J. Cryptology (2000) 13: 437–447
DOI: 10.1007/s001450010010

Journal of
CRYPTOLOGY

Kangaroos, Monopoly and Discrete Logarithms

J. M. Pollard
Tidmarsh Cottage, Manor Farm Lane, Tidmarsh,
Reading RG8 8EX, England
jmptidcott@hotmail.com

Communicated by Andrew Odlyzko

Received 23 January 1998 and revised 27 September 1999
Online publication 10 August 2000

Abstract. The kangaroo method computes a discrete logarithm in an arbitrary cyclic group, given that the value is known to lie in a certain interval. A parallel version has been given by van Oorschot and Wiener with "linear speed-up". We improve the analysis of the running time, both for serial and parallel computers. We explore the variation of the running time with the set of "jumps" of the kangaroos, and confirm that powers of two are a good choice (we do not claim they are the best choice). We illustrate the theory with some calculations of interest to Monopoly players, and the method itself with a card trick due to Kruskal.

Key words. Discrete logarithms, Kangaroo method.

1. Introduction

Pollard [5] gave two method for discrete logs in $GF(p)$, the "rho" method and the "kangaroo" method (or "lambda method"). Both use very little storage, and can be employed in any cyclic group, unlike the index calculus methods. For the rho method, the order of the group, g say, must be known. The method runs in time $O(q^{1/2})$, where q is the largest prime factor of g.

In the kangaroo method, we do not need to know g—instead, the discrete logarithm must be known to lie in some interval. The time is now $O(w^{1/2})$, if w is the width of the interval. We can take $w = g$ (if g is known), but then the first method is best (see below).

van Oorschot and Wiener [11] showed how to give efficient parallel versions of both methods, achieving linear speed-up. Thus the running times become $O(q^{1/2}/P)$ and $O(w^{1/2}/P)$, with P processors. This behaviour is not shared by the "rho" method for factoring, where we gain only a factor of $P^{1/2}$ by using P processors, see [1]. The algorithms of [11] in fact improve on those of [5] even in the serial case, but are slightly more complex.

Further improvements to the rho method have been made by Teske [8]–[10]. The rho method in [8] and [10] is faster, and can find the order of the group, unlike that of [5]. The algorithm of [9] can find the structure of the group.

The best versions of the rho method take close to $(\pi q/2)^{1/2}$ (expected) group operations. This compares with close to $2w^{1/2}$ expected operations for the kangaroo method [11]. Thus, if $w = g = q$, the kangaroo method is expected to take 1.60 times longer.

Here we aim to improve the analysis of the running time of the kangaroo method, for both serial and parallel computers. Our serial algorithm is that of [11]. Our parallel algorithm is different, and has the property that two kangaroos of the "same herd" cannot meet. This is convenient for both implementation and analysis. It also allows us to impose congruence conditions on the required discrete log (see Section 6). This might sometimes be of value. The running times of the two parallel algorithms are thought to be the same—experiments of Stein and Teske [7] seem to support this.

We confirm that the methods work well when the jumps of the kangaroos are powers of two, as suggested in [5] and [11], or powers of another number. We are not claiming that these are the only good choices. Possibly most sufficiently large sets are good choices.

The original name [5] was the "lambda method for catching kangaroos". This lengthy title matched that of the preceding section. We need a shorter name. Here we prefer "kangaroo method" to "lambda method". The latter is perhaps confusing since the parallel version of the rho method is shaped like a lambda, not a rho.

The kangaroos came from a fascinating article [3] in *Scientific American*—"Energetic cost of locomotion, measured in terms of oxygen consumption at various speeds, was determined by placing kangaroos on a treadmill. . . " (The same issue contained Martin Gardner's exposition of the RSA method).

I am grateful to Michael Wiener for his comments and the numerical calculations mentioned in Section 5, and to the referee for helpful comments. Sections 2 and 8 are perhaps recreational mathematics, but do illustrate the algorithm and its analysis.

