THE OPEN BOOK SERIES 1

ANTS X
Proceedings of the Tenth
Algorithmic Number Theory Symposium

Two grumpy giants and a baby

Daniel J. Bernstein and Tanja Lange

msp

THE OPEN BOOK SERIES **1** (2013)
Tenth Algorithmic Number Theory Symposium
dx.doi.org/10.2140/obs.2013.1.87

Two grumpy giants and a baby

Daniel J. Bernstein and Tanja Lange

Pollard's rho algorithm, along with parallelized, vectorized, and negating variants, is the standard method to compute discrete logarithms in generic prime-order groups. This paper presents two reasons that Pollard's rho algorithm is farther from optimality than generally believed. First, "higher-degree local anticollisions" make the rho walk less random than the predictions made by the conventional Brent-Pollard heuristic. Second, even a truly random walk is suboptimal, because it suffers from "global anticollisions" that can at least partially be avoided. For example, after $(1.5+o(1))\sqrt{\ell}$ additions in a group of order ℓ (without fast negation), the baby-step-giant-step method has probability $0.5625+o(1)$ of finding a uniform random discrete logarithm; a truly random walk would have probability $0.6753\ldots+o(1)$; and this paper's new two-grumpy-giants-and-a-baby method has probability $0.71875+o(1)$.

1. Introduction

Fix a prime ℓ. The discrete-logarithm problem for a group G of order ℓ is the problem of finding $\log_g h$, given a generator g of G and an element h of G. The notation $\log_g h$ means the unique $s \in \mathbb{Z}/\ell$ such that $h = g^s$, where G is written multiplicatively.

The difficulty of finding discrete logarithms depends on G. For example, if G is the additive group $\mathbb{Z}/\ell$ (encoded as bit strings representing $\{0, 1, \ldots, \ell-1\}$ in the usual way), then $\log_g h$ is simply h/g, which can be computed in polynomial time using the extended Euclidean algorithm. As a more difficult example, consider the case that $p = 2\ell + 1$ is prime and G is the order-ℓ subgroup of the multiplicative group $\mathbb{F}_p^*$ (again encoded in the usual way); index-calculus attacks then run in time subexponential in p and thus in ℓ. However, if G is the order-ℓ subgroup of $\mathbb{F}_p^*$ where $p-1$ is a much larger multiple of ℓ, then index-calculus attacks become

MSC2010: 11Y16.
Keywords: Pollard rho, baby-step giant-step, discrete logarithms, complexity.

much slower in terms of ℓ; the standard algorithms are then the baby-step-giant-step method, using at most $(2+o(1))\sqrt{\ell}$ multiplications in G, and the rho method, which if tweaked carefully uses on average $(\sqrt{\pi/2}+o(1))\sqrt{\ell}$ multiplications in G.

This paper focuses on generic discrete-logarithm algorithms such as the baby-step-giant-step method and the rho method. "Generic" means that these algorithms work for any order-ℓ group G, using oracles to compute $1 \in G$ and to compute $a, b \mapsto ab$ for any $a, b \in G$. See Section 2 for a precise definition.

If G is an elliptic-curve group chosen according to standard criteria then the best discrete-logarithm algorithms available are variants of the baby-step-giant-step method and the rho method, taking advantage of the negligible cost of computing inverses in G. There is a standard "inverting" (or "negating") variant of the concept of a generic algorithm, also discussed in Section 2. This paper emphasizes the noninverting case, but all of the ideas can be adapted to the inverting case.

Measuring algorithm cost. The most fundamental metric for generic discrete-logarithm algorithms, and the metric used throughout this paper, is the probability of discovering a uniform random discrete logarithm within m multiplications. By appropriate integration over m one obtains the average number of multiplications to find a discrete logarithm, the variance, and so on. We caution the reader that comparing probabilities of two algorithms for one m can produce different results from comparing averages, maxima, and so forth; for example, the rho method is faster than baby-step-giant-step on average but much slower in the worst case.

One can interpret a uniform random discrete logarithm as $\log_g h$ for a uniform random pair (g, h), or as $\log_g h$ for a fixed g and a uniform random h. The following trivial "worst-case-to-average-case reduction" shows that a worst-case discrete logarithm is at most negligibly harder than a uniform random discrete logarithm: One computes $\log_g h$ as $\log_g h' - r$ where $h' = hg^r$ for a uniform random $r \in \mathbb{Z}/\ell$.

There are many reasons that simply counting multiplications, the number m above, does not adequately capture the cost of these algorithms:

- A multiplication count ignores overhead; that is, the costs of computations other than multiplications. For example, the ongoing ECC2K-130 computation uses a very restricted set of Frobenius powers, sacrificing approximately 2% in the number of multiplications, because this reduces the overhead enough to speed up the entire computation.
- A multiplication count ignores issues of memory usage. For some algorithms, such as the baby-step-giant-step method, memory usage grows with $\sqrt{\ell}$, while for others, such as the rho method, memory usage is constant (or near-constant).
- A multiplication count is blind to optimizations of the multiplication operation. The question here is not simply how fast multiplication can be, but how multiplication algorithms interact with higher-level choices in these algorithms.

For example, Cheon, Hong, and Kim in [10] showed how to look ahead one step in the rho method for $\mathbb{F}_p^*$ and combine two multiplications into one at the expense of very little overhead, although memory usage increases.

- A multiplication count ignores issues of parallelization. Pollard's original rho method is difficult to parallelize effectively, but "distinguished point" variants of the rho method are heavily parallelizable with little overhead.
- A multiplication count ignores issues of vectorization. Modern processors can operate on a vector of words in one clock cycle, but this requires that the operation be the same across the entire vector. This issue was raised in a recent discussion of whether the negation map on an elliptic curve can actually be used to speed up the rho method, rather than merely to save multiplications; see [6] and [3] for the two sides of the argument.

An improvement in multiplication counts does not necessarily indicate an improvement in more sophisticated cost metrics. It is nevertheless reasonable to begin with an analysis of multiplication counts, as is done in a large fraction of the literature; followup analyses can then ask whether improved multiplication counts are still achievable by algorithms optimized for other cost metrics.

Contents of this paper. Brent and Pollard in [7] identified a source of nonrandomness in the rho method, and quantified the loss of success probability produced by this nonrandomness, under plausible heuristic assumptions. The Brent-Pollard nonrandomness (with various simplifications and in various special cases) has been stated by many authors as the main deficiency in the rho method, and the rho method has been the workhorse of large-scale discrete-logarithm computations. There appears to be a widespread belief that, except for the Brent-Pollard nonrandomness, the rho method is the best conceivable generic discrete-logarithm algorithm. Of course, the rho method can take more than $2\sqrt{\ell}$ multiplications in the worst case while the baby-step-giant-step method is guaranteed to finish within $2\sqrt{\ell}$ multiplications, but the rho method is believed to be the best way to spend a significantly smaller number of multiplications.

This paper shows that there are actually at least two more steps separating the rho method from optimality. First, the rho method is actually less random and less successful than the Brent-Pollard prediction, because the rho method suffers from a tower of what we call "local anticollisions"; Brent and Pollard account only for "degree-1 local anticollisions". Second, and more importantly, the rho method would not be optimal even if it were perfectly random, because it continues to suffer from what we call "global anticollisions". We introduce a new "two grumpy giants and a baby" algorithm that avoids many of these global anticollisions.

This new algorithm, like the original baby-step-giant-step algorithm, has low overhead but high memory. We have not found a low-memory variant. This means that, for the moment, the algorithm is useful only for discrete-logarithm problems small enough to fit into fast memory. The algorithm nevertheless challenges the idea that the rho method is optimal for larger problems. The same approach might also be useful for "implicit" discrete-logarithm problems in which rho-type iteration is inapplicable, such as stage 2 of the $p-1$ factorization method, but those problems involve many overheads not considered in this paper.

Section 2 describes the general concept of anticollisions. Section 3 reviews the Brent-Pollard nonrandomness. Section 4 discusses higher-degree anticollisions in the rho method. Section 5 reports computations of optimal discrete-logarithm algorithms for small ℓ. Section 6 presents our new algorithm.

2. Anticollisions

This section introduces the concept of anticollisions in generic discrete-logarithm algorithms. This section begins by reviewing one of the standard ways to define such algorithms; readers familiar with the definition should still skim it to see our notation.

Generic discrete-logarithm algorithms. The standard way to formalize the idea that a generic algorithm works for any order-ℓ group G is to give the algorithm access to an oracle that computes $1 \in G$ and an oracle that computes the function $a, b \mapsto ab$ from $G \times G$ to G. The elements of G are encoded as a size-ℓ set $\overline{G}$ of strings.

An m-multiplication generic algorithm is one that calls the $a, b \mapsto ab$ oracle m times. The algorithm obtains 1 for free, and has g and h as inputs, so overall it sees $m+3$ group elements. We write $w_0 = 1$, $w_1 = g$, $w_2 = h$, and w_i for $i \geq 3$ as the $(i-2)$nd output of the $a, b \mapsto ab$ oracle: In other words, $w_i = w_j w_k$ for some $j, k \in \{0, 1, \ldots, i-1\}$ computed by the algorithm as functions of $w_0, w_1, \ldots, w_{i-1}$. These functions can also flip coins (that is, take as an additional input a sequence $b_0, b_1, \ldots$ of uniform random bits that are independent of each other, of g, of h, and so on.), but cannot make oracle calls.

The standard way to formalize the idea that a generic algorithm does not take advantage of the structure of G is to hide this structure by randomizing it. For example, one can take G as the additive group $\mathbb{Z}/\ell$, and take $\overline{G}$ as the usual binary representation of $\{0, 1, \ldots, \ell-1\}$, but choose a uniform random injection from G to $\overline{G}$ rather than the usual encoding. One defines the *generic* success probability of a generic algorithm by averaging not only over $\log_g h$ but also over the choices of this injection.

