SQUARE-ROOT ALGORITHMS FOR THE DISCRETE LOGARITHM PROBLEM (A SURVEY)

EDLYN TESKE

Abstract. The best algorithms to compute discrete logarithms in arbitrary groups (of prime order) are the baby-step giant-step method, the rho method and the kangaroo method. The first two have (expected) running time $O(\sqrt{n})$ group operations (n denoting the group order), thereby matching Shoup's lower bounds. While the baby-step giant-step method is deterministic but with large memory requirements, the rho and the kangaroo method are probabilistic but can be implemented very space efficiently, and they can be parallelized with linear speed-up. In this paper, we present the state of the art in these methods.

1. Introduction

The discrete logarithm problem (DLP) is one of the most important mathematical problems in temporary cryptography. Important public-key cryptosystems, such as the Diffie-Hellman key exchange protocol, the ElGamal encryption and signature schemes, the U.S. Government's Digital Signature Algorithm (DSA), and its elliptic curve analogue (ECDSA), rely on the difficulty of the DLP [MvOV96]. These and other systems are being deployed on the Internet (to enable secure electronic mail, home banking, and Internet browsers), in the financial services industry (in electronic cash applications, credit card transactions, and instant teller banking), in wireless communications (such as pagers and cellular telephones) and in telecommunications (fax encryptors, modems, secure telephones, cable TV and pay-per-view). The hardness of the DLP is not proven, but researchers are confident that discrete-logarithm-based cryptographic systems provide sufficient security if the parameters are chosen appropriately. For an excellent survey on the discrete logarithm problem in cryptographically significant groups such as groups of points on elliptic curves over finite fields and multiplicative groups of finite fields, see [Odl00].

For the moment, we define the DLP as follows: Let g be a generator of a finite cyclic group denoted by $\langle g\rangle$, and let $h \in \langle g\rangle$. Find the least positive integer x such that $g^x = h$. Then we write $x = \log_g h$. Let $\operatorname{ord} g$ denote the order of g, that is $\operatorname{ord} g = \log_g 1$. Then using exhaustive search, it takes at most $\operatorname{ord} g$ multiplications and equality checks in $\langle g\rangle$ to solve the DLP.

In this survey, we consider *generic square-root algorithms* for the DLP. Here, a generic algorithm is an algorithm that does not exploit any special properties of the objects to which it is applied. More precisely, a generic algorithm for a group G uses only that

Date: January 26, 2001.

1991 *Mathematics Subject Classification.* 11Y16, 94A60.

- Each group element has a unique encoding as a binary string.
- We have a "black box" oracle for the group operations: multiplication, inversion:
  1. For all $g, h \in G$ we can compute the product $g * h$.
  2. For each $g \in G$ we can compute the inverse g^{-1}.

Notice that from every group element g we can determine the identity element $1 = g * g^{-1}$. Further, the unique encoding of the group elements implies the possibility of an equality check: given $g, h \in G$ we can decide whether $g = h$. The running time complexity of a generic algorithm is measured by counting the number of group multiplications, inversions and equality checks. Similarly, space requirements of generic algorithms are measured in terms of group elements to store. Now, let n be the number of group operations one is expected to invest to solve a certain computational problem using exhaustive search. Then we call an algorithm a square-root algorithm if it needs to perform (expected) $O(\sqrt{n}\,)$ group operations to compute the solution.

All generic square-root algorithms for the DLP known to date are based on only a few methods: the baby-step giant-step method due to Shanks [Sha71], and the rho method and the kangaroo method, due to Pollard [Pol78]. The baby-step giant-step method is a deterministic method that uses a time-memory trade-off and takes $const \cdot \sqrt{\operatorname{ord} g}$ group operations and has to store $const \cdot \sqrt{\operatorname{ord} g}$ group elements. The rho method is a probabilistic method based on the birthday paradoxon. It can be implemented very space efficient, with only a constant number of group elements to store and, under certain heuristic assumptions, has expected running time $const \cdot \sqrt{\operatorname{ord} g}$ group operations. The kangaroo method is also a probabilistic method designed to compute the discrete logarithm when it is known to lie in an interval $[a, b]$. It also can be implemented very space efficient and has expected running time $const \cdot \sqrt{b-a}$ group operations, again under certain heuristic assumptions.

Observe that the index calculus is not a generic method; it exploits specifics of the representation of its elements. Index calculus algorithms can be applied to solve the DLP in certain finite abelian groups, including $(\mathbb{Z}/p\mathbb{Z})^*$, ideal class groups of quadratic orders and divisor class groups of hyperelliptic function fields of large genus. They have subexponential running time. What is important for cryptography is that there are no index calculus methods known for elliptic curve groups (with a randomly chosen elliptic curve) and subgroups of finite fields. More precisely, the only known algorithms for these groups are the aforementioned square-root algorithms.

Shoup [Sho97] has shown that if p is the largest prime dividing the group order, a generic algorithm to solve the DLP with a probability bounded away from zero has to perform at least $const \cdot \sqrt{p}$ group operations. Here it is assumed that $\operatorname{ord} g$ and its factorization into primes is known. In this case, the Pohlig-Hellman method [PH78] applies which is a generic method to reduce the DLP in a group of order n to DLPs in groups of order p where p runs through all the prime factors of n.

So, while Shoup has established an $\Omega(\sqrt{p}\,)$ lower bound for the DLP, the existence of the above generic square-root algorithms implies that we cannot find any group where the DLP is harder than $O(\sqrt{p}\,)$. Thus, given our current knowledge, the DLP in elliptic curve groups and subgroups of multiplicative groups of finite fields is (up to constant factors) as hard as a DLP can be.

In the following, we describe the generic square-root algorithms known to date. We summarize the recent advances that lead to optimized running times, thereby reducing the original running times by constant factors. We show extensions to DLPs where various degrees of knowledge about the discrete logarithm are exploited. In the case of the probabilistic algorithms, we also address the heuristic assumptions underlying the running time analysis, and recent advances in reducing these assumptions. In the next section we re-define the DLP according to what additional knowledge is available. Then, in Section 3 we deal with the baby-step giant-step method, in Section 4 we discuss the rho method, and in Section 5 we cover the kangaroo method.

Acknowledgments. The author wishes to thank Douglas Stinson for a helpful discussion about the low Hamming weight DLP, and Ramarathnam Venkatesan for helpful comments on Cayley graphs and discrete logarithms.

2. Degrees of Knowledge

A standard assumption in cryptographic applications is that we know $\operatorname{ord} g$, which implies that $x \in [1, \operatorname{ord} g]$, and that x is chosen uniformly at random from that interval; we call the corresponding DLP the *standard-DLP*. But there are settings where this is not the case. For example, when $h = 1$ then finding $x = \log_g h$ means finding (a multiple of) the order of g. In this case we certainly do not know $\operatorname{ord} g$, but instead we might have some other knowledge about x, such as factors of x, or an approximation. We call the DLP where we do not know $\operatorname{ord} g$ the *unknown-order-DLP*. There are also applications where we have additional knowledge about $x = \log_g h$, such as a certain amount of high or low order bits, or a probability distribution of x. Such knowledge can be exploited to reduce the running time for solving the DLP, which needs to be taken into account in implementations of cryptographic systems based on the DLP.