2. A Card Trick

Gardner [4] describes the following trick, which he calls "Kruskal's principle". It has a remarkable resemblance to the kangaroo method. Take a well-shuffled single pack of cards. Deal it face up in a row, from left to right. Place a coin on a card near the right-hand end, apparently chosen at random. Ask a friend to choose a number between 1 and 10. If he chooses 4, count to the fourth card from the left. If this is a 7, count on a further seven cards. Continue in this way. When you land on a picture, count five cards. Hopefully you will land on the coin.

The trick does not always succeed! It only works with probability "about $5/6$", according to [4]. My recent experiments [6] confirm this. At the end of the paper, we explain how to do the trick.

3. The Kangaroo Method on a Serial Computer

Given a cyclic group G, we aim to solve an equation:

$$r^z = q, \qquad L \le z \le U. \tag{1}$$

Put $w = U - L$. We can use the "baby-step giant-step" method of Shanks (see [2]) to find z in time and space $O(w^{1/2})$. The method requires us to find the common member of two sets of about $w^{1/2}$ group elements. We give two ways. In the first, we compute the first set, storing it in a hash table. Then we compute the second, one member at a time, checking to see if it is in the table. On average, we do $3w^{1/2}/2$ group operations.

The second way is slightly faster, but may require up to twice the storage. Compute members of both sets alternately, storing all of them in the same hash table. On average, this will require only $4w^{1/2}/3$ group operations. The constant $\frac{4}{3}$ arises for the following reason. If two random numbers x and y are chosen in the interval [0, 1], the expected value of $\max(x, y)$ is $\frac{2}{3}$.

The kangaroo method (of [11]) can solve (1) in about $2w^{1/2}$ expected operations, but with a very small storage requirement. The earlier version of [5] is expected to take 1.64 times longer [11, p. 13].

We choose a small set of positive integers:

$$S = (s_1, \ldots, s_k), \tag{2}$$

the set of "jumps" of the kangaroos. The integers, which may contain repetitions, are listed in nondecreasing order. The main purpose of this paper is to explore the effect of this choice. The mean m of the set will be comparable with $w^{1/2}$.

We compute the elements:

$$R_i = r^{s_i}, \qquad i = 1, \ldots, k.$$

Let $h(x)$ be a (conventional) hash function mapping G into the interval $[1 \cdots k]$. Finally, let D be a subset of G, the set of "distinguished points". It will be appropriate to take

$$|D|/|G| = c/w^{1/2},$$

where c is a constant, $c \gg 1$.

A kangaroo is a sequence of elements of G of the form

$$x_{i+1} = x_i * R_{h(x_i)}, \qquad i = 1, 2, \ldots, \tag{3}$$

starting from a given x_0. We also define a sequence d_i, putting $d_0 = 0$ and

$$d_{i+1} = d_i + s_{h(x_i)}, \qquad i = 1, 2, \ldots.$$

Thus d_i is the "distance" travelled by the kangaroo in its first i jumps, and

$$x_i = x_0 * r^{d_i}, \qquad i = 0, 1, \ldots.$$

We give a brief account of the original kangaroo method of [5]. There are two kangaroos. Our "tame" kangaroo T starts at the right-hand end of the interval, i.e. $x_0 = r^U$. After $N = O(w^{1/2})$ jumps, he stops and we store the pair (x_N, d_N). His final position is our "trap". The "wild" kangaroo W starts at at unknown position in the interval, with $x'_0 = q$. *If* his path meets that of T, he continues down that path and falls into the trap, i.e. $x'_i = x_N$. Then we can compute z in (1). Indeed we have $z = U + d_N - d'_i$. The method succeeds with a probability which depends on N.