To allow inverting algorithms one also allows free access to an oracle that computes $a \mapsto 1/a$. Equivalently, one allows the algorithm to compute w_i as either $w_j w_k$ or w_j / w_k, and one also provides $1/w_i$. Of course, one can simulate this inversion oracle using approximately $\log_2 \ell$ calls to the multiplication oracle, since

$1/a = a^{\ell-1}$; an algorithm that uses only a small number of inversions can thus be simulated at negligible cost without inversions.

Slopes. Each w_i can be written as $h^{x_i} g^{y_i}$ for a pair $(x_i, y_i) \in (\mathbb{Z}/\ell)^2$ trivially computable by the algorithm. Specifically, $w_0 = 1 = h^{x_0} g^{y_0}$ where $(x_0, y_0) = (0, 0)$; $w_1 = g = h^{x_1} g^{y_1}$ where $(x_1, y_1) = (0, 1)$; $w_2 = h = h^{x_2} g^{y_2}$ where $(x_2, y_2) = (1, 0)$; if w_i is computed as $w_j w_k$ then $w_i = h^{x_i} g^{y_i}$ where $(x_i, y_i) = (x_j, y_j) + (x_k, y_k)$; and if an inverting algorithm computes w_i as w_j / w_k then $w_i = h^{x_i} g^{y_i}$ where $(x_i, y_i) = (x_j, y_j) - (x_k, y_k)$.

Normally these algorithms find $\log_g h$ by finding collisions in the map

$$(x, y) \mapsto h^x g^y$$

from $(\mathbb{Z}/\ell)^2$ to G. A collision $h^{x_i} g^{y_i} = h^{x_j} g^{y_j}$ with $(x_i, y_i) \neq (x_j, y_j)$ must have $x_i \neq x_j$ (otherwise $g^{y_i} = g^{y_j}$ so $y_i = y_j$ since g generates G), so the negative of the slope $(y_j - y_i)/(x_j - x_i)$ is exactly $\log_g h$. The discrete logarithms found by $w_0, w_1, \ldots, w_{m+2}$ are thus exactly the negatives of the $(m+3)(m+2)/2$ slopes (excluding any infinite slopes) between the $m+3$ points $(x_0, y_0), \ldots, (x_{m+2}, y_{m+2})$ in $(\mathbb{Z}/\ell)^2$. The number of discrete logarithms found in this way is the number d of distinct non-infinite slopes. The generic chance of encountering such a collision is exactly d/ℓ.

In the remaining cases, occurring with probability $1 - d/\ell$, these algorithms simply guess $\log_g h$. The success chance of this guess is 0 if the guess matches one of the negated slopes discussed above; otherwise the conditional success chance of this guess is $1/(\ell - d)$, so the success chance of this guess is $1/\ell$. The overall generic success chance of the algorithm is thus between d/ℓ and $(d+1)/\ell$, depending on the strategy for this final guess. In the extreme case $d = \ell$ this guess does not exist and the generic success chance is 1.

(Similar comments apply to inverting algorithms, but the bound on d is doubled, because there are twice as many opportunities to find $-\log_g h$. Specifically, comparing w_j to w_i finds the slope $(y_j - y_i)/(x_j - x_i)$, while comparing w_j to $1/w_i$ finds $(y_j + y_i)/(x_j + x_i)$.)

A similar model for generic discrete-logarithm algorithms was introduced by Shoup in [23], along with the bound $O(m^2/\ell)$ on the generic success probability of m-multiplication algorithms. Nechaev in [15] three years earlier had proven the collision-probability bound $O(m^2/\ell)$ in a weaker model, where algorithms are permitted only to remotely manipulate group elements without inspecting strings representing the group elements. Nechaev's model is equivalent to Shoup's model when one measures algorithm cost as the number of multiplications, but is more restrictive than Shoup's model in more sophisticated cost metrics; for example, Nechaev's model is unable to express the rho algorithm.

Chateauneuf, Ling, and Stinson in [9] introduced the idea of counting distinct slopes. They pointed out that the success probability of the baby-step-giant-step method is a factor $2+o(1)$ away from the obvious quantification of the Nechaev-Shoup bound: m multiplications allow only $m/2$ baby steps and $m/2$ giant steps (if m is even), producing $(m/2+2)(m/2+1) \approx m^2/4$ slopes, while one can imagine $m+3$ points in $(\mathbb{Z}/\ell)^2$ potentially having as many as $(m+3)(m+2)/2 \approx m^2/2$ distinct slopes.

Computer searches reported in [9, Section 3] found for each $\ell < 100$ a set of only marginally more than $\sqrt{2\ell}$ points with slopes covering $\mathbb{Z}/\ell$. However, these sets of points do not form addition chains, and as far as we can tell the shortest addition chains for all of the constructions in [9] are worse than the baby-step-giant-step method in the number of multiplications used. The cost model used in [9] allows $a, b \mapsto a^s b^t$ as a single oracle call for any (s,t); we view that cost model as excessively simplified, and are skeptical that algorithms optimized for that cost model will be of any use in practice.

Anticollisions. We use the word "anticollision" to refer to an appearance of a useless slope — a slope that cannot create a new collision because the same slope has appeared before. Formally, an anticollision is a pair (i, j) with $i > j$ such that either

- $x_i = x_j$ or
- $(y_j - y_i)/(x_j - x_i)$ equals $(y_{j'} - y_{i'})/(x_{j'} - x_{i'})$ for some pair (i', j') lexicographically smaller than (i, j) with $i' > j'$.

The number of anticollisions is exactly the gap $(m+3)(m+2)/2 - d$, where as above d is the number of distinct non-infinite slopes. Our objective in this paper is to understand *why* anticollisions occur in addition chains in $(\mathbb{Z}/\ell)^2$, and how these anticollisions can be avoided.

In Section 3 we review a standard heuristic by Brent and Pollard that can be viewed as identifying some anticollisions in the rho method, making the rho method somewhat less effective than a truly random walk would be. In Section 4 we identify a larger set of anticollisions in the rho method, making the rho method even less effective than predicted by Brent and Pollard. This difference is most noticeable for rho walks that use a very small number of steps, such as hardware-optimized walks or typical walks on equivalence classes modulo Frobenius on Koblitz curves.

It should be obvious that even a truly random walk produces a large number of anticollisions when m grows to the scale of $\sqrt{\ell}$. In Section 6 we show that at least a constant fraction of these anticollisions can be eliminated: We construct an explicit and efficient addition chain with significantly fewer anticollisions, and thus significantly higher success probability, than a truly random walk.

3. Review of the Brent-Pollard nonrandomness

This section reviews the nonrandomness that Brent and Pollard pointed out in the rho method. The literature contains three formulas for this nonrandomness, in three different levels of generality, backed by two different heuristic arguments. As discussed in Section 4, these heuristics account for "degree-1 local anticollisions" but do not account for "higher-degree local anticollisions".

The rho method. The rho method precomputes r distinct "steps"

$$s_1, s_2, \ldots, s_r \in G - \{1\}$$

(as some initial w's), and then moves from w_i to $w_{i+1} = w_i s_j$, where j is a function of w_i. Write p_j for the probability that step s_j is used.

We suppress standard details of efficient parallelization and collision detection here, since our emphasis is on the success probability achieved after m multiplications. Inserting each new group element into an appropriate data structure will immediately recognize the first collision without consuming any multiplications.

The $\sqrt{V}$ formula. Brent and Pollard in [7, Section 2] introduced the following heuristic argument, concluding that if the values $w_0, \ldots, w_m$ are distinct then w_{m+1} collides with one of those values with probability approximately mV/ℓ, where V is defined below. This implies that the total chance of a collision within m multiplications (that is, within $w_0, \ldots, w_{m+2}$) is approximately $1-(1-V/\ell)^{m^2/2}$, which in turn implies that the average number of multiplications for a collision is approximately $\sqrt{\pi/2}\sqrt{\ell}/\sqrt{V}$. For comparison, a truly random walk would have $V = 1$.

This argument applies to a more general form of the rho method, in which some function F is applied to w_i to produce w_{i+1}. The first collision might be unlucky enough to involve w_0, but otherwise it has the form $w_{i+1} = w_{j+1}$ with $w_i \neq w_j$, revealing a collision $F(w_i) = F(w_j)$ in the function F. Applications vary in how they construct F and in the use that they make of a collision.

Assume, heuristically, that the probability of w_i matching any particular value y is proportional to the number of preimages of y; in other words, assume that $\Pr[w_i = y] = \#F^{-1}(y)/\ell$, where $F^{-1}(y)$ means $\{x : F(x) = y\}$. This heuristic is obviously wrong for w_0, but this is a minor error in context; the heuristic seems plausible for $w_1, \ldots, w_m$, which are each generated as outputs of F.

Assume that $w_0, \ldots, w_m$ are distinct. Define X as the set of preimages of $w_1, \ldots, w_m$, so that X is the disjoint union of $F^{-1}(w_1), \ldots, F^{-1}(w_m)$. Then the expected size of X is

$$\sum_x \Pr[x \in X] = \sum_x \sum_i \Pr[F(x) = w_i] = \sum_x \sum_i \sum_y \Pr[F(x) = y \text{ and } w_i = y].$$

Assume, heuristically, that $F(x) = y$ and $w_i = y$ are independent events. Then

$$\begin{aligned}\sum_x \Pr[x \in X] &= \sum_i \sum_y \sum_x \Pr[F(x) = y] \Pr[w_i = y] \\ &= \sum_i \sum_y \#F^{-1}(y)^2/\ell \\ &= m \sum_y \#F^{-1}(y)^2/\ell.\end{aligned}$$

Define V as the variance over y of $\#F^{-1}(y)$. The average over y of $\#F^{-1}(y)$ is 1, so $V = \left(\sum_y \#F^{-1}(y)^2/\ell\right) - 1$, so the expected size of X is $mV + m$. There are m known elements $w_0, \ldots, w_{m-1}$ of X; the expected number of elements of X other than $w_0, \ldots, w_{m-1}$ is mV. By hypothesis w_m is none of $w_0, \ldots, w_{m-1}$; if w_m were uniformly distributed subject to this constraint then it would have probability $mV/(\ell - m) \approx mV/\ell$ of being in X and thus leading to a collision in the next step.

