Chapter 14

Factoring and Discrete Logarithms using Pseudorandom Walks

This chapter is devoted to the rho and kangaroo methods for factoring and discrete logarithms (which were invented by Pollard) and some related algorithms. These methods use pseudorandom walks and require low storage (typically a polynomial amount of storage, rather than exponential as in the time/memory tradeoff). Although the rho factoring algorithm was developed earlier than the algorithms for discrete logarithms, the latter are much more important in practice.[1] Hence we focus mainly on the algorithms for the discrete logarithm problem.

As in the previous chapter, we assume G is an algebraic group over a finite field $\mathbb{F}_q$ written in multiplicative notation. To solve the DLP in an algebraic group quotient using the methods in this chapter one would first lift the DLP to the covering group (though see Section 14.4 for a method to speed up the computation of the DLP in an algebraic group by essentially working in a quotient).

14.1 Birthday Paradox

The algorithms in this chapter rely on results in probability theory. The first tool we need is the so-called "birthday paradox". This name comes from the following application,

[1] Pollard's paper [487] contains the remark "We are not aware of any particular need for such index calculations" (i.e., computing discrete logarithms) even though [487] cites the paper of Diffie and Hellman. Pollard worked on the topic before hearing of the cryptographic applications. Hence Pollard's work is an excellent example of research pursued for its intrinsic interest, rather than motivated by practical applications.

which surprises most people: among a set of 23 or more randomly chosen people, the probability that two of them share a birthday is greater than 0.5 (see Example 14.1.4).

Theorem 14.1.1. *Let $\mathcal{S}$ be a set of N elements. If elements are sampled uniformly at random from $\mathcal{S}$ then the expected number of samples to be taken before some element is sampled twice is less than $\sqrt{\pi N/2} + 2 \approx 1.253\sqrt{N}$.*

The element that is sampled twice is variously known as a **repeat**, **match** or **collision**. For the rest of the chapter, we will ignore the $+2$ and say that the expected number of samples is $\sqrt{\pi N/2}$.

Proof: Let X be the random variable giving the number of elements selected from $\mathcal{S}$ (uniformly at random) before some element is selected twice. After l distinct elements have been selected then the probability that the next element selected is also distinct from the previous ones is $(1 - l/N)$. Hence the probability $\Pr(X > l)$ is given by

$$p_{N,l} = 1(1 - 1/N)(1 - 2/N) \cdots (1 - (l-1)/N).$$

Note that $p_{N,l} = 0$ when $l \geq N$. We now use the standard fact that $1 - x \leq e^{-x}$ for $x \geq 0$. Hence,

$$\begin{aligned} p_{N,l} \leq 1\, e^{-1/N} e^{-2/N} \cdots e^{-(l-1)/N} &= e^{-\sum_{j=0}^{l-1} j/N} \\ &= e^{-\frac{1}{2}(l-1)l/N} \\ &\leq e^{-(l-1)^2/2N}. \end{aligned}$$

By definition, the expected value of X is

$$\begin{aligned} \sum_{l=1}^{\infty} l \Pr(X = l) &= \sum_{l=1}^{\infty} l(\Pr(X > l-1) - \Pr(X > l)) \\ &= \sum_{l=0}^{\infty} (l + 1 - l) \Pr(X > l) \\ &= \sum_{l=0}^{\infty} \Pr(X > l) \\ &\leq 1 + \sum_{l=1}^{\infty} e^{-(l-1)^2/2N}. \end{aligned}$$

We estimate this sum using the integral

$$1 + \int_0^{\infty} e^{-x^2/2N} dx.$$

Since $e^{-x^2/2N}$ is monotonically decreasing and takes values in $[0, 1]$ the difference between the value of the sum and the value of the integral is at most 1. Making the change of variable $u = x/\sqrt{2N}$ gives

$$\sqrt{2N} \int_0^{\infty} e^{-u^2} du.$$

A standard result in analysis (see Section 11.7 of [340] or Section 4.4 of [636]) is that this integral is $\sqrt{\pi}/2$. Hence, the expected value for X is $\leq \sqrt{\pi N/2} + 2$. □

The proof only gives an upper bound on the probability of a collision after l trials. A lower bound of $e^{-l^2/2N - l^3/6N^2}$ for $N \geq 1000$ and $0 \leq l \leq 2N \log(N)$ is given in

Wiener [631]; it is also shown that the expected value of the number of trials is $> \sqrt{\pi N/2} - 0.4$. A more precise analysis of the birthday paradox is given in Example II.10 of Flajolet and Sedgewick [205] and Exercise 3.1.12 of Knuth [343]. The expected number of samples is $\sqrt{\pi N/2} + 2/3 + O(1/\sqrt{N})$.