Now we give the improved algorithm of [11] (for a serial computer). The two kangaroos now make alternate jumps, and T starts at the *middle* of the interval, i.e. $x_0 = r^M$, where $M = [(L + U)/2]$. Whenever x_i falls in D, we store the pair (x_i, d_i) together with the name of the kangaroo (T or W). This information may be stored in a hash table, hashing on x_i. When some x_i is stored for the second time, having been reached by both kangaroos, we can compute the solution of (1). Thus, if x_i (for T) $= x'_j$ (for W), we have $z = M + d_i - d'_j$.

In this version, unlike [5], we do not know which kangaroo is in the lead, but the idea is the same. When the back kangaroo B lands on the path of the front kangaroo F, he continues along that path until he reaches a point of D, when the algorithm stops. *This* algorithm is almost certain to succeed if $w \ll p$.

4. Approximate Analysis

We estimate the time in terms of group operations. The algorithm falls into three stages:

Stage 1. B must catch up to the starting point of F. This involves travelling an expected distance $w/4$, in steps of mean m. The expected time is $w/2m$ (since both kangaroos jump alternately).

Stage 2. B must land on the path of F. We have the following simple argument. This path includes a fraction $1/m$ of the integers. Thus B lands on the path with probability $1/m$ at each jump, and takes an expected m jumps to do so. (We see later that this is generally optimistic). Thus Stage 2 takes time about $2m$ units.

Stage 3. B continues until he reaches a point of D. This takes an expected time $2w^{1/2}/c$.

The expected total time is

$$E = w/2m + 2m + 2w^{1/2}/c.$$

The best choice of m is $w^{1/2}/2$, giving

$$E = 2w^{1/2}(1 + 1/c), \tag{4}$$

or if c is large, $E \simeq 2w^{1/2}$. Equation (4) is given in [11]. We see that the expected time varies from $w^{1/2}$ when z is near the middle of the interval, to $3w^{1/2}$ when it is near the ends. If z is outside the interval, the algorithm will still work but will take longer.

The value of c decides the storage requirement. The number of pairs (x_i, d_i) we expect to store is $Ec/w^{1/2} = 2(c + 1)$.

5. Improved Analysis

We say that Stage 1 ends when the back kangaroo B lands beyond the starting point of the front kangaroo F (by an amount i, say), or lands on this point ($i = 0$). The probability

distribution of $i, q(i)$ say, is easily found. We have $q(0) = 1/m$. For $i \geq 1$, we must multiply by the probability that the last jump exceeds i. Thus

$$q(i) = n(i)/mk, \qquad i = 0, \ldots, s_k - 1,$$

where $n(i) = \#\{j, s_j > i\}$ (that is, the number of elements of S greater than i).

Now fix $i > 0$. We must find the expected numbers of jumps, $f(i)$ and $b(i)$ say, of the front and back kangaroos up to the point when their paths meet (f refers to the one which is *now* in front). It is convenient here to imagine that it is always the hindmost kangaroo which moves, jumping a distance which is a random choice from the set S. We stop when both are on the same point. Then their numbers of jumps will give the required functions.

It is interesting to extend these functions to the case $i = 0$. Let $f(0) = b(0)$ denote the expected number of jumps when both kangaroos start at the same point. Their first jumps are now independent random choices from S, and may or may not be the same. We state the

Theorem 1.

$$f(0) = b(0) = m.$$

We give a proof in Section 7. A short informal proof is also possible (see Section 8). For the other values, it seems that there is in general no such simple answer. By considering one jump we get the sparse equations:

$$f(i) = \frac{1}{k}\left\{\sum_{s_j<i} f(i - s_j) + \sum_{s_j>i} b(s_j - i)\right\}, \qquad i = 1 \cdots s_k, \tag{5}$$

$$b(i) = 1 + \frac{1}{k}\left\{\sum_{s_j<i} b(i - s_j) + \sum_{s_j>i} f(s_j - i)\right\}, \qquad i = 1 \cdots s_k. \tag{6}$$

In the same way we find

$$m = f(0) = 1 + \frac{1}{k}\sum_{j=1}^{k} f(s_j) = \frac{1}{k}\sum_{j=1}^{k} b(s_j). \tag{7}$$

We can solve (5) and (6) by the relaxation method. Then (7) gives a check on the result. A better method is to use (7) to speed-up the convergence. After each step of the relaxation method, multiply all the values of $b(i)$ and $f(i)$ by the factor m/X, where X is the last expression in (7).