In the following, we define various DLPs according to what is known about the discrete logarithm prior to the computation. Only in the case of the first DLP below we are not even sure about the mere existence of the discrete logarithms. In all the other definitions, we assume that $x = \log_h g$ exists.

2.1. The decision-DLP.

Definition 2.1. The *decision-DLP* is the following: Given group elements g and h. Decide whether there exists an x such that $g^x = h$. If this is the case, find x.

Note that a necessary condition for $h \in \langle g \rangle$ is that $h^{\operatorname{ord} g} = 1$. This can be used as a quick test in the case $\operatorname{ord} g$ is known.

The decision-DLP can be solved in running time $O(\sqrt{\operatorname{ord} g}\,)$ with space requirements $O(\sqrt{\operatorname{ord} g}\,)$.

2.2. **An interval is known.**

Definition 2.2. Let a and b be integers, $0 \leq a < b$, such that $x \in [a,b]$. The *interval-$[a,b]$-DLP* is: given a, b, g and $h = g^x$, find x.

If $a = 1$ and $b = \operatorname{ord} g$, the interval-$[a,b]$-DLP is just the standard-DLP. A typical cryptographic application is that $a = 0$ and $b \ll \operatorname{ord} g$, so that a certain number of high-order bits of x are known to be zero. The reason why one would choose the secret key x of such a special form is that then the discrete exponentiation g^x is faster than for randomly chosen x. In the case that $h = 1$, an interval for $x = \operatorname{ord} g$ may be obtained from specific properties of the group. For example, for a cyclic elliptic curve groups over $\mathbb{F}_q$ generated by some point g we know that $|\operatorname{ord} g - (q+1)| \leq 2\sqrt{q}$. Similar bounds hold for the number of points on the Jacobian of a hyperelliptic curve.

The interval-$[a,b]$-DLP can be solved in running time $O(\sqrt{b-a}\,)$ with space requirements $O(\sqrt{b-a}\,)$, or in expected running time $O(\sqrt{b-a}\,)$ with constant space requirements.

2.3. **The Hamming weight is known.** 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 $0 \leq i \leq \mu - 1$. The number of 1's in this representation is called the *Hamming weight* of x, and is denoted by $\operatorname{wt}(x)$.

Definition 2.3. Let $t < \mu$ and such that $\operatorname{wt}(x) = t$. Then the *Hamming-weight-t-DLP* is: given μ, t, g and $h = g^x$, find x.

In a cryptographic application, working with an exponent x of (sufficiently) low Hamming weight has the advantage that the exponentiation g^x is faster than with a randomly chosen x.

The Hamming-weight-t-DLP can be solved in running time $O(\mu\binom{\mu/2}{t/2})$ with space requirements $O(\binom{\mu/2}{t/2})$, or in expected running time $O(\sqrt{t}\binom{\mu/2}{t/2})$ with space requirements $O(\binom{\mu/2}{t/2})$.

2.4. **The solution lies in some known residue class.** Let q be an integer, $2 \leq q < \operatorname{ord} g$.

Definition 2.4. Let a, q and x be such that $x \equiv a \mod q$. The *mod-q-DLP* is the following: Given a, q, g and $h = g^x$, find x.

For example, q can be a power of two, which means that a certain number of the lowest bits of x are known.

Observe that if $\operatorname{ord} g$ (or an approximate upper bound of $\operatorname{ord} g$) is known, the mod-q-DLP can be easily reduced to an interval-$[a,b]$-DLP. This works as follows: Without loss of generality, let $q \leq n := \operatorname{ord} g$. For $x \equiv c \mod q$, let $l \geq 0$ such that $x = c + lq$. Then $l \leq \lfloor (n-c)/q \rfloor$, and $(g^q)^l = h * g^{-c}$. Thus, with $N = \lfloor (n-c)/q \rfloor$, $g' = g^q$ and $h' = h * g^{-c}$ the DLP $g^x = h$ is equivalent to the problem of finding l in the equation $(g')^l = h'$, where l is known to lie in the interval $[0, N]$.

Accordingly, the mod-q-DLP can be solved in (expected) running time $O(\sqrt{\operatorname{ord} g/q}\,)$.

2.5. **The probability distribution is known.** Let $p_0, p_1, \ldots$ be non-negative real numbers such that $\sum_{i=0}^{\infty} p_i = 1$.

Definition 2.5. Assume $h = g^x$ where x is chosen at random with respect to the probability distribution given by (p_i), where $p_i = \text{Prob}(x = i)$. The *distribution-(p_i)-DLP* is: Given (p_i), g and h, find x.

If $p_i = 1/\operatorname{ord} g$ for $1 \le i \le \operatorname{ord} g$ and $p_i = 0$ otherwise, then the distribution-(p_i)-DLP becomes the standard-DLP.

A related scenario is when we just know the expected value of the discrete logarithm x. This gives rise to the following definition:

Definition 2.6. Let $E(x)$ be the expected value of x. The *$E(x)$-DLP* is: given $E(x)$, g and $h = g^x$, find x.

The $E(x)$-DLP can be solved in expected running time $O(\sqrt{E(x)}\,)$.

3. Baby-Step Giant-Step Methods

The baby-step giant-step method has been introduced by D. Shanks [Sha71] to compute ideal class numbers of quadratic number fields. Shanks was looking for certain multiples of element orders that were known to lie in certain residue classes and were expected to lie in certain intervals.

Applied to the standard DLP, the baby-step giant-step method can be described as follows: Let g and h be such that $h = g^x$ for some unknown x and let n be an upper bound on x, for example $n = \operatorname{ord} g$. Let $q = \lceil \sqrt{n}\, \rceil$. Then we can write

$$x = dq + s, \qquad \text{where } 0 \le d \le q \text{ and } 0 \le s < q.$$

We need to find d and s, which are unique. For this, for $r = 0, 1, \ldots, q-1$ one computes the group elements $h * g^{-r}$. When doing this, for each r one checks whether $h * g^{-r} = 1$. If this is the case, we set $x = r$ and stop. Otherwise, this element is stored in a table R, together with the corresponding number r. We end up with $R = \{(h * g^{-r}, r) : 0 \le r \le q-1\}$. These are the *baby steps*. Then for $c = 1, 2, \ldots, q$ one computes g^{cq}. These are the *giant steps*. For each c, one checks whether $(g^{cq}, r) \in R$ for some $r \in \{0, 1, \ldots, q-1\}$. As soon as this happens, $\log_g h$ is found, since then $g^{cq+r} = h$. Note that $cq + r$ is the smallest positive number for which this is possible. Thus, set $d = c$ and $s = r$.