The $\sqrt{1 - \sum_i p_i^2}$ formula. As part of [1] we introduced the following streamlined heuristic argument, concluding that the collision probability for w_{m+1} is approximately $m(1 - \sum_i p_i^2)/\ell$. This implies that the average number of multiplications for a collision is approximately $\sqrt{\pi/2}\sqrt{\ell}/\sqrt{1 - \sum_i p_i^2}$.

Fix a group element v, and let w and w' be two independent uniform random elements. Consider the event that w and w' both map to v but $w \neq w'$. This event occurs if there are distinct i, j such that the following three conditions hold simultaneously:

- $v = s_i w = s_j w'$;
- s_i is chosen for w;
- s_j is chosen for w'.

These conditions have probability $1/\ell^2$, p_i, and p_j respectively. Summing over all (i, j) gives the overall probability

$$\Bigl(\sum_{i \neq j} p_i p_j\Bigr)/\ell^2 = \Bigl(\sum_{i,j} p_i p_j - \sum_i p_i^2\Bigr)/\ell^2 = \Bigl(1 - \sum_i p_i^2\Bigr)/\ell^2.$$

Hence the probability of an immediate collision from w and w' is $\left(1 - \sum_i p_i^2\right)/\ell$, where we added over the ℓ choices of v.

After $m + 3$ group elements one has approximately $m^2/2$ potentially colliding pairs. If the inputs to the iteration function were independent uniformly distributed random points then the probability of success would be $1 - \bigl(1 - \bigl(1 - \sum_i p_i^2\bigr)/\ell\bigr)^{m^2/2}$ and the average number of iterations before a collision would be approximately

$\sqrt{\pi/2}\sqrt{\ell}/\sqrt{1-\sum_i p_i^2}$. The inputs to the iteration function in Pollard's rho method are not actually independent, but this has no obvious effect on the average number of iterations.

Relating the two formulas. We originally obtained the formula $\sqrt{1-\sum_i p_i^2}$ by specializing and simplifying the Brent-Pollard $\sqrt{V}$ formula as follows.

The potential preimages of y are $y/s_1, y/s_2, \dots, y/s_r$, which are actual preimages with probabilities $p_1, p_2, \dots, p_r$ respectively. A subset I of $\{1, 2, \dots, r\}$ matches the set of indices of preimages with probability $\left(\prod_{i\in I} p_i\right)\left(\prod_{i\notin I}(1-p_i)\right)$, so the average of $\#F^{-1}(y)^2$ is

$$\sum_I \#I^2\Bigl(\prod_{i\in I} p_i\Bigr)\Bigl(\prod_{i\notin I}(1-p_i)\Bigr).$$

It is easy to see that most monomials (for example, $p_1p_2p_3$) have coefficient 0 in this sum; the only exceptions are linear monomials p_i, which have coefficient 1, and quadratic monomials p_ip_j with $i<j$, which have coefficient 2. The sum therefore equals

$$\sum_i p_i + 2\sum_{i,j:i<j} p_ip_j = \sum_i p_i + \Bigl(\sum_i p_i\Bigr)^2 - \sum_i p_i^2 = 2 - \sum_i p_i^2.$$

Hence $V = 1-\sum_i p_i^2$.

The $\sqrt{1-1/r}$ formula. In traditional "adding walks" (credited to Lenstra in [20, p. 66]; see also [21, p. 295] and [25]), each p_i is $1/r$, and $\sqrt{1-\sum_i p_i^2}$ is $\sqrt{1-1/r}$. This $\sqrt{1-1/r}$ formula first appeared in [25], with credit to the subsequent paper [4] by Blackburn and Murphy. The heuristic argument in [4] is the same as the Brent-Pollard argument.

Case study: Koblitz curves. The $\sqrt{1-\sum_i p_i^2}$ formula was first used to optimize walks on Koblitz curves. These walks map a curve point W to $W+\varphi^i(W)$, where φ is the Frobenius map and i is chosen as a function of the Hamming weight of the normal-basis representation of the x-coordinate of W. The Hamming weight is not uniformly distributed, and any reasonable function of the Hamming weight is also not uniformly distributed, so the $\sqrt{1-1/r}$ formula does not apply. Note that these are "multiplying walks" rather than "adding walks" (if $W = x_iH + y_iG$ then $W+\varphi^i(W) = s_ix_iH + s_iy_iG$ for certain constants $s_i \in (\mathbb{Z}/\ell)^*$), but the heuristics in this section are trivially adapted to this setting.

As a concrete example we repeat from [1] the analysis of our ongoing attack on ECC2K-130. All Hamming weights of x-coordinates of group elements are even, and experiments show that the distribution of even-weighted words of length 131 is close to the distribution of x-coordinates of group elements. Any iteration function defined in this way therefore inevitably introduces an extra factor to the running time of

$$1/\sqrt{1-\sum_i \binom{131}{2i}^2/2^{260}} \approx 1.053211,$$

even if all 66 weights use different scalars s_i. We extract just 3 bits of weight information, using only 8 different values for the scalars, to reduce the time per iteration. The values are determined by $\mathrm{HW}(x_{P_i})/2 \bmod 8$; the distribution of $\sum_i \binom{131}{16i+2j}$ for $0 \le j \le 7$ gives probabilities

$$0.1414,\ 0.1443,\ 0.1359,\ 0.1212,\ 0.1086,\ 0.1057,\ 0.1141,\ 0.1288,$$

giving a total increase of the number of iterations by a factor of 1.069993.

4. Higher-degree local anticollisions

Consider the rho method using r "steps" $s_1, s_2, \dots, s_r \in G$, as in the previous section. The method multiplies w_i by one of these steps to obtain w_{i+1}, multiplies w_{i+1} by one of these steps to obtain w_{i+2}, and so on.

Assume that the step w_{i+1}/w_i is different from the step w_{i+2}/w_{i+1}, but that w_{i+1}/w_i is the same as an earlier step w_{j+2}/w_{j+1}, and that w_{i+2}/w_{i+1} is the same as the step w_{j+1}/w_j. There are anticollisions $(i+1, j+2)$ and $(i+2, j+1)$, exactly the phenomenon discussed in the previous section: For example, w_{i+1} cannot equal w_{j+2} unless w_i equals w_{j+1}. There is, however, also a local anticollision $(i+2, j+2)$ not discussed in the previous section: w_{i+2} cannot equal w_{j+2} unless w_i equals w_j. The point is that the ratio w_{i+2}/w_i is a product of two steps, and the ratio w_{j+2}/w_j is a product of the same two steps in the opposite order.

We compute the heuristic impact of these "degree-2 local anticollisions", together with the degree-1 local anticollisions of Section 3, as follows. Assume for simplicity that 1, s_1, s_2, ..., s_r, s_1^2, $s_1 s_2$, ..., $s_1 s_r$, s_2^2, ..., $s_2 s_r$, ..., s_{r-1}^2, $s_{r-1} s_r$, s_r^2 are distinct. Write $F(w)$ for the group element that w maps to. Fix a group element v, and consider the event that two independent uniform random group elements w, w' have $F(F(w)) = v = F(F(w'))$ with no collisions among $w, w', F(w), F(w')$. This event occurs if there are i, i', j, j' with $s_j \ne s_{j'}$ and $s_j s_i \ne s_{j'} s_{i'}$ such that the following conditions hold simultaneously:

- $v = s_j s_i w = s_{j'} s_{i'} w'$;
- $F(w) = s_i w$;
- $F(s_i w) = s_j s_i w$;
- $F(w') = s_{i'} w'$;
- $F(s_{i'} w') = s_{j'} s_{i'} w'$.

These conditions have probability $1/\ell^2$, p_i, p_j, $p_{i'}$, and $p_{j'}$ respectively. Given the first condition, the remaining conditions are independent of each other, since $w = v/(s_j s_i)$, $s_i w = v/s_j$, $w' = v/(s_{j'} s_{i'})$, and $s_{i'} w' = v/s_{j'}$ are distinct. This event thus has probability $\sum p_i p_j p_{i'} p_{j'}/\ell^2$ where the sum is over all i, j, i', j' with $s_j \ne s_{j'}$ and $s_j s_i \ne s_{j'} s_{i'}$. The complement of the sum is over all i, j, i', j' with $s_j = s_{j'}$ or $s_j s_i = s_{j'} s_{i'}$ — that is, with $j = j'$ or with $i' = j \ne j' = i$. The complement is thus

$$\sum_j p_j^2 + \sum_{i,j:i\ne j} p_i^2 p_j^2 = \sum_j p_j^2 + \Bigl(\sum_j p_j^2\Bigr)^2 - \sum_j p_j^4,$$

and the original sum is $1 - \sum_j p_j^2 - \bigl(\sum_j p_j^2\bigr)^2 + \sum_j p_j^4$. Adding over all v gives probability $\bigl(1 - \sum_j p_j^2 - \bigl(\sum_j p_j^2\bigr)^2 + \sum_j p_j^4\bigr)/\ell$ of this type of two-step collision between w and w'.

For example, if $p_i = 1/r$ for all i, then the degree-1-and-2 nonrandomness factor is $1/\sqrt{1 - 1/r - 1/r^2 + 1/r^3}$, whereas the Brent-Pollard (degree-1) nonrandomness factor is $1/\sqrt{1 - 1/r}$. These factors are noticeably different if r is small.