We can now given an improved estimate for the time for Stage 2, namely, $2m \cdot A(S)$, where

$$A(S) = \frac{1}{m}\sum_{i=1}^{s_k-1} q(i) * f(i).$$

Now the best choice for m is $m = \{w/A(S)\}^{1/2}/2$, giving an improved estimate for the expected total time

$$E = 2w^{1/2}\{B(S) + 1/c\},$$

where $B(S) = \{A(S)\}^{1/2}$. If we again assume c is large,

$$E \simeq 2w^{1/2} \cdot B(S). \tag{8}$$

So our previous estimate $2w^{1/2}$ has been multiplied by $B(S)$. This assumes that $A(S)$ is known. If not, we take $m = w^{1/2}/2$ as before, and find that the time is multiplied by the factor $B'(S) = \{A(S) + 1\}/2$. Of course, $B(S)$ and $B'(S)$ are close when $A(S)$ is close to 1.

Example 1. $S = (1, \ldots, k), m = (k+1)/2$.

In this simple case, we can give exact formulae:

$$\begin{aligned} b(i) &= k/2 + i/(k+1), \qquad i = 1, \ldots, k-1, \\ f(i) &= k/2 - i/(k+1), \qquad i = 1, \ldots, k-1, \\ A(S) &= (3k-2)(k-1)/3(k+1)^2, \end{aligned}$$

so $A(S) \to 1$ and $B(S) \to 1$ as $k \to \infty$. Of course, such a large set is not acceptable.

Example 2. $S = (1, x), (x\ odd), m = (x+1)/2$.

Here the values of $f(i)$ are very variable, e.g.

$$\begin{aligned} f(1) &= x/2 - 1/(x+1), \\ f(m) &= (x-1)(x+3)/8. \end{aligned}$$

We find that $A(S) \sim m/3$ as $m \to \infty$. This means that the running time is no longer $O(w^{1/2})$ (at best it is $O(w^{2/3})$).

Now we give some numerical results for a case of serious interest, namely,

$$S = (1, x, x^2, \ldots, x^{k-1}).$$

The use of powers of two was suggested in [5] and [11]. The results below were mainly calculated by Wiener [12]. The corresponding function $B(S)$ we now write as $C(x, k)$:

k	$C(2,k)$	$C(4,k)$	$C(8,k)$	$C(16,k)$
3	0.62736	1.00813	1.27473	1.48462
4	0.79627	1.10292	1.22768	1.28447
5	0.90913	1.12056	1.16814	1.18001
6	0.98075	1.11126	1.12674	1.12885
7	1.02337	1.09610	1.10033	
8	1.04657	1.08212		
9	1.05741	1.07082		
10	1.06081	1.06198		
11	1.06005	1.05503		
12	1.05717			
13	1.05340			
14	1.04943			
16	1.04210			
20	1.03157			

We conjecture that each column tends to 1, indeed that $C(n,k) \to 1$ as $k \to \infty$, for any n.

Possibly even smaller sets might be used. An interesting question is the behaviour of $C(x,3), C(x,4), \ldots$ as $x \to \infty$. More numerical calculations of Wiener [12] show these to be increasing functions, but possibly tending to limits. Note that $C(x,2)$ was discussed above.

6. The Kangaroo Method on a Parallel Computer

Again we want to solve (1). A bad method is to divide the interval $[L, U]$ into P equal parts, and use each of P processors to search one part, using the serial method. This will take time $O((w/P)^{1/2})$.

van Oorschot and Wiener [11] employ two herds of kangaroos in a method which runs in time $O((w^{1/2}/P))$. We analyse a variation of their method, which avoids the possibility of "collisions" between members of the same herd.