If $x \ge q$, this algorithm requires one inversion and $q + \lfloor x/q \rfloor + O(\log q)$ group multiplications. One has q elements to store. It requires $\lfloor x/q \rfloor$ table look-ups in a table of size q. Since the group elements can be encoded as unique binary strings, hashing on the group elements is possible so that the table-look ups can be executed efficiently.

The unknown-order DLP can be solved in a similar way. Here we work with an assumed bound n on x and compute a baby-step set as above. Then we compute giant steps until the solution is found. This works even if $n < x$. The running times are the same as above, but note that when $n < x$, the complexity might be worse than $O(\sqrt{\operatorname{ord} g}\,)$.

The baby-step giant-step method can also be used to solve the decision-DLP. If $\operatorname{ord} g$ is known, we proceed just as in case of the standard-DLP, and stop when $x = \log_h g$ is found, or when q giant steps have been executed, whatever happens first. If $\operatorname{ord} g$ is not known, we use an assumed bound n and work with a baby-step set $R = \{(g^{-r}, r) : 0 \leq r \leq q-1\}$. Before adding a new element g^{-r} to R, we check whether $g^{-r} = h^{-1}$ or $g^{-r} = 1$. If one of these is the case, we have found the discrete logarithm, or have found that it does not exists. Otherwise, we proceed. Now, for each giant step element g^{cq}, we check whether $(h^{-1} * g^{cq}, r) \in R$ form some r and, if this is not the case, whether $(g^{cq}, r) \in R$. Eventually, one of these happens. The running time for this is two inversions and $q + 2\lfloor x/q \rfloor + O(\log q)$ multiplications. Here, $x = \min\{\log_g h, \operatorname{ord} g\}$, where $\log_g h = \infty$ if $h \notin \langle g \rangle$.

The interval-$[a,b]$-DLP can be solved by putting $n = b - a$, $q = \lceil n \rceil$ and letting (g^{-a}, a) be the first element in the baby-step set R. Finding $x = \log_g h$ then takes one inversion and $q + \lfloor (x-a)/q \rfloor + O(\log(aq))$ multiplications.

3.1. **Dynamic increase of the giant-step width.** By a modification of the above method it is possible to obtain complexity results relative to the discrete logarithm rather than relative to an upper bound on it. We discuss two such approaches, which work with increasing the giant-step width in the course of the algorithm. The first [BJT97] works with doubling the step width at certain intervals. We describe a simplified version, which is based on the following statement:

Lemma 3.1. [BJT97, Lemma 2.1] *For every positive integer x there are uniquely determined integers k, d and s with $k \geq 0$, $\lfloor 2^{k-1} \rfloor v^2 \leq d < 2^{k+1}$ and $1 \leq s \leq 2^{k+1}$ such that $x = 2^{k+1}d + s$.*

Notice that an integer x can be written in the form $2^{k+1}d + s$ with d as in Lemma 3.1 if and only if x lies in the interval $I_k = \{\lfloor 2^{k-1} \rfloor 2^{k+1}, \ldots, 4^{k+1}\}$.

We next explain how to solve the standard-DLP. The idea to compute $x = \log_g h$ is to search those intervals for $k = 0, 1, 2, \ldots$ until x is found. For a fixed k, to check whether $x \in I_k$ we test whether $g^{y+d} = h$ with $y = 2^{k+1}d$, and s and d as in Lemma 3.1. For this purpose, we compute the baby-step set $R_k = \{(h * g^{-r}, r) : 1 \leq r \leq 2^{k+1}\}$. Then for $c = \lfloor 2^{k-1} \rfloor, \lfloor 2^{k-1} \rfloor + 1, \ldots, 2^{k+1}$ we compute $g^{2^{k+1}c}$. The elements g^y with $y = 2^{k+1}c$ are the giant steps; their step width is 2^{k+1}. For each such giant step we check whether there exists $(g^y, r) \in R_k$ for some r. If so, $\log_g h = y + r$. When we have searched the whole interval I_k without finding x, we increase k by 1, extend the baby-step set accordingly and perform the next set of giant steps.

This algorithm can be implemented such that it executes one inversion and at most $4\lceil \sqrt{x}\, \rceil + O(\log x)$ multiplications. One has at most $2\lceil \sqrt{x}\, \rceil$ elements to store, and it requires at most $2\lceil \sqrt{x}\, \rceil$ table look-ups.

The second approach uses giant steps whose step width is incremented after *each* baby step [Ter00]. It is based on the following way of representing integers:

Lemma 3.2. *For every non-negative integer x there are uniquely determined integers j and t with $0 \leq t < j$ such that $x = T_{j+1} - t$, where T_j is the j-th triangle number, i.e., $T_1 = 0$, $T_{j+1} = T_j + j$ for $j \geq 1$.*

We show how to solve the standard-DLP. To compute $x = \log_g h$, we need to find the correct value of j in the representation above. This is done alternating baby steps and giant steps as follows: In the beginning, we compute the baby step $g^0 = 1$ and the giant step g^0; the baby step is stored as pair $(g^0, 0)$. Now for $j \geq 1$, the j-th iteration consists of computing the baby step $h * g^j = h * g^{j-1} * g$ (which is stored as pair (g^j, j) in the baby step set R) and the giant step $g^{T_{j+1}} = g^{T_j} * g^j$. If $j \geq 2$, we check whether $(g^{T_{j+1}}, t) \in R$ for some t, $0 \leq t < j$. If this is the case, we put $x = T_{j+1} - t$ and stop. This algorithm terminates when $T_j < x \leq T_{j+1}$. Since $T_{j+1} = j(j+1)/2$ for $j \geq 0$, this happens if $j = \lfloor \sqrt{2x + 1/4} \rceil$.[1]

Since this method requires as many baby steps as giant steps it requires $2\lfloor \sqrt{2x + 1/4} \rceil$ group multiplications and $\lfloor \sqrt{2x + 1/4} \rceil$ table look-ups in a baby-step set of maximum size $\lfloor \sqrt{2x + 1/4} \rceil$. Although these values are by a factor of $\sqrt{2}$ smaller than the upper bounds given for the previous algorithms, in practice both methods perform equally well (see [Ter00]).

Observe that in both variants [BJT97, Ter00], the algorithm for the standard-DLP is the same as for the unknown-order-DLP. The modifications for the decision-DLP and the interval-$[a, b]$-DLP are similar as for the original method, and given above.