***Beyond degree* 2.** More generally, a "degree-k local anticollision" $(i + k,\ j + k)$ occurs when the product of k successive steps w_{i+1}/w_i, w_{i+2}/w_{i+1}, ... matches the product of k successive steps w_{j+1}/w_j, w_{j+2}/w_{j+1}, ..., without a lower-degree local anticollision occurring. We define a "degree-(k, k') local anticollision" $(i + k,\ j + k')$ similarly.

Given the vector $(s_1, s_2, \ldots, s_r)$, one can straightforwardly compute the overall heuristic effect of local anticollisions of degree at most k, by summing the products $p_{i_1} \cdots p_{i_k} p_{i'_1} \cdots p_{i'_k}$ for which $1, s_{i_1}, s_{i'_1}, s_{i_1} s_{i_2}, s_{i'_1} s_{i'_2}, \ldots$ are distinct. Experiments indicate that the largest contribution is usually from the smallest degrees.

We emphasize that the results depend on the vector $(s_1, s_2, \ldots, s_r)$, because generic commutative-group equations such as $s_1 s_2 = s_2 s_1$ are not the only multiplicative dependencies among $s_1, s_2, \ldots, s_r$. One can check that $s_1, s_2, \ldots, s_r$ have no nongeneric multiplicative dependencies of small degree (and modify them to avoid such dependencies), but they always have medium-degree nongeneric multiplicative dependencies, including mixed-degree nongeneric multiplicative dependencies.

If $s_1, s_2, \ldots, s_r$ have only generic dependencies of degree at most k then the sum described above is expressible as a polynomial in the easily computed quantities $I_2 = \sum_j p_j^2$, $I_4 = \sum_j p_j^4$, and so forth, by a simple inclusion-exclusion argument. For example, the degree-1 nonrandomness factor is $1/\sqrt{1 - I_2}$, as in Section 3; the degree-≤ 2 nonrandomness factor is $1/\sqrt{1 - I_2 - I_2^2 + I_4}$, as explained above; the degree-≤ 3 nonrandomness factor is $1/\sqrt{1 - I_2 - I_2^2 + I_4 - 3I_2^3 + 7I_2I_4 - 4I_6}$; the degree-$\le 4$ nonrandomness factor is

$$1/\sqrt{1-I_2-I_2^2+I_4-3I_2^3+7I_2I_4-4I_6-13I_2^4+53I_2^2I_4-56I_2I_6-17I_4^2+33I_8};$$

and so on. In the uniform case these factors are

$$1/\sqrt{1-1/r},$$
$$1/\sqrt{1-1/r-1/r^2+1/r^3},$$
$$1/\sqrt{1-1/r-1/r^2-2/r^3+7/r^4-4/r^5},$$

and so on.

Case study: $r = 6$. Hildebrand showed in [13] that almost every r-adding walk (with $p_j = 1/r$) reaches a nearly uniform distribution in $\mathbb{Z}/\ell$ within $O(\ell^{2/(r-1)})$ steps; in particular, within $o(\sqrt{\ell})$ steps for $r \ge 6$. Implementors optimizing Pollard's rho method for hardware often want r to be as small as possible to minimize overhead (the storage required for precomputed steps and the cost of accessing that storage), and in light of Hildebrand's result can reasonably choose $r = 6$. This raises the question of how random a 6-adding walk is; perhaps it is better to take a larger value of r, increasing overhead but reducing nonrandomness.

For $r = 6$, with $p_1 = p_2 = p_3 = p_4 = p_5 = p_6 = 1/6$ and generic $s_1, \dots, s_6$, the heuristic nonrandomness factors are given (to 6 decimal places) in Table 1. These factors converge to approximately 1.129162 as the degree increases; see Appendix A. Evidently the Brent-Pollard heuristic captures *most* of the impact of local anticollisions for $r = 6$, but not all of the impact.

We tried 2^{32} experiments for $\ell = 1009$. Each experiment generated 6 uniform random steps $s_1, s_2, \dots, s_6$ (without enforcing distinctness, and without any constraints on higher-degree multiplicative dependencies), carried out a random walk using $s_1, s_2, \dots, s_6$ with equal probability, and stopped at the first collision. The average walk length was approximately 1.150076 times $\sqrt{\pi/2}\sqrt{\ell}$; note that this

Degree	Factor	Degree	Factor	Degree	Factor
1	1.095445	≤ 6	1.123767	≤ 11	1.126654
≤ 2	1.110984	≤ 7	1.124696	≤ 12	1.126926
≤ 3	1.117208	≤ 8	1.125383	≤ 13	1.127151
≤ 4	1.120473	≤ 9	1.125909	≤ 14	1.127341
≤ 5	1.122452	≤ 10	1.126322	≤ 15	1.127503

Table 1. Approximate values of heuristic nonrandomness factors for the case $r = 6$, with $p_1 = p_2 = p_3 = p_4 = p_5 = p_6 = 1/6$ and generic $s_1, \dots, s_6$.

ℓ	Factor	Experiments	ℓ	Factor	Experiments
1009	1.150076	2^{32}	100000007	1.131149	2^{32}
10007	1.147874	2^{32}	1000000007	1.130194	2^{32}
100003	1.141283	2^{32}	10000000019	1.129680	2^{32}
1000003	1.136122	2^{32}	100000000003	1.129395	2^{28}
10000019	1.132946	2^{32}	1000000000039	1.129326	2^{26}

Table 2. Observed average walk length until a collision, for a uniform random walk in $\mathbb{Z}/\ell$ using 6 uniform random adding steps. "Factor" is the observed average walk length divided by $\sqrt{\pi/2}\sqrt{\ell}$, rounded to 6 digits after the decimal point. "Experiments" is the number of experiments carried out for ℓ.

does not count the multiplications used to generate $s_1, s_2, \ldots, s_6$. We then tried several larger values of ℓ; the resulting nonrandomness factors are shown in Table 2. Our heuristics predict that these numbers will converge to approximately 1.129162 as $\ell \to \infty$, rather than 1.095445.

Note that for small ℓ there is a larger chance of low-degree dependencies among the steps s_i, so it is not a surprise that smaller values of ℓ have larger nonrandomness factors. We do not know whether a quantitative analysis of this phenomenon would predict the numbers shown in Table 2 for small ℓ, or whether other phenomena also play a role.

Case study: Koblitz curves, revisited. Consider again the ECC2K-130 walk introduced in [1]. Here $\ell = 680564733841876926932320129493409985129$.

For $0 \le j \le 7$ define φ as the Frobenius map on the ECC2K-130 curve, and define $s_j \in \mathbb{Z}/\ell$ as $1 + 196511074115861092422032515080945363956^{j+3}$. This walk moves from P to $P + \varphi^{j+3}(P) = s_j P$ if the Hamming weight of the x-coordinate of P is congruent to $2j$ modulo 16; this occurs with probability (almost exactly) $p_j = \sum_i \binom{131}{16i+2j}/2^{130}$.

The only small-degree multiplicative dependencies among $s_0, \ldots, s_7$ are generic commutative-group equations such as $s_1 s_2 = s_2 s_1$. We already reported this in [1, Section 2] to explain why the walk is highly unlikely to enter a short cycle. We point out here that this has a larger effect, namely minimizing small-degree anticollisions. We now analyze the impact of the small-degree anticollisions that remain, those that arise from the generic commutative-group equations.

For degree 1 the nonrandomness factor is $1/\sqrt{1 - I_2} \approx 1.069993$. For degree ≤ 2 the nonrandomness factor is $1/\sqrt{1 - I_2 - I_2^2 + I_4} \approx 1.078620$. For degree ≤ 3 it is $1/\sqrt{1 - I_2 - I_2^2 - 3I_2^3 + I_4 + 7I_2I_4 - 4I_6} \approx 1.081370$. For degree ≤ 4 it is ≈ 1.082550.

Case study: Mixed walks. The same type of analysis also applies to "mixed walks" combining noncommuting steps such as $w \mapsto ws_1$, $w \mapsto ws_2$, and $w \mapsto w^2$.

Degree	Factor	Degree	Factor	Degree	Factor
1	1.224745	≤ 5	1.285444	≤ 8	1.293067
≤ 2	1.248075	≤ 6	1.288605	≤ 9	1.294325
≤ 3	1.269973	≤ 7	1.291514	≤ 10	1.295107
≤ 4	1.277533				

Table 3. Approximate values of heuristic nonrandomness factors for the mixed walk $w \mapsto ws_1$, $w \mapsto ws_2$, and $w \mapsto w^2$, with generic s_1 and s_2 and equiprobable steps.

ℓ	Factor	Experiments	ℓ	Factor	Experiments
1009	1.292381	2^{41}	10000019	1.297130	2^{36}
10007	1.298240	2^{41}	100000007	1.297071	2^{32}
100003	1.297896	2^{40}	1000000007	1.297020	2^{32}
1000003	1.297360	2^{37}	10000000019	1.297018	2^{32}

Table 4. Observed average walk length until a collision, for a uniform random walk in $\mathbb{Z}/\ell$ using 2 uniform random adding steps and 1 doubling step. Columns have the same meaning as in Table 2.

A sequence of such steps maps w to a monomial such as $w^4 s_1 s_2^3$; we sum the products $p_{i_1} \cdots p_{i_k} p_{i'_1} \cdots p_{i'_k}$ for which the monomials corresponding to $()$, (i_1), (i'_1), (i_1, i_2), (i'_1, i'_2), and so on, are distinct. The heuristic nonrandomness factor for degree $\le k$ is the reciprocal of the square root of this sum.

For three equiprobable steps $w \mapsto ws_1$, $w \mapsto ws_2$, and $w \mapsto w^2$, with generic s_1 and s_2, the heuristic nonrandomness factors are given (to 6 decimal places) in Table 3.