The numbers of kangaroos in the herds will be coprime integers u and v, nearly equal, with $u + v \leq P$. Thus if $P = 4p$, we could take $u = 2p + 1$ and $v = 2p - 1$. Their jumps will all be multiples of uv, say

$$S' = (uvs_1, \ldots, uvs_k).$$

As before let $M = [(L + U)/2]$. The tame kangaroos, known as $T_0, \ldots, T_{u-1}$, start at positions

$$M, M + v, \ldots, M + (u-1)v.$$

The wild kangaroos $W_0, \ldots, W_{v-1}$ start at positions

$$z, z + u, \ldots, z + (v-1)u.$$

The appropriate values of x_0 can be found from (1).

Now consider the equation

$$M + iv = z + ju \pmod{uv}, \qquad 0 \leq i \leq u-1, \quad 0 \leq j \leq v-1. \tag{9}$$

There is a unique solution which we could find, if we knew z, by solving for $i \pmod u$ and $j \pmod v$. Thus there is a pair, T_i and W_j, which move in the same residue class $(\mathrm{mod}\ uv)$. This is the only pair whose paths can meet. The others are wasting their time and ours! Sadly, we do not know which pair matters and have to run them all.

We describe the algorithm. The kangaroos all advance simultaneously, one on each of $u + v$ processors. Whenever a processor computes a value x_i in D, the pair (x_i, d_i) is sent to a special processor, together with the name of the kangaroo (T_i or W_j). When some x_i has been stored twice, necessarily by kangaroos of different types, we can compute the solution of (1).

As we said, there is a single pair T_i and W_j which matters. To analyse the algorithm, we can suppose we know which it is. We estimate the total number of jumps of this pair. This is just the time of the serial algorithm with interval w/uv, and set of steps S in (2).

Thus we should choose $m \simeq (w/uv)^{1/2}/2$ and obtain the required result from (8), with w/uv for w. Finally, the expected time of the parallel algorithm is half of this, since T_i and W_j run simultaneously, and is

$$E_p = (w/uv)^{1/2} \cdot B(S),$$

where $B(S)$ is the correction factor for the serial algorithm with set S as in (2).

Thus the parallel algorithm speeds up the serial one by a factor $2(uv)^{1/2}$. Since $u \sim v \sim P/2$, this is close to P, as claimed.

Usually, as we said, we do not know which pair of kangaroos will meet. However, suppose that the required integer z is known to lie in a certain set of u' residue classes (mod u). Then (9) restricts the possible values of i to a set of u' values, and only u' tame kangaroos need be used. Similarly, a restriction on the values of z(mod v) reduces the number of wild kangaroos.

Suppose now we have information on z modulo a number of small primes. Then we divide the primes into two sets with roughly equal products, and take these as u and v. The final algorithm can be implemented on a serial computer if required.

The situation just described arises if we wish to implement Fermat's method of factoring with kangaroos (see [5]). However, we believe it may have more topical applications.

7. Proof of the Theorem

As in Section 5, imagine that the hindmost kangaroo jumps by a random amount from the set S of (2). If we cease to distinguish between the kangaroos, we have a Markov process whose state is just the size of the gap, with values $0, \ldots, s_k$. From state i, we move to $|i - s_j|$, where j is random in $1, \ldots, k$. In this statement, we allow $i = 0$, i.e. we allow the process to continue indefinitely.

The process has a steady-state probability distribution $p(i)$, $i = 0, \ldots, s_k$, given by

$$p(0) = \frac{1}{k} \cdot \sum_{j=1}^{k} p(s_j),$$

$$p(i) = \frac{1}{k} \cdot \left\{ \sum_{i+s_j \le s_k} p(i + s_j) + \sum_{s_j \ge i} p(s_j - i) \right\}, \qquad i = 1, \ldots, s_k,$$

and

$$\sum_{i=0}^{s_k} p(i) = 1.$$

The solution is found to have a simple form. Let $N(i)$ be the number of occurrences of i in S, let $n(i)$ be the number of s_j greater than i (as above), and let

$$s = s_1 + \cdots + s_k = km.$$

Then

$$p(0) = 1/2m,$$
$$p(i) = (N(i) + 2n(i))/2s, \qquad i = 1, \ldots, s_k.$$

Thus state 0 recurs after a mean time $1/p(0) = 2m$. This is equivalent to our theorem.