3.2. **A systematic approach.** The original version of baby-step giant-step method with $n = \operatorname{ord} g$ is optimal in the sense that in the worst case, that is, $x \approx n$, it takes the least number of group operations. The modifications [BJT97] and [Ter00] have the advantages that their running times are lower if $x \ll n$. The points of turnover are $x = n/14$ in the case of [BJT97] and $x = n/6$ in the case of [Ter00]. Blackburn and Teske [BT00] took a systematic approach and asked: given the probability distribution for the solution x on $\mathbb{N}_0$, what is the baby-step giant-step strategy that computes x using the smallest expected number of operations? The answer that optimizes the baby-step giant-step method for the distribution-(p_i)-DLP is as follows:

If the distribution has an increasing hazard rate[2] (IHR), which is the case for the uniform, the normal and the geometric distributions, the optimal strategy is to compute $q = \sqrt{E(x)}$ baby steps at the beginning, and then proceed exclusively with giant steps; here, $E(x)$ denotes the expected value of x. In particular, if x is uniformly distributed over $[1, \operatorname{ord} g]$, then the choice that minimizes the average running time is $q = \sqrt{\operatorname{ord} g/2}$. Thus, remarkably, for IHR distributions the distribution-(p_i)-DLP cannot be solved any faster than the $E(x)$-DLP. In other words, the additional knowledge of the exact probability distribution cannot be exploited to achieve lower expected running times. Only if the distribution has a decreasing hazard rate, which is the case for the discrete Pareto or the Weibull distribution, the optimal strategy is to gradually increase the giant-step width. This optimal strategy is completely determined by the p_i, and there is an explicit formula for it [BT00]. In particular, Terr's method [Ter00] is close to optimal if the distribution of the discrete logarithm is a discrete Pareto distribution.

[1] $\lfloor y \rceil$ is the unique integer y such that $-\frac{1}{2} < y - \lfloor y \rceil \leq \frac{1}{2}$.

[2] For a probability distribution (p_i), the *hazard rate* (a.k.a. *failure rate*) at k measures the likelihood of x lying in a small interval beyond k, given that $x \geq k$. It is defined as $p_k / (\sum_{i=k}^{\infty} p_i)$ and is the reciprocal of the Mills' ratio.

3.3. **Low Hamming weight.** The Hamming-weight-t-DLP can be solved by a probabilistic algorithm as follows. Let $X = \mathbb{Z}_\mu$. For a set $Y \subset 2^X$ let $\mathrm{val}(Y) = \sum_{i \in Y} 2^i \in \{0, 1, \ldots, 2^\mu - 1\}$. Assume for simplicity that both μ and t are even. Now we pick a random subset B of X with $\mu/2$ elements and compute the "baby"-step set $R_B = \{(g^{\mathrm{val}(Y)}, \mathrm{val}(Y)) : Y \subset B, |Y| = t/2\}$. Then for all subsets Z of $X \setminus B$ such that $|Z| = t/2$, we check whether $(h * (g^{\mathrm{val}(Z)})^{-1}, r) \in R_B$ for some r. As soon as this happens for some Z, we put $x = r + \mathrm{val}(Z)$ and stop. If this is the case for no such Z, we pick another random subset $B \subset X$, $|B| = \mu/2$, and repeat the computation.

We can efficiently compute R_B using a minimal change ordering on $\{Y \subset B, |Y| = t/2\}$ where any two successive sets differ by only two elements (cf. [KS99]). Then, working with a precomputed list $\{g^{2^i} : i = 0, \ldots, \mu - 1\}$, we can obtain the next term $g^{\mathrm{val}(Y)}$ from the previous one just by multiplying by g^{2^j} and dividing by g^{2^k} for some j and k. Thus, for each B, the computation of R_B takes at most $2\binom{\mu/2}{t/2} + O(t)$ group operations. A similar method can be applied to compute the elements $h * (g^{\mathrm{val}(Z)})^{-1}$ for all $Z \subset X \setminus B$ with $|Z| = t/2$. Hence, for each B we need to perform at most $4\binom{\mu/2}{t/2} + O(t)$ group operations and at most $\binom{\mu/2}{t/2}$ table look-ups in a table of size $\binom{\mu/2}{t/2}$. Stinson [Sti] shows that this algorithm, which is due to Coppersmith, is expected to succeed after $O(\sqrt{t}\,)$ sets B have been considered. Thus, the expected running time is $O(\sqrt{t}\binom{\mu/2}{t/2})$ group operations.

A deterministic algorithm can be obtained if instead of randomly picking $B \subset X$ we use (μ, t)-splitting systems. This algorithm is up to a factor of $\mu/\sqrt{t}$ slower. See [Sti] for details.

3.4. **Parallelization.** If the baby-step giant-step method is applied parallel on m processors, a speed-up by a factor of $\sqrt{m}$ can be easily achieved, simply by dividing the interval $[1, n]$ into m approximately equally long subintervals and applying the method independently on each of these subintervals. Unless the m processors have fully shared memory with unlimited and immediate access to it a $\sqrt{m}$-fold speed-up is the best one can achieve. In other words, the baby-step giant-step method cannot be efficiently parallelized.

4. Pollard's Rho Method

The rho method has been developed by J. Pollard [Pol78] to compute discrete logarithms in $(\mathbb{Z}/p\mathbb{Z})^*$.

The key idea is that for any finite set W and mapping $F : W \to W$, the sequence $(w_k)_{k \in \mathbb{N}_0}$ formed by the rule $w_0 \in W$, $w_{k+1} = F(w_k)$ $(k \in \mathbb{N}_0)$ 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 $w_k = w_{k+\lambda}$, $k \geq \mu$. Under the assumption that $w_0 \in W$ is randomly chosen (with respect to the uniform distribution) and F is a random mapping[3], the expected values for μ and λ are both close to $\sqrt{\pi|W|/8} = 0.626\ldots\sqrt{|W|}$ ([Har60]). The name *rho method* stems from the picture one obtains when drawing the terms of (w_k), starting at the bottom and ending in a cycle, which shows the Greek letter rho.

[3] A *random mapping* is a mapping that is chosen uniformly at random from the set of all $|W|^{|W|}$ mappings $F : W \to W$

We describe the rho method in $(\mathbb{Z}/p\mathbb{Z})^*$. Let p be a prime, let h be any integer, and let g be a primitive root modulo p. To compute $x = \log_g h \pmod p$, Pollard [Pol78] used the iterating function $F : (\mathbb{Z}/p\mathbb{Z})^* \to (\mathbb{Z}/p\mathbb{Z})^*$ given by

$$F(y) = \begin{cases} gy & \text{if } 0 < y \le p/3 \ , \\ y^2 & \text{if } p/3 < y \le 2p/3 \ , \\ hy & \text{if } 2p/3 < y < p \ . \end{cases} \tag{4.1}$$

and defined a sequence (y_k) according to the rule $y_0 = 1$, $y_{k+1} \equiv F(y_k) \pmod p$. 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. Notice that the reduction modulo $p-1$ is necessary because otherwise the doubling of α_k and β_k would cause an explosion of the exponents.