We remind the reader of the meaning of expected value. Suppose the experiment of sampling elements of a set $\mathcal{S}$ of size N until a collision is found is repeated t times and each time we count the number l of elements sampled. Then the average of l over all trials tends to $\sqrt{\pi N/2}$ as t goes to infinity.

Exercise 14.1.2. Show that the number of elements that need to be selected from $\mathcal{S}$ to get a collision with probability 1/2 is $\sqrt{2\log(2)N} \approx 1.177\sqrt{N}$.

Exercise 14.1.3. One may be interested in the number of samples required when one is particularly unlucky. Determine the number of trials so that with probability 0.99 one has a collision. Repeat the exercise for probability 0.999.

The name "birthday paradox" arises from the following application of the result.

Example 14.1.4. In a room containing 23 or more randomly chosen people, the probability is greater than 0.5 that two people have the same birthday. This follows from $\sqrt{2\log(2)365} \approx 22.49$. Note also that $\sqrt{\pi 365/2} = 23.944\ldots$.

Finally, we mention that the expected number of samples from a set of size N until $k > 1$ collisions are found is approximately $\sqrt{2kN}$. A detailed proof of this fact is given by Kuhn and Struik as Theorem 1 of [355].

14.2 The Pollard Rho Method

Let g be a group element of prime order r and let $G = \langle g \rangle$. The discrete logarithm problem (DLP) is: Given $h \in G$ to find a, if it exists, such that $h = g^a$. In this section we assume (as is usually the case in applications) that one has already determined that $h \in \langle g \rangle$.

The starting point of the rho algorithm is the observation that if one can find $a_i, b_i, a_j, b_j \in \mathbb{Z}/r\mathbb{Z}$ such that

$$g^{a_i} h^{b_i} = g^{a_j} h^{b_j} \tag{14.1}$$

and $b_i \not\equiv b_j \pmod{r}$ then one can solve the DLP as

$$h = g^{(a_i - a_j)(b_j - b_i)^{-1} \pmod{r}}.$$

The basic idea is to generate pseudorandom sequences $x_i = g^{a_i} h^{b_i}$ of elements in G by iterating a suitable function $f : G \to G$. In other words, one chooses a starting value x_1 and defines the sequence by $x_{i+1} = f(x_i)$. A sequence $x_1, x_2, \ldots$ is called a **deterministic pseudorandom walk**. Since G is finite there is eventually a collision $x_i = x_j$ for some $1 \le i < j$ as in equation (14.1). This is presented as a collision between two elements in the same walk, but it could also be a collision between two elements in different walks. If the elements in the walks look like uniformly and independently chosen elements of G then, by the birthday paradox (Theorem 14.1.1), the expected value of j is $\sqrt{\pi r/2}$.

It is important that the function f be designed so that one can efficiently compute $a_i, b_i \in \mathbb{Z}/r\mathbb{Z}$ such that $x_i = g^{a_i} h^{b_i}$. The next step x_{i+1} depends only on the current step x_i and not on (a_i, b_i). The algorithms all exploit the fact that when a collision

$x_i = x_j$ occurs then $x_{i+t} = x_{j+t}$ for all $t \in \mathbb{N}$. Pollard's original proposal used a cycle-finding method due to Floyd to find a self-collision in the sequence; we present this in Section 14.2.2. A better approach is to use distinguished points to find collisions; we present this in Section 14.2.4.

14.2.1 The Pseudorandom Walk

A true random walk in a finite set $\mathcal{S}$ chooses elements uniformly at random at each stage. In contrast we are concerned with walks whose next step is determined by the current position. Such a walk is given by a function $f : \mathcal{S} \to \mathcal{S}$. Hence, a truly random walk on a finite set means a uniformly chosen function from the set of all functions from $\mathcal{S}$ to itself.