Example 3. $S = (1, 2, 4, 8)$, $s = 15$, $m = \frac{15}{4}$.

$$\begin{aligned} p(0) &= \tfrac{2}{15}, \\ p(1) &= \tfrac{7}{30}, \\ p(2) &= \tfrac{5}{30}, \\ p(3) &= \tfrac{4}{30}, \\ p(4) &= \tfrac{3}{30}, \\ p(5) &= p(6) = p(7) = \tfrac{2}{30}, \\ p(8) &= \tfrac{1}{30}. \end{aligned}$$

Note that the $p(i)$ are always nonincreasing for $i > 0$, as here. The distribution $p(i)$ is quite similar to but *not* the same as the distribution $q(i)$ of Section 5. The latter describes the width of the gap immediately after a crossing (or meeting) of the kangaroos.

8. Coda for Monopoly Players

In Monopoly the players move round a circular board of 40 squares, according to the totals of two dice. We ignore special rules such as "going to jail", and unwind the board, numbering the squares 0, 1, 2, ... The players start on square 0.

We consider first a single player. What is the chance that he lands on square n? The mean throw of two dice is 7, being twice the mean throw of a single dice ($\frac{7}{2}$). Thus for large n, $p(n) \simeq \frac{1}{7}$. Clearly, this is not true for small n. Thus $p(1) = 0$ and $p(2) = \frac{1}{36}$.

To calculate exact values we use the equation

$$p(i) = \tfrac{1}{36}p(i-2) + \tfrac{2}{36}p(i-3) + \cdots + \tfrac{1}{36}p(i-12),$$

where the coefficients $\frac{1}{36}, \frac{2}{36}, \ldots, \frac{7}{36}, \frac{6}{36}, \ldots, \frac{1}{36}$ are the probabilities of throwing $2, \ldots, 12$. The initial conditions are $p(0) = 1$ and $p(i) = 0$ for $i < 0$. We find $p(i)$ shows decreasing oscillations about the value $\frac{1}{7} = 0.1429$:

$$\begin{aligned} p(1) &= 0, \\ p(2) &= 0.0278, \\ p(3) &= 0.0556, \\ p(4) &= 0.0841, \\ p(5) &= 0.1142, \\ p(6) &= 0.1466, \\ p(7) &= 0.1822, \qquad \text{first peak (and maximum value)}, \\ p(8) &= 0.1663, \\ p(9) &= 0.1555, \end{aligned}$$

$$
\begin{aligned}
p(10) &= 0.1478,\\
p(11) &= 0.1413,\\
p(12) &= 0.1342,\\
p(13) &= 0.1247, \qquad \text{first trough},\\
p(14) &= 0.1386,\\
p(15) &= 0.1458,\\
p(16) &= 0.1484, \qquad \text{second peak},\\
&\vdots\\
p(40) &= 0.1428.
\end{aligned}
$$

It is of interest to know the probability of landing on one (or more) squares of an "estate" (or colour group). These are of two kinds:

(i) Those of two squares of form n and $n+2$. Assuming n is large, the required probability is $\frac{1}{7}+\frac{1}{7}-\frac{1}{7}*\frac{1}{36}=0.2811$.
(ii) Those of three squares of form n, $n+1$ and $n+3$ OR n, $n+2$ and $n+3$. These are mirror images, and have the same probability (for large n) which is found to be $\frac{5}{12}=0.4167$.

Note. These values assume that you are currently standing on square 0—if you are on square $n-1$ the probabilities are much smaller!