A match, that is, two terms y_k, y_l with $y_k = y_l$ and $k \neq l$, leads to an equation $\alpha_k + x\beta_k \equiv \alpha_l + x\beta_l \pmod{p-1}$, from which we can compute x using the Extended Euclidean algorithm if $\gcd(p-1, \beta_k - \beta_l) = 1$.

To find a match, there is, for example, an efficient algorithm due to Brent [Bre80]. It uses an auxiliary variable, say w, which at each stage of the algorithm holds y_j, where j is the largest power of two strictly less than the current index of the sequence k. At the beginning we have $w = h_0$. For each newly computed term of the sequence, the algorithm checks for equality with w. As soon as the current index k equals a power of two, we assign $w = y_k$. This is repeated until a match (y_k, w) is found.

Brent's algorithm.
Input: Initial value y_0, iterating function $F : W \to W$.
Output: j and k such that $y_j = y_k$.

$y = h_0$; $r = 1$; $k = 0$; done = false
repeat
  $w = y$, $j = k$
  $r = 2r$
  repeat
    $k = k + 1$, $y = F(y)$
    done= $(w = y)$
  until (done) or $(k \ge r)$
until (done)
return j, k

Under the assumption that F is a random mapping, the first match is expected to occur after $\sqrt{\pi n/2} \approx 1.25\sqrt{n}$ iterations ($n = \operatorname{ord} g$), and then with Brent's algorithm the first match is found after an expected number of $\approx 1.97\sqrt{n}$ iterations. Modifications of Brent's algorithm that require slightly more storage but less iterations can be found in [SL84] and [Tes98a].

Pollard's original application can easily be generalized to any other group. All we need is a rule how to partition the group into 3 disjoint sets of equally large size, which can be done based on the unique encoding of the group elements as binary strings. Experiments show, however, that iterating functions of the form (4.1) do not behave like random functions. For example, in $(\mathbb{Z}/p\mathbb{Z})^*$ with the rule of partitioning given above, matches occur only after an average number of $1.37\sqrt{p-1}$ iterations. In other groups, or for other ways of partitioning $(\mathbb{Z}/p\mathbb{Z})^*$, they occur even later, after an (experimental) average of $\approx 1.56\sqrt{n}$ iterations, where $n = \operatorname{ord} g$ (see [Tes00]).

4.1. **Better random walks.** Better walks for the rho method have been introduced that yield the same performance as we expect from a random mapping, which in the case of an arbitrary group means a speed-up by about 20% compared to a function of the type (4.1). Let g, h be group elements, $G = \langle g, h\rangle$. Let r be a small positive integer: $r \in [3, 100]$. We need a partitioning of $G = T_1 \cup \cdots \cup T_r$ with roughly equally large sets T_s. For this, we take a hash function $v : G \to \{1, \ldots, r\}$ and define $T_s := \{y \in G \ : \ v(y) = s\}$, $\quad s = 1, \ldots, r$. A popular example (see [Tes98b]) is the 20-adding walk, which works with $r = 20$ sets T_s and is generated by an iterating function of the form $F(y) = y * M_{v(y)}$, where the multipliers $M_1, \ldots, M_{20}$ are computed according to the rule $M_s = g^{m_s} * h^{n_s}$ with m_s and n_s randomly chosen from $\{1, \ldots, |G|\}$. We compute y_0 by taking a random integer in $\{1, \ldots, |G|\}$ and putting $y_0 = g^{\alpha}$, and then iteratively compute the sequence (y_k) following the rule $y_{k+1} = F(y_k)$. Then each term y_k can be represented in the form $y_k = g^{\alpha_k} * h^{\beta_k}$, where $\beta_0 = 0$ and

$$\alpha_{k+1} = \alpha_k + m_{v(Y_k)} \qquad \text{and} \qquad \beta_{k+1} = \beta_k + n_{v(Y_k)} \ . \tag{4.2}$$

These addings (whence the name of the walks) are computed modulo $|G|$. However, since the increase of the terms in (4.2) 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 means that r-adding walks can be computed even if the group order is not known. Indeed, 8-adding walks have been first used by Sattler and Schnorr [SS85] to compute element orders.

In experiments with cyclic elliptic curve (sub)groups, matches in 20-adding walks occur after an average of $1.26\sqrt{n}$ iterations. This is approximately the performance of a random walk.

4.2. **Degrees of knowledge.** Let us now look at what knowledge is needed to apply the rho method, and to what extent additional knowledge can be exploited. In principle, the rho method only suits to solve the standard-DLP.

First of all, even if the sequence (y_k) and the corresponding sequences of exponents can be computed without knowledge of the order $n = \operatorname{ord} g$, we need to know n to find the discrete logarithm once a match has occured. This is because a match only gives an equation of the form $bx \equiv a \mod n$, which is very unlikely to hold over the integers, and thus can only be solved for x if n is known. (One certainly can, however, apply the rho method twice, once to compute $\operatorname{ord} g$ and then to compute $x = \log_g h$.)

Also, the rho method cannot be used to solve the decision-DLP. Assume we do not know whether $h \in \langle g\rangle$. W.l.o.g. we can assume that $\operatorname{ord} h$ divides $ordg$. We apply the rho method until we find a match $g^{\alpha_l} * h^{\beta_l} = g^{\alpha_k} * h^{\beta_k}$. Let $a = \alpha_l - \alpha_k \pmod{|G|}$ and $b = \beta_k - \beta_l$. Then $g^a = h^b$. Let $n = \operatorname{ord} g$ and let $c = \gcd(b, n)$. If $c = 1$, we know that $h \in \langle g\rangle$. In any case, we use the extended Euclidean algorithm to compute two integers s and t such that $c = sb + tn$ and obtain that $g^{sa} = h^c$. If $c \neq 1$, it might be that $h \notin \langle g\rangle$, or we just might have had bad luck. There are two ways to proceed. The first is to repeat the rho method several times until we either have found that $h \in \langle g \rangle$, or until we can conclude with a high enough probability that $h \notin \langle g \rangle$. The second way is to check for the c solutions of the equation $sa \equiv cz \pmod{n}$ whether $g^z = h$. This method works very well in practice. Notice that $h \in \langle g \rangle$ if and only if $g^z = h$ for one of those c values z. However, in the worst case where $\operatorname{ord} g = \operatorname{ord} h = p$ (p prime) and $\langle g \rangle \cap \langle h \rangle = \emptyset$, it may take p such checks!

Knowledge of an interval cannot be exploited to obtain a significant speed-up of the rho method (but there is the kangaroo method that takes care of this setting). It is not known how the Hamming-weight-t-DLP, the distribution-(p_i)-DLP or the $E(x)$-DLP could be solved any faster than the standard-DLP using the rho method. The knowledge of $x \bmod q$ can be used to decrease the running time by a factor of $\sqrt{\gcd(q, \operatorname{ord} g)}$, by working in the group generated by g^q (which gives no speed-up if $\operatorname{ord} g$ is prime). Also note that it is not possible to modify the method to obtain running time estimates in terms of the actual logarithm rather than in terms of $\operatorname{ord} g$.