We tried experiments analogous to the 6-adding experiments described above. Each experiment generated 2 uniform random group elements s_1, s_2, carried out a random walk using $w \mapsto ws_1$, $w \mapsto ws_2$, and $w \mapsto w^2$ starting from a uniform random group element, and stopped at the first collision. Table 4 shows the resulting average walk lengths for various values of ℓ. The dependence on ℓ is much smaller here than it was in Table 2. The numerical data seems consistent with the idea that the limit of the actual nonrandomness factors as $\ell \to \infty$ matches the limit of the degree-$\le k$ heuristic nonrandomness factors as $k \to \infty$: somewhere between 1.295 and 1.298, very far from the traditional degree-1 nonrandomness factor $\sqrt{3/2} \approx 1.224745$.

For comparison, Teske in [25, Table 5] reported using $1.776\sqrt{\ell}$ multiplications on average for 2000 experiments with the same type of walk. Teske's cycle-detection method cost a factor of approximately 1.13 in the number of multiplications, according to [25, Section 2.2], so $1.776\sqrt{\ell}$ corresponds to an observed nonrandomness factor of $1.776/(1.13\sqrt{\pi/2}) \approx 1.254$. This might seem noticeably different not just from 1.224745 but also from our 1.297. However, since the standard deviation of random-walk lengths is on the same scale as the average, it is statistically unremarkable to see differences of a few percent after only 2000 experiments.

Optimizing asymptotics. It is frequently stated that the rho method, like a truly random walk, finishes in $(\sqrt{\pi/2}+o(1))\sqrt{\ell}$ multiplications on average.

However, the experimental results by Sattler and Schnorr [20, p. 76] and by Teske [25] showed clearly that $\sqrt{\pi/2}+o(1)$ is not achieved by small values of r, and in particular by Pollard's original rho method. The Brent-Pollard nonrandomness, and in particular the $\sqrt{1-1/r}$ formula, indicates that $\sqrt{\pi/2}+o(1)$ is not achieved by any bounded r; one must have $1/r \in o(1)$, that is, $r \to \infty$ as $\ell \to \infty$. On the other hand, if r grows too quickly then the cost of setting up r steps is nonnegligible.

This analysis does not contradict $\sqrt{\pi/2}+o(1)$. However, it does indicate that some care is required in the algorithm details, and that $\sqrt{\pi/2}+o(1)$ can be replaced by $\sqrt{\pi/2}+O(\ell^{-1/4})$ but not by $\sqrt{\pi/2}+o(\ell^{-1/4})$.

To optimize the $o(1)$ one might try choosing steps that are particularly easy to compute. For example, one might take $s_3 = s_1 s_2$, $s_4 = s_2 s_3$, and so on, where s_1, s_2 are random. We point out, however, that such choices are particularly prone to higher-degree anticollisions. We recommend taking into account not just the number of steps and the number of multiplications required to precompute those steps, but also the impact of higher-degree anticollisions.

5. Searching for better chains for small primes

If ℓ is small then by simply enumerating addition chains one can find generic discrete-logarithm algorithms that use fewer multiplications than the rho method.

This section reports, for each small prime ℓ, the results of two different computer searches. One search greedily obtained as many slopes as it could after each multiplication, deferring anticollisions as long as possible. The other search minimized the number of multiplications required to find an average slope. Chains found by such searches are directly usable in discrete-logarithm computations for these values of ℓ; perhaps they also provide some indication of what one can hope to achieve for much larger values of ℓ. These searches also show that merely counting the size of a slope cover, as in [9, Section 3], underestimates the cost of discrete-logarithm algorithms, although one can hope that the gap becomes negligible as ℓ increases.

A continuing theme in this section is that the obvious quantification of the Nechaev-Shoup bound is not tight. The bound says that an m-addition chain has $\leq(m+3)(m+2)/2$ slopes; but there is actually a gap, increasing with m, between $(m+3)(m+2)/2$ and the maximum number of slopes in an m-addition chain.

This section explains part of this gap by identifying two types of anticollisions that addition chains cannot avoid and stating an improved bound that accounts for these anticollisions. However, the improved bound is still not tight for most of these values of ℓ, and for long chains the improved bound is only negligibly stronger than the Nechaev-Shoup bound.

Greedy slopes. Define d_i as the number of distinct finite slopes among the points $(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)$ in $(\mathbb{Z}/\ell)^2$. For example, the chain

$$(0,0), (0,1), (1,0), (0,2), (1,2), (1,4)$$

in $(\mathbb{Z}/7)^2$ has $(d_0, d_1, d_2, d_3, d_4, d_5) = (0, 0, 2, 3, 5, 7)$: There are 2 distinct finite slopes among $(0,0)$, $(0,1)$, $(1,0)$; 3 distinct finite slopes among $(0,0)$, $(0,1)$, $(1,0)$, $(0,2)$; 5 distinct finite slopes among $(0,0)$, $(0,1)$, $(1,0)$, $(0,2)$, $(1,2)$; and 7 distinct finite slopes among $(0,0)$, $(0,1)$, $(1,0)$, $(0,2)$, $(1,2)$, $(1,4)$.

For each prime $\ell < 128$ we computed the lexicographically maximum sequence $(d_0, d_1, \ldots)$ for all infinite addition chains starting $(0,0), (0,1), (1,0)$ in $(\mathbb{Z}/\ell)^2$. These maxima, truncated to the first occurrence of ℓ, are displayed in Table 5. For example, Table 5 lists $(0, 0, 2, 3, 5, 7)$ for $\ell = 7$, indicating that the lexicographic maximum is $(0, 0, 2, 3, 5, 7, 7, 7, 7, 7, \ldots)$: One always has $d_0 = 0$, $d_1 = 0$, and $d_2 = 2$; the maximum possible d_3 is 3; given $d_3 = 3$, the maximum possible d_4 is 5; given $d_3 = 3$ and $d_4 = 5$, the maximum possible d_5 is 7.

This computation was not quite instantaneous, because it naturally ended up computing all finite chains achieving the truncated maximum (and, along the way, all chains achieving every prefix of the truncated maximum). There are, for example, 5420 length-21 chains that match the $(d_0, d_1, \ldots)$ shown in Table 5 for $\ell = 109$.

Minimal weight. We also computed ℓ-slope addition chains of minimal weight for each prime $\ell < 48$. Here "weight" means $\sum_{i \geq 1} i\,(d_i - d_{i-1})$. Dividing this weight by ℓ produces the average, over all $s \in \mathbb{Z}/\ell$, of the number of multiplications (plus 2 to account for the inputs g and h) used to find slope s. It might make more sense to compute $(\ell - 1)$-slope addition chains of minimal weight, since a generic discrete-logarithm algorithm that finds $\ell - 1$ slopes also recognizes the remaining slope by exclusion, but the gap becomes negligible as ℓ increases.

Lexicographically maximizing $(d_0, d_1, \ldots)$, as in Table 5, does not always produce minimal-weight ℓ-slope addition chains. For example, the chain

$$(0,0), (0,1), (1,0), (0,2), (0,3), (1,3), (1,6), (2,12), (2,14), (2,16), (3,17), (4,28)$$

for $\ell = 29$ has weight 210 with $(d_0, d_1, \ldots) = (0, 0, 2, 3, 4, 7, 10, 14, 19, 23, 27, 29)$, while chains achieving the lexicographic maximum in Table 5 have weight 211. We similarly found weight 299 (compared to 300) for $\ell = 37$, weight 372 (compared

ℓ	Weight	$d_0\, d_1\, \ldots$
2	4	0 0 2
3	7	0 0 2 3
5	15	0 0 2 3 5
7	25	0 0 2 3 5 7
11	50	0 0 2 3 5 7 10 11
13	64	0 0 2 3 5 7 10 13
17	96	0 0 2 3 5 7 10 14 16 17
19	113	0 0 2 3 5 7 10 14 17 19
23	148	0 0 2 3 5 7 10 14 19 22 23
29	211	0 0 2 3 5 7 10 14 19 23 26 28 29
31	230	0 0 2 3 5 7 10 14 19 23 28 31
37	300	0 0 2 3 5 7 10 14 19 23 29 33 36 37
41	347	0 0 2 3 5 7 10 14 19 24 29 34 39 41
43	375	0 0 2 3 5 7 10 14 19 24 29 34 38 42 43
47	425	0 0 2 3 5 7 10 14 19 24 30 35 40 44 47
53	510	0 0 2 3 5 7 10 14 19 24 30 36 41 45 50 52 53
59	596	0 0 2 3 5 7 10 14 19 24 30 36 42 48 52 57 58 59
61	631	0 0 2 3 5 7 10 14 19 24 30 35 42 48 52 56 59 61
67	727	0 0 2 3 5 7 10 14 19 24 30 36 41 47 53 59 63 66 67
71	788	0 0 2 3 5 7 10 14 19 24 30 36 42 48 54 60 66 70 71
73	815	0 0 2 3 5 7 10 14 19 24 30 36 43 50 56 62 67 71 73
79	919	0 0 2 3 5 7 10 14 19 24 30 37 43 49 57 64 69 73 76 79
83	978	0 0 2 3 5 7 10 14 19 24 30 37 44 51 59 65 72 77 80 83
89	1081	0 0 2 3 5 7 10 14 19 24 30 37 44 53 60 66 74 80 84 87 89
97	1224	0 0 2 3 5 7 10 14 19 24 30 37 44 51 61 69 78 83 88 92 96 97
101	1307	0 0 2 3 5 7 10 14 19 24 30 37 45 53 60 69 76 82 89 93 97 100 101
103	1351	0 0 2 3 5 7 10 14 19 24 30 37 45 52 60 67 74 83 89 94 98 102 103
107	1422	0 0 2 3 5 7 10 14 19 24 30 37 45 53 61 70 77 84 91 96 100 104 107
109	1466	0 0 2 3 5 7 10 14 19 24 30 37 44 52 60 68 77 84 91 98 102 106 108 109
113	1536	0 0 2 3 5 7 10 14 19 24 30 37 44 52 62 70 78 86 94 99 105 109 113
127	1806	0 0 2 3 5 7 10 14 19 24 30 37 45 53 63 73 84 92 98 105 112 118 122 126 127

Table 5. For each $\ell < 128$, the lexicographically maximum $(d_0, d_1, \ldots)$. "Weight" means $\sum_{i\geq 1} i\,(d_i - d_{i-1})$.

to 375) for $\ell = 43$, and weight 423 (compared to 425) for $\ell = 47$. It is not clear whether this gap becomes negligible as ℓ increases.