Now we consider two players, starting on square 0. How soon will they land on the same square (not necessarily after the same numbers of turns)? Suppose they both land on square n, after t_1 and t_2 turns, respectively. Then we have the expected values

$$E(t_1) = E(t_2) = m = 7, \tag{10}$$
$$E(n) = m^2 = 49. \tag{11}$$

The first statement is just the theorem of Section 5. The second easily follows. For each player expects to move a distance m^2 in m turns. Of course, the probability that both players do land on square 49 is small, close to $1/49$.

We can also prove (10) and (11) directly, at least in an informal way, and deduce the theorem. For large n, both players land on square n with probability close to $1/m^2$. Thus a fraction $1/m^2$ of all squares have this property. The expected distance between two consecutive squares of this kind is m^2, giving (11). However, each player takes an expected m turns to travel a distance m^2, giving (10).

Now suppose the players start a small distance i apart. We again ask how soon they will land on the same square. The expected numbers of turns of the front and back players are now the functions $f(i)$ and $b(i)$ of Section 5. We now take

$$S = (2, 3, 3, 4, 4, 4, \ldots, 12), \qquad m = 7.$$

We find that both are "V-shaped" with minimum at $i = 7$. Here are a few values:

$$\begin{aligned} b(1) &= 7.0941, & f(1) &= 6.9512, \\ b(7) &= 6.7079, & f(7) &= 5.7079, \\ b(12) &= 7.7388, & f(12) &= 6.0245. \end{aligned}$$

The functions are in fact related:

$$b(i) = 1 + f(14 - i), \qquad i = 2, \ldots, 12.$$

For completeness we give $A(s) = 0.7831$, $m \cdot A(S) = 5.4814$.

Finally we explain the card trick of Section 2. Start a tame kangaroo at the first card. It is quite easy to follow his path as the cards are dealt. Thus, if the first card is a 5, count silently up to 5 as the next five cards are dealt. Without a pause, note the value of the last card and start another count. Place the coin on the last card of the last completed count. For more about the trick, see [6].

References

[1] R. P. Brent, Parallel algorithms for integer factorisation, in J. H. Loxton (ed.), *Number Theory and Cryptography*, LMS Lecture Note Series 154, Cambridge University Press, Cambridge, 1990, pp. 26–37.

[2] H. Cohen, *A Course in Computational Algebraic Number Theory*, Graduate Texts in Mathematics 138, Springer-Verlag, Berlin, 2nd printing, 1995.

[3] T. J. Dawson, Kangaroos, *Scientific American*, August 1977, pp. 78–89.

[4] M. Gardner, Mathematical games, *Scientific American*, Feb. 1978, pp. 19–32.

[5] J. M. Pollard, Monte Carlo methods for index computation (mod p), *Math. Comp.*, **32** (1978), 918–924.

[6] J. M. Pollard, Kruskal's card trick, *Math. Gaz.*, **84** (July 2000), 46–49.

[7] A. Stein and E. Teske, Catching kangaroos in function fields, *Proceedings of the Conference on the Mathematics of Public-Key Cryptography*, Toronto, June 1999.

[8] E. Teske, Speeding up Pollard's rho method for computing discrete logarithms, *ANTS* III (1998), Lecture Notes in Computer Science, Vol. 1423, Springer-Verlag, Berlin, 1998, pp. 541–554.

[9] E. Teske, A space efficient algorithm for group structure computation, *Math Comp.*, **67** (1998), 1637–1663.

[10] E. Teske, On random walks for Pollard's rho method, *Math. Comp.*, posted on February 18, 2000, PII: 5 0025-5718(00)01213-8 (to appear in print).

[11] P. C. van Oorschot and M. J. Wiener, Parallel collision search with cryptanalytic applications, *J. Cryptology*, **12** (1999), 1–28.

[12] M. J. Wiener, Personal communications, 6 Oct. and 6 Nov. 1997.