A major advantage of the rho method over the baby-step giant-step method is that it is space efficient. It only needs a constant number of group elements to store, independently of the size of the group in which one works. Moreover, the rho method can be efficiently parallelized, where storage requirements on a central server are somewhat higher but can effectively be monitored.

4.3. **Parallelization.** Van Oorschot and Wiener [vOW99] have shown that the rho method can be parallelized with linear speed-up.

Assume we are given m processors. On each processor we use the same iterating function, initialize a sequence (y_k) by computing a random power of g and then proceed as in the serial rho method. However, we look for a match not among the terms of that single sequence (which would give only a speed-up of a factor of $\sqrt{m}$ [Bre90]), but among all terms computed on all m processors. This is done using the *distinguished-point method* which works as follows: One defines a set $\mathcal{D} \in \langle g \rangle$ that consists of all group elements that satisfy a certain distinguishing property. The elements in $\mathcal{D}$ are called distinguished point. A popular way of defining $\mathcal{D}$ is to fix an integer f and to define that $z \in \mathcal{D}$ if and only if the f least significant bits in the representation of z as a binary string are zero. Notice that this definition allows a fast check for the distinguishing property to hold, and the size of $\mathcal{D}$ can be easily monitored as well. Now, while computing a sequence (y_k) together with the corresponding exponent sequences (α_k) and (β_k), for each y_k the processor checks whether y_k is a distinguished point. If this is the case, it submits the triple (y_k, α_k, β_k) to a central server, and then goes on. The central server, which collects all distinguished points found by all m processors, searches for a match among those points. That can be done totally independent of the m processors.

Van Oorschot and Wiener have analyzed this method and showed that, under the assumption that the iterating function behaves like a random mapping, the expected running time on each processor is $\sqrt{\pi\,(\operatorname{ord} g)/2}\,/m + 1/\Theta$, where Θ is the proportion of distinguished points (i.e., $\Theta = 2^{-f}$ in the above example for $\mathcal{D}$). The central server has to store an expected number of $\Theta\sqrt{\pi\,(\operatorname{ord} g)/2}$ distinguished points.

In November 1997, the Mississauga company Certicom issued a challenge: a list of elliptic curve DLPs in elliptic curve groups of various sizes [Cer]. The DLPs in the smallest groups up to a prime group order of 108 bits have been solved by parallelized attacks distributed over the Internet. The extensive experimental work related to this challenge (see, for example, [ESST98]) confirms the theoretically predicted linear speed-up for the parallelized rho method.

4.3.1. *Inverse-point strategy.* In certain groups, a speed-up up to a factor of $\sqrt{2}$ can be achieved by applying the *inverse-point strategy.* In this method, each group element is paired with its inverse: The iterating function is defined such that if the group element a is mapped to the group element b, its inverse a^{-1} is mapped to b^{-1}. Further, the distinguished point set is defined such that if b is a distinguished point, then b^{-1} is also a distinguished point. Also, we require from the representation of the group elements that it can be detected if the inverse of a previously submitted point is submitted, and then such a collision also suits to reveal the discrete logarithm.

The inverse point strategy has been successfully applied in groups of elliptic curves over finite fields [ESST98] (see also [WZ98]).

A related technique has been applied in the special case of elliptic curves over $\mathbb{F}_{2^{nk}}$ defined over $\mathbb{F}_{2^n}$ [GLV00, WZ98]. Here, a speed-up up to a factor of $\sqrt{2k}$ is achieved by assembling the group elements in equivalence classes via the Frobenius map. Non-trivial automorphisms have also been exploited to speed up the rho method for discrete logarithm computation on certain hyperelliptic curves [DGM99].

4.4. **On the analysis of the running time.** The usual heuristic assumption in the analysis of the rho method is that the iterating function is a random mapping, and then the birthday paradoxon can be applied. However, in practice, one is restricted to a small set of mappings $F : G \to G$, namely those which can be evaluated efficiently and allow us to keep track of the exponent sequences (α_k) and (β_k).

The iterating functions that correspond to r-adding walks as defined in Section 4.1 seem to be a good choice despite the fact that they are not random mappings. Indeed, let $M_1, \ldots, M_r$ be randomly chosen elements in $\langle g \rangle$ and let $n = \operatorname{ord} g$ be prime. Then from Hildebrand's work [Hil94] on random walks on the integers modulo n, Teske [Tes00] concludes that if in the iteration $y_{k+1} = y_k * M_{\nu(y_k)}$ the $\nu(y_k)$ are randomly chosen from $\{1, \ldots, r\}$, r-adding walks are getting close to uniformly distributed on $\langle g \rangle$ after $O(n^{2/(r-1)})$ steps. Consequently, for $r \geq 6$ the number of steps in the rho method is expected to be $O(\sqrt{n})$, under the above randomness assumption on the $\nu(y_k)$. In practice, for $r \geq 16$ and with ν a hash function we find the same performance as expected from a random mapping, that is, cycles occur after an expected number of $1.25\sqrt{n}$ iterations ([Tes00]).

Horwitz and Venkatesan [HV] view r-adding walks as random walks on r-regular Cayley graphs over $\langle g \rangle$ that are generated by r random group elements $M_1, \ldots, M_r$. Then solving the DLP with the rho method can be related to finding non-trivial cycles in random Cayley graphs, and the rapid mixing property of such Cayley graphs with $O(\log n)$ generators, which are expander graphs, can be exploited to analyze the running time of the rho method. Indeed, for n prime, $r = const \cdot \log n$, and under the assumption that a t-wise independent hash function is used to navigate on the Cayley graph ($t = const \cdot \log^2 n$), the running time is shown to be $O((\log^c n)\sqrt{n})$ operations (c a small constant). Horwitz and Venkatesan also give methods to implement t-wise independent hash functions.