Some obstructions. We explain here two simple ways that anticollisions appear in addition chains. Every addition chain produces at least a linear number of anticollisions that follow these simple patterns.

First, doubling a point (x_j, y_j) produces two anticollisions: The slopes from $2(x_j, y_j)$ to (x_j, y_j) and to $(0, 0)$ are the same as the slope from (x_j, y_j) to $(0, 0)$. Doubling another point (x_k, y_k) produces three anticollisions: The slope from $2(x_k, y_k)$ to $2(x_j, y_j)$ is the same as the slope from (x_k, y_k) to (x_j, y_j). A third doubling produces four anticollisions, and so on; doubling n points produces a total of $n(n+3)/2$ anticollisions of this type.

Second, adding (x_i, y_i) to a distinct point (x_j, y_j) produces two anticollisions: The slopes from $(x_i, y_i) + (x_j, y_j)$ to (x_i, y_i) and to (x_j, y_j) are the same as the slopes from (x_j, y_j) and from (x_i, y_i) to $(0, 0)$. Subsequently adding the same (x_i, y_i) to another point (x_k, y_k) produces three anticollisions: The slope from $(x_i, y_i) + (x_k, y_k)$ to $(x_i, y_i) + (x_j, y_j)$ is the same as the slope from (x_k, y_k) to (x_j, y_j), exactly as in Section 3.

Applying these principles easily explains the initial pattern $0, 0, 2, 3, 5, 7$ that appears in Table 5. The first addition (whether or not a doubling) must produce at least two anticollisions, and therefore produces at most one new slope to the previous three points; this explains the 3. The second addition also produces at least two anticollisions, and therefore at most two new slopes to the previous four points; this explains the 5. One might think that the next step is 8, but having only two anticollisions in each of the first three additions would imply that those three additions include at most one doubling and no other reuse of summands, for a total of at least five summands, while there are only four nonzero summands available for the first three additions.

More generally, a chain of $m \ge 2$ nontrivial additions involves $2m$ inputs selected from $m+1$ nonzero points, so there must be at least $m-1$ repetitions of inputs. These repetitions produce at least $m-2$ occurrences of three anticollisions (one doubling is free), on top of m occurrences of two anticollisions and one anticollision for the infinite slope from $(0, 0)$ to $(0, 1)$, for a total of at least $3m-1$ anticollisions, and thus a total of at most $(m+3)(m+2)/2 - (3m-1) = (m^2 - m + 8)/2$ slopes. This explains $5, 7, 10, 14, 19$ in Table 5 but does not explain 24.

6. Two grumpy giants and a baby

This section presents the algorithm featured in the title of this paper. This algorithm is, as the name suggests, a modification to the standard baby-step-giant-step method. The modification increases the number of different slopes produced within m multiplications, and for a typical range of m increases the number beyond the effectiveness of the rho method.

In the baby-step-giant-step algorithm the baby steps compute $h^{x_i} g^{y_i}$ for $(x_i, y_i) \in (0, 0) + \{0, 1, 2, \dots, \lceil\sqrt{\ell}\rceil\}(0, 1)$ and the giant steps compute $h^{x_i} g^{y_i}$ for $(x_i, y_i) \in (1, 0) + \{0, 1, 2, \dots, \lfloor\sqrt{\ell}\rfloor\}(0, \lceil\sqrt{\ell}\rceil)$. The first observation is that the slopes within one type of step are constant; the second observation is that once all steps are done all ℓ slopes appear. Our idea is to make the lines of fixed slope shorter; that is, we introduce more players. Note that introducing a second baby is not useful: Lines between the points in $(x, y) + \{0, 1, 2, \dots, \lceil\sqrt{\ell}\rceil\}(0, 1)$ and $(0, 0) + \{0, 1, 2, \dots, \lceil\sqrt{\ell}\rceil\}(0, 1)$ repeat each slope $\approx \sqrt{\ell}$ times. We thus need to introduce more giants to make progress.

The two-grumpy-giants-and-a-baby method is parametrized by a positive integer n, normally proportional to $\sqrt{\ell}$; the reader should imagine n being approximately $0.5\sqrt{\ell}$. The number of multiplications in the method is approximately $3n$. Here is the set of points $(x_i, y_i) \in (\mathbb{Z}/\ell)^2$ produced by the method:

$$\begin{aligned}\text{Baby}: &\quad (0,0) + \{0, \dots, n-1\}(0,1)\\ \text{Giant1}: &\quad (1,0) + \{1, \dots, n\}(0,n)\\ \text{Giant2}: &\quad (2,0) - \{1, \dots, n\}(0,n+1)\end{aligned}$$

The initial negation $(0, -(n+1))$ for Giant2 has negligible cost, approximately $\log_2 \ell$ multiplications. Choosing n and $n+1$ for the steps in the y direction for the two giants gives a good coverage of slopes since n and $n+1$ are coprime. The grumpy giants make big steps (on the scale of $\sqrt{\ell}$) and quickly walk in opposite directions away from each other. Luckily they are not minding the baby.

We now analyze the slopes covered by this method. Again it is not interesting to look at the slopes among one type of points. The slope between a point $(0, i)$ in the Baby set and a point $(1, jn)$ in the Giant1 set is $jn - i$; this means that all slopes in $\{1, \dots, n^2\}$ are covered. The slope between $(0, i)$ in the Baby set and $(2, -j(n+1))$ in the Giant2 set is $(-j(n+1) - i)/2 \in \{-n^2 - 2n + 1, \dots, -n-1\}/2$; there are n^2 distinct slopes here, almost exactly covering $\{-n^2 - 2n + 1, \dots, -n-1\}/2$. The slope between $(1, in)$ in the Giant1 set and $(2, -j(n+1))$ in the Giant2 set is $-j(n+1) - in \in \{-2n^2 - n, \dots, -2n-1\}$; there are another n^2 distinct slopes here, covering about half the elements of $\{-2n^2 - n, \dots, -2n-1\}$.

To summarize, there are three sets of n^2 distinct slopes here, all between $-2n^2 - n + 1$ and n^2. One can hope for a total of $3n^2$ distinct slopes if $\ell > 3n^2 + n$, but this hope runs into two obstacles. The first obstacle is that the "odd" elements of $\{-n^2 - 2n + 1, \dots, -n-1\}$ can bump into the other sets when computing $(2i+1)/2 = i + (\ell+1)/2$; but for $\ell \in 4n^2 + O(n)$ this effect loses only $O(n)$ elements. The second obstacle is that any Giant1–Giant2 slopes between $(-n^2 - 2n)/2$ and $(-n-2)/2$ will bump into $\{-n^2 - 2n + 1, \dots, -n-1\}/2$ for the "even" elements of $\{-n^2 - 2n + 1, \dots, -n-1\}$. This is approximately the rightmost $1/4$ of the Giant1–Giant2 interval, but only $n^2/8 + O(n)$ of the Giant1–Giant2 slopes are in this interval. Overall there are $23n^2/8 + O(n)$ distinct slopes, that is, $(0.71875 + o(1))\ell$ distinct slopes.

For comparison, the same $(3+o(1))n$ multiplications allow the original baby-step-giant-step method to compute $(1.5+o(1))n$ baby steps and $(1.5+o(1))n$ giant steps, producing only $(2.25+o(1))n^2 = (0.5625+o(1))\ell$ distinct slopes. The same number of multiplications in the rho method (with $r \in 1/o(1)$ different steps, simulating a uniform random walk within a factor $1+o(1)$) produces $(9+o(1))n^2/2 = (1.125+o(1))\ell$ random slopes, and thus $(1-\exp(1.125)+o(1))\ell = (0.6753\ldots+o(1))\ell$ distinct slopes with overwhelming probability. We have performed computer experiments to check each of these numbers.

Weighing the giants. We repeat a warning from Section 1: One algorithm can be better than another after a particular number of multiplications but nevertheless have worse average-case performance.

For example, the baby-step-giant-step method has two standard variants, which we call the baby-steps-then-giant-steps method (introduced by Shanks in [22, pages 419–420]) and the interleaved-baby-step-giant-step method (introduced much later by Pollard in [17, p. 439, top]). Both variants (with giant steps chosen to be of size $(1+o(1))\sqrt{\ell}$) reach 100% success probability using $(2+o(1))\sqrt{\ell}$ multiplications, while the rho method has a lower success probability for that number of multiplications. Average-case performance tells a quite different story: The baby-steps-then-giant-steps method uses $(1.5+o(1))\sqrt{\ell}$ multiplications on average; the interleaved-baby-step-giant-step method is better, using $(4/3+o(1))\sqrt{\ell} = (1.3333\ldots+o(1))\sqrt{\ell}$ multiplications on average; the rho method (again with $1/r \in o(1)$) is best, using $(\sqrt{\pi/2}+o(1))\sqrt{\ell} = (1.2533\ldots+o(1))\sqrt{\ell}$ multiplications on average.

Our analysis above shows that the two-grumpy-giants-and-a-baby method is more effective than the rho method (and the baby-step-giant-step method) as a way to use $(1.5+o(1))\sqrt{\ell}$ multiplications. One might nevertheless guess that the rho method has better average-case performance; for example, an anonymous referee stated that the new method "presumably has worse average-case running time".

