hamming weight* of x, and is denoted by $\operatorname{wt}(x)$. For $t < \mu$, the Hamming weight t DLP is to find $x = \log_g h$ given the fact that $\operatorname{wt}(x) = t$. There exist algorithms to solve the Hamming weight t DLP using baby-step giant-step techniques [Sti], but they have large storage requirements. A space efficient algorithm is desirable, and the kangaroo method might be an approach for that. However, the open problem is how to define the paths of the kangaroos such that (1) the Hamming weight is invariant on both paths (in order to exploit the fact that $\operatorname{wt}(x)$ is known and low) and (2) a useful collision gives enough information to compute x.

9.3. **Real quadratic function fields.** The parallelized kangaroo method can also be applied to compute invariants such as the regulator and the divisor class number of real quadratic function fields [ST00]. This is remarkable because in such fields, the objects with which we deal are principal reduced ideals, which do *not* constitute a group. Instead of a group structure we find a structure known as *infrastructure*, which provides a notion of a distance associated with each reduced principal ideal, and a binary operation on the set of reduced principal ideals. These features give an arithmetic that suffices to define kangaroos and a rule for their jumping. Using the parallelized kangaroo method (together with an estimate for the divisor class number obtained from certain truncated Euler products), we were able to compute a 29-digit regulator of a random real quadratic function field, a computation that would not have been possible with baby-step giant-step methods because of memory restrictions.

References

[LiD] LiDIA Group, Technische Universität Darmstadt, Germany. *LiDIA - A library for computational number theory, Version 2.0.*

[MvOV96] A. Menezes, P. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[Pol78] J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[Pol00] J. M. Pollard. Kangaroos, Monopoly and discrete logarithms. *Journal of Cryptology*, 13:437–447, 2000.

[SG00] E. Schulte-Geers. Collision search in a random mapping: some asymptotic results. Talk at ECC 2000, The Fourth Workshop on Elliptic Curve Cryptography, Essen, Germany, 2000. Slides available from `http://www.cacr.math.uwaterloo.ca/conferences/2000/ecc2000/slides.html`.

[ST00] A. Stein and E. Teske. The parallelized Pollard kangaroo method in real quadratic function fields. To appear in *Mathematics of Computation.*

[Sti] D. Stinson. Some baby-step giant-step algorithms for the low Hamming weight discrete logarithm problem. To appear in *Mathematics of Computation.*

[Tes00] E. Teske. On random walks for Pollard's rho method. *Mathematics of Computation*, 70:809–825, 2001.

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

Dept. of Combinatorics and Optimization, University of Waterloo, Waterloo, ON N2L 3G1, Canada

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