5. Pollard's Kangaroo Method

The kangaroo method has been invented by Pollard [Pol78] as a space efficient method to solve the interval-$[a,b]$-DLP. In its original version, it works with a tame kangaroo T with starting point $t_0 = g^b$ and a wild kangaroo W with starting point $w_0 = h$. In terms of the exponents of g, T starts at the upper end of the interval $[a,b[$, while W starts at an unknown spot x. Let $S = \{g^{s_1}, \ldots, g^{s_r}\}$ ($s_i > 0$ and of size $O(\sqrt{b-a})$, $r = O(\log(b-a))$) be a set of jumps, and let $\nu : G \to \{1, \ldots, r\}$ be a hash function. We think of the exponents s_i as travel distances. The tame kangaroo travels through the group following the path

$$t_{j+1} = t_j * g^{s_{\nu(t_j)}} , \qquad j \in \mathbb{N}_0 .$$

While computing the path (t_j), we keep track of the travelled distances $\delta_j(T)$, where $\delta_0(T) = 0$ and

$$\delta_{j+1}(T) = \delta_j(T) + s_{\nu(t_j)} , \qquad j \in \mathbb{N}_0 .$$

Hence, $t_j = g^{b+\delta_j(T)}$, for each $j \in \mathbb{N}_0$. After a certain number of jumps, the tame kangaroo stops and installs a trap at its final spot, say t_M. Then the wild kangaroo is set off, following the path

$$w_{j+1} = w_j * g^{s_{\nu(w_j)}} , \qquad j \in \mathbb{N}_0 ,$$

with the distances $\delta_j(W)$ given by $\delta_0(W) = 0$ and

$$\delta_{j+1}(W) = \delta_j(W) + s_{\nu(w_j)} , \qquad j \in \mathbb{N}_0 .$$

Then $w_j = h * g^{\delta_j(W)}$ ($j \in \mathbb{N}_0$). Hence, in terms of the exponents of g, at each stage we know the exact position of T, but we do not know the position of W since its starting spot x is unknown; this is why W is wild. After each of W's jumps it is checked whether W has fallen into the trap. If this is the case, say at w_N, the equation $t_M = w_N$ immediately yields a solution of $g^x = h$, namely $x = \delta_M(T) - \delta_N(W)$. Otherwise, after a certain number of jumps the wild kangaroo is halted, and a new wild kangaroo with starting point $w_0 = h * g^z$ and initial distance $\delta_0(W) = z$ (z small) is set off. If W falls into T's trap, this means that the paths of the two kangaroos must have met earlier during the travel, from which point on they have been identical. In this case, if we draw the two paths on a piece of paper starting at the bottom left and the bottom right, respectively, we obtain the Greek letter lambda, which explains why the method is also called the lambda method. Van Oorschot and Wiener [vOW99] have analyzed this method and found out that the running time is expected to be minimal if the mean value of the s_i is approximately $\sqrt{b-a}\,/2$, and if T makes approximately $0.7\sqrt{b-a}$ jumps before placing the trap. After about $2.7\sqrt{b-a}$ jumps of the wild kangaroo, W either must have fallen into the trap or it must have safely passed the trap and then should be halted. The former happens with with probability ≈ 0.75 after an expected number of $1.7\sqrt{b-a}$ jumps, while the latter is expected to happen about 0.33 times. Thus, the expected total running time amounts to $(0.7 + 0.33 \cdot 2.7 + 1.7)\sqrt{b-a} \approx 3.3\sqrt{b-a}$ group operations. The algorithm has to store only the set of jumps and the current positions of the two kangaroos. If one allows more storage, one can use the distinguished point method to obtain an algorithm that takes expected $2\sqrt{b-a}$ operations. That algorithm is a special case ($m=2$) of the parallelized version described next.

5.1. **Parallelization.** The distinguished point method that is used to parallelize the rho method can also be used for the kangaroo method [vOW99]. Assume there are m processors, m even. (The serial case corresponds to $m=2$, with two processors simulated on one machine.) On each processor, we have one kangaroo, tame or wild. Everywhere we use the same set of jumps S whose jump distances s_i are chosen such that their mean value, say β, satisfies

$$\beta = m\sqrt{b-a}\,/4 \ .$$

Let $\sigma \approx \frac{1}{m/2}\beta$ be an integer that indicates the spacing between members of the same herd. That is, $m/2$ tame and $m/2$ wild kangaroos $T_1,\ldots,T_{m/2}$ and $W_1,\ldots,W_{m/2}$ are set off at the group elements

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

one on each processor. (Observe that the tame kangaroos are set off near the middle of the interval rather than its end – this leads to lowest average running times.) The initial travel distances are set to $\delta_{0,\mathrm{tame}}(T_i) = \delta_{0,\mathrm{wild}}(W_i) = (i-1)\sigma$ and each kangaroo is provided with a tag indicating whether it is tame or wild. Then each kangaroo jumps on its processor, 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. The central server checks whether that point has already been previously submitted (in order to detect a collision between a tame and a wild kangaroo), and stores it if this is not the case. Under certain randomness assumptions (see Section 5.2), such a collision is expected to occur after $2\sqrt{b-a}\,/m$ jumps of each kangaroo. Thus, with Θ denoting the proportion of distinguished points in $\langle g\rangle$, the expected running time on each processor amounts to $2\sqrt{b-a}\,/m + 1/\Theta$ group operations.

It is possible in the above situation that kangaroos of the same herd collide. Such a collision is useless, because it does not reveal anything about the discrete logarithm. An analysis of this phenomenon [Tes01] shows that at most 2 useless collisions are expected to occur, independent of the number of processors.

Pollard [Pol00] has developed a version of parallelization where useless collisions *cannot* occur. Here 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 $u+v \le m$. The r jump distances in the set of jumps are multiples of uv, say $s_i = q_i uv$ for positive integers $q_1,\ldots,q_r$. Here, the mean value of the q_i should be close to $\beta = \sqrt{(b-a)/(uv)}\,/2$ for optimal results. The set-off points of the tame and wild kangaroos are $g^{(a+b)/2+iv}$ ($i = 0,\ldots,u-1$) and $h * g^{ju}$ ($j = 0,\ldots,v-1$), respectively. This implies that any two tame kangaroos (and also any two wild kangaroos) travel with travel distances that are in pairwise distinct residue classes modulo uv. As above, the actual travel is determined by a hash function $\nu : G \to \{1,\ldots,r\}$. 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, and this is the only pair that can collide. The expected running time in this case 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 expected running time as for the van Oorschot-Wiener parallelization.

In the above versions of the kangaroo method, proper choices for the jump distances s_i, respectively q_i in the Pollard parallelization, are (a) consecutive powers of two starting from $2^0 = 1$, or (b) integers randomly chosen from $[1, 2\beta]$ subject to the condition that they are pairwise distinct and have no non-trivial common divisor.

Experimental work with the (parallelized) kangaroo method shows that the theoretically predicted running times can be met. See [Tes01] for a corresponding report.

5.2. **On the running time analysis.** All expected running times given above have been derived under the assumption that the discrete logarithm is uniformly distributed over the interval $[a, b]$. For an arbitrary distribution, the expected running time can always be bounded by $3\sqrt{b-a}/m + 1/\Theta$ group operations.

Under the assumption that the hash function ν behaves like a random function $G \to \{1, \ldots, r\}$, Pollard [Pol00] shows that a collision between a wild and a tame kangaroo is expected to occur after about $2\sqrt{b-a}B_S$ iterations (in the serial version). Here, B_S is a constant that depends on the set of jump distances s_i and can be determined by solving a certain large set of equations. For the case that the s_i are the consecutive powers of two starting from 1, Pollard shows that B_S is close to one if $|S| \geq 6$. We cannot compute B_S for the case that the jump distances are randomly chosen integers from $[1, 2\beta]$. Thus, in this case, the running times are subject to the additional assumption that for each jump of a fixed wild and tame kangaroo in the same region (on the line through $[a, b]$), a collision occurs with probability $1/\beta$. This assumption is obviously fulfilled if each jump were with a distance randomly chosen from $[1, 2\beta]$.