Our computer experiments indicate that the (interleaved-)two-grumpy-giants-and-a-baby method actually has better average-case running time than the rho method. For example, for $\ell = 65537$, we found a chain of weight $20644183 = (1.23046\ldots)\ell^{1.5}$ with the two-grumpy-giants-and-a-baby method. Here we chose $n = 146$, used (suboptimal) binary addition chains for $(0, n)$ and $(0, \ell-n-1)$, and then cycled between points $(0, i)$ and $(1, in)$ and $(2, -i(n+1))$ until we had ℓ different slopes. For $\ell = 1000003$ we found a chain of weight $1205458963 = (1.20545\ldots)\ell^{1.5}$ in the same way with $n = 558$.

Variants. We have been exploring many variants of this algorithm. We have found experimentally that a 4-giants algorithm (two in one direction, two in the other, with computer-optimized shifts of the initial positions) outperforms this 2-giants algorithm for $m \approx \sqrt{\ell}$. We speculate that gradually increasing the number of giants will produce an algorithm with $(0.5 + o(1))m^2$ distinct slopes, the best possible result (automatically also optimizing the average number of multiplications, the maximum, and so on), but it is not clear how to choose the shift distances properly.

Acknowledgments

This work was supported by the National Science Foundation under grants 0716498 and 1018836 and by the European Commission under Contract ICT-2007-216676 ECRYPT II. Computations were carried out on the LISA cluster at the SARA supercomputer center, supported by NCF grant MP-230-11. We thank the anonymous referees for several useful comments and questions. No babies (or giants) were harmed in the preparation of this paper.

Appendix A. Computing limits of anticollision factors

This appendix shows, for each integer $r > 3$, a reasonably fast method to compute the limit of the sequence of generic uniform heuristic nonrandomness factors

$$\begin{aligned}
&1/\sqrt{1-1/r},\\
&1/\sqrt{1-1/r-1/r^2+1/r^3},\\
&1/\sqrt{1-1/r-1/r^2-2/r^3+7/r^4-4/r^5},\\
&\ldots
\end{aligned}$$

considered in Section 4. For example, these factors converge to approximately 1.129162 for $r = 6$.

We are indebted to Neil Sloane's Online Encyclopedia of Integer Sequences [24] for leading us to [5] (by a search for the integer 4229523740916 shown below), and to Armin Straub for explaining how to use [2] and [18] to compute the sum $\sum_k u_k/r^{2k}$ discussed here. Our contribution here is the connection described below between anticollision factors and sums of squares of multinomials.

Review of sums of squares of multinomials. Define $U = \sum_i \sum_j s_i/s_j$ in the r-variable function field $\mathbb{Q}(s_1, \ldots, s_r)$, and define u_k as the constant coefficient of U^k. Consider the problem of computing $\sum_{k\ge 0} u_k/r^{2k}$.

Note that $U^k = \sum_{i_1,\ldots,i_k} \sum_{j_1,\ldots,j_k} s_{i_1}\cdots s_{i_k}/s_{j_1}\cdots s_{j_k}$, so u_k is the number of tuples $(i_1, \ldots, i_k, j_1, \ldots, j_k)$ such that $s_{i_1}\cdots s_{i_k}/s_{j_1}\cdots s_{j_k} = 1$; that is, such that $(i_1, \ldots, i_k)$ is a permutation of $(j_1, \ldots, j_k)$. The tuples counted here were named "abelian squares" by Erdős in 1961, according to [19]; u_k here is "$f_r(k)$" in the notation of [19].

For example, $u_0 = 1$; $u_1 = r$; and $u_2 = 2r^2 - r$, which one can partition into counting $2r^2 - 2r$ tuples (i_1, i_2, j_1, j_2) with $i_1 \neq i_2$ and $\{i_1, i_2\} = \{j_1, j_2\}$, and r tuples with $i_1 = i_2 = j_1 = j_2$. More generally, the number of ways for $s_{i_1} \cdots s_{i_k}$ to equal $s_1^{a_1} \cdots s_r^{a_r}$ is the multinomial coefficient $\binom{k}{a_1, a_2, \ldots, a_r}$, so

$$\begin{aligned} u_k &= \sum_{\substack{a_1, a_2, \ldots, a_r: \\ a_1 + a_2 + \cdots + a_r = k}} \binom{k}{a_1, a_2, \ldots, a_r}^2 \\ &= \sum_{m \geq 0} \binom{r}{m} \sum_{\substack{a_1, a_2, \ldots, a_m: \\ a_1 + a_2 + \cdots + a_m = k, \\ a_1 > 0, a_2 > 0, \ldots, a_m > 0}} \binom{k}{a_1, a_2, \ldots, a_m}^2. \end{aligned}$$

Richmond and Rousseau, proving a conjecture of Ruehr, showed in [18] that u_k is asymptotically $r^{2k+r/2}/(4\pi k)^{(r-1)/2}$ as $k \to \infty$. See also [19, Theorem 4] for another proof. We conclude that $\sum_k u_k/r^{2k}$ converges for $r > 3$ (and not for $r = 3$). For example, with $r = 6$, the ratio u_k/r^{2k} is asymptotically $6^3/(4\pi k)^{2.5}$, so $\sum_k u_k/r^{2k}$ converges, and the tail $\sum_{k>n} u_k/r^{2k}$ is $\Theta(1/n^{1.5})$.

This Θ is not an explicit bound; [18] and [19] are not stated constructively. However, inspecting examples strongly suggests that $(u_k/r^{2k})/(r^{r/2}/(4\pi k)^{(r-1)/2})$ converges upwards to 1 as $k \to \infty$, so it seems reasonably safe to hypothesize that u_k/r^{2k} is at most $2r^{r/2}/(4\pi k)^{(r-1)/2}$. This hypothesis implies that

$$\begin{aligned} \sum_{k>n} \frac{u_k}{r^{2k}} &\leq \sum_{k>n} \frac{2r^{r/2}}{(4\pi k)^{(r-1)/2}} \\ &< \int_n^\infty \frac{2r^{r/2}}{(4\pi k)^{(r-1)/2}} \, dk \\ &= \frac{4r^{r/2}}{(4\pi)^{(r-1)/2}(r-3)n^{(r-3)/2}}, \end{aligned}$$

so to compute tight bounds on $\sum_k u_k/r^{2k}$ it suffices to compute $\sum_{0 \leq k \leq n} u_k/r^{2k}$ for a moderately large integer n.

One can easily use the multinomial formula above to compute, for example, that $u_{10} = 4229523740916$ for $r = 6$, but if k and r are not very small then it is much more efficient to compute u_k from the generating function $\sum_k u_k x^k/k!^2 = \left(\sum_k x^k/k!^2\right)^r$ in the power-series ring $\mathbb{Q}[[x]]$. Barrucand in [2] pointed out this formula for u_k and explained how to use it to compute a recurrence for u_k. For $r = 6$ we simply computed the 6th power of $\sum_k x^k/k!^2$ in $\mathbb{Q}[x]/x^{5001}$, obtaining the exact values of u_k for $0 \leq k \leq 5000$ and concluding that $\sum_{0 \leq k \leq 5000} u_k/6^{2k} \approx 1.275007093$. This computation was fast enough that we did not bother to explore optimizations such as computing $\left(\sum_k x^k/k!^2\right)^r$ modulo various small primes or analyzing the numerical stability of Barrucand's recurrence.

Anticollision factors via sums of squares of multinomials. Define h_k as the number of tuples $(i_1, i_2, \ldots, i_k, j_1, j_2, \ldots, j_k) \in \{1, \ldots, r\}^{2k}$ such that

$$s_{i_1} \neq s_{j_1}, \quad s_{i_1}s_{i_2} \neq s_{j_1}s_{j_2}, \quad \ldots, \quad \text{and } s_{i_1}s_{i_2}\cdots s_{i_k} \neq s_{j_1}s_{j_2}\cdots s_{j_k}$$

in the polynomial ring $\mathbb{Z}[s_1, \ldots, s_r]$. For example, $h_0 = 1$; $h_1 = r^2 - r$; and $h_2 = r^4 - r^3 - r^2 + r$.

The degree-$\le k$ generic uniform heuristic nonrandomness factor is $1/\sqrt{h_k/r^{2k}}$. The goal of this appendix is to compute $\lim_{k\to\infty} 1/\sqrt{h_k/r^{2k}}$.

Define H_k as the sum of quotients $s_{i_1}\cdots s_{i_k}/s_{j_1}\cdots s_{j_k}$ over the same tuples $(i_1, \ldots, i_k, j_1, \ldots, j_k)$ counted by h_k. For $k \ge 1$ the product

$$H_{k-1}U = H_{k-1}\sum_{i_k}\sum_{j_k}\frac{s_{i_k}}{s_{j_k}}$$

is the sum of quotients $s_{i_1}\cdots s_{i_k}/s_{j_1}\cdots s_{j_k}$ over the tuples $(i_1, \ldots, i_k, j_1, \ldots, j_k)$ with

$$s_{i_1} \neq s_{j_1}, \quad s_{i_1}s_{i_2} \neq s_{j_1}s_{j_2}, \quad \ldots, \quad \text{and } s_{i_1}s_{i_2}\cdots s_{i_{k-1}} \neq s_{j_1}s_{j_2}\cdots s_{j_{k-1}}.$$

These are the same as the tuples contributing to H_k, except for tuples having $s_{i_1}s_{i_2}\cdots s_{i_k} = s_{j_1}s_{j_2}\cdots s_{j_k}$. The product $H_{k-1}U$ is therefore the same as H_k, except for its constant coefficient. The constant coefficient of H_k is 0, so $H_k = H_{k-1}U - c_k$ where c_k is the constant coefficient of $H_{k-1}U$.

By induction $H_k = U^k - c_1U^{k-1} - c_2U^{k-2} - \cdots - c_k$. Recall that the constant coefficient of U^k is u_k, so $0 = u_k - c_1u_{k-1} - c_2u_{k-2} - \cdots - c_k$. In other words, $(1 - c_1x - c_2x^2 - \cdots)(1 + u_1x + u_2x^2 + \cdots) = 1$ in the power-series ring $\mathbb{Z}[\![x]\!]$. For the same reason, the product $(1 - c_1x - \cdots - c_kx^k)(1 + u_1x + \cdots + u_kx^k)$ is $1 - (c_1u_k + \cdots + c_ku_1)x^{k+1} - \cdots - c_ku_kx^{2k}$, so

$$\left(1 - \frac{c_1}{r^2} - \cdots - \frac{c_k}{r^{2k}}\right)\left(1 + \frac{u_1}{r^2} + \cdots + \frac{u_k}{r^{2k}}\right) = 1 - \epsilon_k$$

where $\epsilon_k = (c_1u_k + \cdots + c_ku_1)/r^{2k+2} + \cdots + c_ku_k/r^{4k}$. The bounds

$$0 \le \epsilon_k \le \frac{u_{k+1}}{r^{2k+2}} + \frac{u_{k+2}}{r^{2k+4}} + \cdots$$

show that $\epsilon_k \to 0$ as $k \to \infty$, so

$$\left(1 - \frac{c_1}{r^2} - \frac{c_2}{r^4} - \cdots\right)\left(1 + \frac{u_1}{r^2} + \frac{u_2}{r^4} + \cdots\right) = 1.$$

Mapping $s_1 \mapsto 1$, $s_2 \mapsto 1$, …, $s_r \mapsto 1$ takes H_k to h_k and takes U to r^2, so $h_k = h_{k-1}r^2 - c_k$; that is, $h_k/r^{2k} = h_{k-1}/r^{2k-2} - c_k/r^{2k}$. By induction,

$$\frac{h_k}{r^{2k}} = 1 - \frac{c_1}{r^2} - \frac{c_2}{r^4} - \cdots - \frac{c_k}{r^{2k}}.$$

Hence

$$\lim_{k\to\infty} \frac{h_k}{r^{2k}} = 1 - \frac{c_1}{r^2} - \frac{c_2}{r^4} - \cdots = \frac{1}{1 + u_1/r^2 + u_2/r^4 + \cdots}.$$

The desired value $\lim_{k\to\infty} 1/\sqrt{h_k/r^{2k}}$ is therefore the square root of the sum $\sum_k u_k/r^{2k}$ computed above. In particular, for $r = 6$ we find

$$\lim_{k\to\infty} \frac{1}{\sqrt{h_k/r^{2k}}} \approx 1.129162.$$

References

[1] Daniel V. Bailey, Lejla Batina, Daniel J. Bernstein, Peter Birkner, Joppe W. Bos, Hsieh-Chung Chen, Chen-Mou Cheng, Gauthier van Damme, Giacomo de Meulenaer, Luis Julian Dominguez Perez, Junfeng Fan, Tim Güneysu, Frank Gurkaynak, Thorsten Kleinjung, Tanja Lange, Nele Mentens, Ruben Niederhagen, Christof Paar, Francesco Regazzoni, Peter Schwabe, Leif Uhsadel, Anthony Van Herrewege, and Bo-Yin Yang, *Breaking ECC2K-130*, Cryptology ePrint Archive, Report 2009/541, 2009. http://eprint.iacr.org/2009/541

[2] Pierre Barrucand, *Sur la somme des puissances des coefficients multinomiaux et les puissances successives d'une fonction de Bessel*, C. R. Acad. Sci. Paris **258** (1964), 5318–5320. MR 29 #40

[3] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe, *On the correct use of the negation map in the Pollard rho method*, in Catalano et al. [8], 2011, pp. 128–146, expanded version at http://eprint.iacr.org/2011/003. MR 2012h:94145

[4] Simon R. Blackburn and Sean Murphy, *The number of partitions in Pollard rho*, technical report RHUL-MA-2011-11, Department of Mathematics, Royal Holloway, University of London, 2011. http://www.ma.rhul.ac.uk/static/techrep/2011/RHUL-MA-2011-11.pdf

[5] Jonathan M. Borwein, Dirk Nuyens, Armin Straub, and James Wan, *Some arithmetic properties of short random walk integrals*, Ramanujan J. **26** (2011), no. 1, 109–132. MR 2012j:60114

[6] Joppe W. Bos, Thorsten Kleinjung, and Arjen K. Lenstra, *On the use of the negation map in the Pollard rho method*, in Hanrot et al. [12], 2010, pp. 66–82. MR 2011k:11175

[7] Richard P. Brent and John M. Pollard, *Factorization of the eighth Fermat number*, Math. Comp. **36** (1981), no. 154, 627–630. MR 83h:10014

[8] Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi (eds.), *Public key cryptography — PKC* 2011*: Proceedings of the* 14th *International Conference on Practice and Theory in Public Key Cryptography held in Taormina, March* 6–9, 2011, Lecture Notes in Computer Science, no. 6571, Springer, Heidelberg, 2011. MR 2012h:94007

[9] M. Chateauneuf, A. C. H. Ling, and D. R. Stinson, *Slope packings and coverings, and generic algorithms for the discrete logarithm problem*, J. Combin. Des. **11** (2003), no. 1, 36–50, preprint version at http://eprint.iacr.org/2001/094. MR 2003j:05035

[10] Jung Hee Cheon, Jin Hong, and Minkyu Kim, *Speeding up the Pollard rho method on prime fields*, in Pieprzyk [16], 2008, pp. 471–488. MR 2546112

[11] Walter Fumy (ed.), *Advances in cryptology — EUROCRYPT '97: Proceedings of the 16th International Conference on the Theory and Application of Cryptographic Techniques held in Konstanz, May 11–15, 1997*, Lecture Notes in Computer Science, no. 1233, Springer, Berlin, 1997. MR 98i:94002

[12] Guillaume Hanrot, François Morain, and Emmanuel Thomé (eds.), *Algorithmic number theory: Proceedings of the 9th Biennial International Symposium (ANTS-IX) held in Nancy, July 19–23, 2010*, Lecture Notes in Computer Science, no. 6197, Springer, Berlin, 2010. MR 2011g:11002

[13] Martin Hildebrand, *Random walks supported on random points of $\mathbb{Z}/n\mathbb{Z}$*, Probab. Theory Related Fields **100** (1994), no. 2, 191–203. MR 95j:60015

[14] Donald J. Lewis (ed.), *1969 Number Theory Institute: Proceedings of the 1969 Summer Institutes on Number Theory: Analytic Number Theory, Diophantine Problems, and Algebraic Number Theory; held at the State University of New York at Stony Brook, July 7–August 1, 1969*, Proceedings of Symposia in Pure Mathematics, no. 20, American Mathematical Society, Providence, R.I., 1971. MR 47 #3286

[15] V. I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*, Math. Notes **55** (1994), no. 2, 165–172. MR 96a:11145

[16] Josef Pieprzyk (ed.), *Advances in cryptology — ASIACRYPT 2008: Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security held in Melbourne, December 7–11, 2008*, Lecture Notes in Computer Science, no. 5350, Springer, Berlin, 2008. MR 2010j:94005

[17] J. M. Pollard, *Kangaroos, Monopoly and discrete logarithms*, J. Cryptology **13** (2000), no. 4, 437–447. MR 2001i:94059

[18] Bruce Richmond and Cecil Rousseau, *A multinomial summation (Donald Richards and Stamatis Cambanis)*, SIAM Rev. **31** (1989), no. 1, 122–125, comment on Problem 87-2, *SIAM Rev.* **30** (1988), pp. 128–130.

[19] L. B. Richmond and Jeffrey Shallit, *Counting abelian squares*, Electron. J. Combin. **16** (2009), no. 1, Research Paper 72, 9 pages. MR 2010j:05036

[20] J. Sattler and C.-P. Schnorr, *Generating random walks in groups*, Ann. Univ. Sci. Budapest. Sect. Comput. **6** (1985), 65–79. MR 89a:68108

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

[22] Daniel Shanks, *Class number, a theory of factorization, and genera*, in Lewis [14], 1971, pp. 415–440. MR 47 #4932

[23] Victor Shoup, *Lower bounds for discrete logarithms and related problems*, in Fumy [11], 1997, pp. 256–266. MR 98j:94023

[24] Neil J. A. Sloane, *The on-line encyclopedia of integer sequences*, 2012. http://oeis.org

[25] Edlyn Teske, *On random walks for Pollard's rho method*, Math. Comp. **70** (2001), no. 234, 809–825. MR 2001g:11194

DANIEL J. BERNSTEIN: `djb@cr.yp.to`
Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053, United States

TANJA LANGE: `tanja@hyperelliptic.org`
Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

VOLUME EDITORS

Everett W. Howe
Center for Communications Research
4320 Westerra Court
San Diego, CA 92121-1969
United States

Kiran S. Kedlaya
Department of Mathematics
University of California, San Diego
9500 Gilman Drive #0112
La Jolla, CA 92093-0112

ISSN: 2329-9061 (print), 2329-907X (electronic)

ISBN: 978-1-935107-00-2 (print), 978-1-935107-01-9 (electronic)

First published 2013.

msp

MATHEMATICAL SCIENCES PUBLISHERS
798 Evans Hall #3840, c/o University of California, Berkeley CA 94720-3840
contact@msp.org http://msp.org

THE OPEN BOOK SERIES 1

Tenth Algorithmic Number Theory Symposium

The Algorithmic Number Theory Symposium (ANTS), held biennially since 1994, is the premier international forum for research in computational number theory. ANTS is devoted to algorithmic aspects of number theory, including elementary, algebraic, and analytic number theory, the geometry of numbers, arithmetic algebraic geometry, the theory of finite fields, and cryptography.

This volume is the proceedings of the tenth ANTS meeting, held July 9–13, 2012, at the University of California, San Diego. It includes revised and edited versions of the 25 refereed papers presented at the conference, together with extended abstracts of two of the five invited talks.

TABLE OF CONTENTS