5.3. **Degrees of knowledge.** The kangaroo method is not well suited be to solve the decision-DLP. Of course, a collision between a tame and a wild kangaroo can occur if and only if $h \in \langle g \rangle$. But if no collision has occured after a certain number of steps, this might be due to extremely bad luck or to $h \notin \langle g \rangle$. For example, if $h \in \langle g \rangle$, the probability that a collision has occured after $4\sqrt{b-a}$ steps is about 0.9989. (This probability has been derived under the aforementioned assumption that a collision occurs with probability $1/\beta$ in each pair of jumps, and that the discrete logarithm is uniformly distributed over $[a, b]$. See [Tes01] for details.)

We can solve the unknown-order-DLP with the kangaroo method: As soon as a collision between a tame and wild kangaroo has happened, a solution x for $g^x = h$ can be derived from the respective travel distances without knowing $\operatorname{ord} g$. If $b - a \leq \operatorname{ord} g$, with a probability at least $1 - 10^{-6}$ it even yields the least positive such value x.

The kangaroo method has been designed as a square-root algorithm to solve the interval-$[a, b]$-DLP. It is faster than the rho method if $b - a < \pi/8 \cdot \operatorname{ord} g$. Thus, if the discrete logarithm is only known to lie in an interval of length between $\pi/8 \cdot \operatorname{ord} g$ and $\operatorname{ord} g$, this additional knowledge cannot be exploited for a speed-up

Due to the reduction from the mod-q-DLP to the interval-$[a, b]$-DLP, the kangaroo method can be used to solve the mod-q-DLP in expected running time $2\sqrt{\operatorname{ord} g/q} + 1/\Theta$ group operations if $\operatorname{ord} g$ is known. For the same reasons as before, this only gives a speed-up if $q > 8/\pi$.

In the case of the distribution-(p_i)-DLP, we can speed-up the kangaroo method by modifying the set-off points for the tame kangaroos accordingly, which reduces the average running time to cover the initial distance between the tame and the wild herds. Here, lowest expected running times are achieved if the tame kangaroos are set off at the expected value for the discrete logarithm. If we are given a distribution of the discrete logarithms over the entire interval $[1, \operatorname{ord} g]$, instead of simply using the rho method we can use the kangaroo method with an interval $[a, b]$ that comprises a high percentage of the solutions. This can yield a speed-up if the distribution is not too wide-spread.

Finally, there is no algorithm known to date to solve the Hamming-weight-t-DLP with some kangaroo method. Since the same is the case for the rho method, this means that so far no space efficient, parallelizable algorithm for the Hamming-weight-t-DLP has been found.

References

[BJT97] J. Buchmann, M. J. Jacobson, Jr., and E. Teske. On some computational problems in finite abelian groups. *Mathematics of Computation*, 66:1663–1687, 1997.

[Bre80] R. P. Brent. An improved Monte Carlo factorization algorithm. *BIT*, 20:176–184, 1980.

[Bre90] R. P. Brent. Parallel algorithms for integer factorization. In J. H. Loxton, editor, *Number theory and cryptography*, volume 154 of *London Mathematical Society Lecture Note Series*, pages 26–37. Cambridge University Press, 1990.

[BT00] S. R. Blackburn and E. Teske. Baby-step giant-step algorithms for non-uniform distributions. In *Algorithmic Number Theory Seminar ANTS-IV*, volume 1838 of *Lecture Notes in Computer Science*, pages 153–168. Springer-Verlag, 2000.

[Cer] The Certicom ECC Challenge. On `http://www.certicom.com/research/ecc_challenge.html`.

[DGM99] I. Duursma, P. Gaudry, and F. Morain. Speeding up the discrete log computation on curves with automorphisms. In *Advances in Cryptology – ASIACRYPT '99*, volume 1716 of *Lecture Notes in Computer Science*, pages 103–121. Springer-Verlag, 1999.

[ESST98] A. E. Escott, J. C. Sager, A. P. L. Selkirk, and D. Tsapakidis. Attacking elliptic curve cryptosystems using the parallel Pollard rho method. CryptoBytes (The technical newsletter of RSA laboratories), 4 (no. 2), pp.15-19, 1998. Available at http://www.rsa.com/rsalabs/pubs/cryptobytes.

[GLV00] R. Gallant, R. Lambert, and S. Vanstone. Improving the parallelized Pollard lambda search on binary anomalous curves. *Mathematics of Computation*, 69:1699–1705, 2000.

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

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

[HV] J. Horwitz and R. Venkatesan. Random cayley graphs and the discrete log. Preprint, 2000.

[KS99] D. L. Kreher and D. R. Stinson. *Combinatorial Algorithms*. CRC Press, 1999.

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

[Odl00] A. M. Odlyzko. Discrete logarithms: The past and the future. *Designs, Codes, and Cryptography*, 19:129–145, 2000.

[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*, 24:106–110, 1978.

[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.

[Sha71] D. Shanks. Class number, a theory of factorization and genera. In *Proc. Symp. Pure Math. 20*, pages 415–440. AMS, Providence, R.I., 1971.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, 1997.

[SL84] C. P. Schnorr and H. W. Lenstra, Jr. A Monte Carlo factoring algorithm with linear storage. *Mathematics of Computation*, 43(167):289–311, 1984.

[SS85] J. Sattler and C. P. Schnorr. Generating random walks in groups. *Ann.-Univ.-Sci.-Budapest.-Sect.-Comput.*, 6:65–79, 1985.

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

[Ter00] D. C. Terr. A modification of Shanks' baby-step giant-step algorithm. *Mathematics of Computation*, 69:767–773, 2000.

[Tes98a] E. Teske. A space efficient algorithm for group structure computation. *Mathematics of Computation*, 67:1637–1663, 1998.

[Tes98b] E. Teske. Speeding up Pollard's rho method for computing discrete logarithms. In *Algorithmic Number Theory Seminar ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 541–554. Springer-Verlag, 1998.

[Tes00] E. Teske. On random walks for Pollard's rho method. *Mathematics of Computation*, posted on February 18, 2000, PII: S 0025-5718(00)01213-8 (to appear in print), 2000.

[Tes01] E. Teske. Computing discrete logarithms with the parallelized kangaroo method. Research Report CORR 01-01, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, 2001. 18 pages.

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

[WZ98] M. Wiener and R. Zuccherato. Faster attacks on elliptic curve cryptosystems. In *Proceedings of SAC – Workshop on Selected Areas in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*, pages 190 – 200. Springer, 1998.

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

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