Pollard simulates a random function from G to itself as follows. The first step is to decompose G into n_S disjoint subsets (usually of roughly equal size) so that $G = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_{n_S - 1}$. Traditional textbook presentations use $n_S = 3$ but, as explained in Section 14.2.5, it is better to take larger values for n_S; typical values in practice are 32, 256 or 2048.

The sets $\mathcal{S}_i$ are defined using a selection function $S : G \to \{0, \dots, n_S - 1\}$ by $\mathcal{S}_i = \{g \in G : S(g) = i\}$. For example, in any computer implementation of G one represents an element $g \in G$ as a unique[2] binary string $b(g)$ and interpreting $b(g)$ as an integer one could define $S(g) = b(g) \pmod{n_S}$ (taking n_S to be a power of 2 makes this computation especially easy). To obtain different choices for S one could apply an $\mathbb{F}_2$-linear map L to the sequence of bits $b(g)$, so that $S(g) = L(b(g)) \pmod{n_S}$. These simple methods can be a poor choice in practice, as they are not "sufficiently random". Some other ways to determine the partition are suggested in Section 2.3 of Teske [605] and Bai and Brent [24]. The strongest choice is to apply a hash function or randomness extractor to $b(g)$, though this may lead to an undesirable computational overhead.

Definition 14.2.1. The **rho walks** are defined as follows. Precompute $g_j = g^{u_j} h^{v_j}$ for $0 \le j \le n_S - 1$ where $0 \le u_j, v_j < r$ are chosen uniformly at random. Set $x_1 = g$. The **original rho walk** is

$$x_{i+1} = f(x_i) = \begin{cases} x_i^2 & \text{if } S(x_i) = 0 \\ x_i g_j & \text{if } S(x_i) = j,\ j \in \{1, \dots, n_S - 1\} \end{cases} \tag{14.2}$$

The **additive rho walk** is

$$x_{i+1} = f(x_i) = x_i g_{S(x_i)}. \tag{14.3}$$

An important feature of the walks is that each step requires only one group operation.

Once the selection function S and the values u_j and v_j are chosen, the walk is deterministic. Even though these values may be chosen uniformly at random, the function f itself is not a random function as it has a compact description. Hence, the rho walks can only be described as pseudorandom. To analyse the algorithm we will consider the expectation of the running time over different choices for the pseudorandom walk. Many authors consider the expectation of the running time over all problem instances and random choices of the pseudorandom walk; they therefore write "expected running time" for what we are calling "average-case expected running time".

It is necessary to keep track of the decomposition

$$x_i = g^{a_i} h^{b_i}.$$

[2] One often uses projective coordinates to speed up elliptic curve arithmetic, so it is natural to use projective coordinates when implementing these algorithms. But to define the pseudorandom walk one needs a unique representation for points, so projective coordinates are not appropriate. See Remark 13.3.2.

The values $a_i, b_i \in \mathbb{Z}/r\mathbb{Z}$ are obtained by setting $a_1 = 1, b_1 = 0$ and updating (for the original rho walk)

$$a_{i+1} = \begin{cases} 2a_i \pmod r & \text{if } S(x_i) = 0 \\ a_i + u_{S(x_i)} \pmod r & \text{if } S(x_i) > 0 \end{cases} \quad \text{and} \quad b_{i+1} = \begin{cases} 2b_i \pmod r & \text{if } S(x_i) = 0 \\ b_i + v_{S(x_i)} \pmod r & \text{if } S(x_i) > 0. \end{cases} \tag{14.4}$$

Putting everything together, we write

$$(x_{i+1}, a_{i+1}, b_{i+1}) = \mathsf{walk}(x_i, a_i, b_i)$$

for the random walk function. But it is important to remember that x_{i+1} only depends on x_i and not on (x_i, a_i, b_i).

Exercise 14.2.2. Give the analogue of equation (14.4) for the additive walk.

14.2.2 Pollard Rho Using Floyd Cycle Finding

We present the original version of Pollard rho. A single sequence $x_1, x_2, \dots$ of group elements is computed. Eventually there is a collision $x_i = x_j$ with $0 \le i < j$. One pictures the walk as having a **tail** (which is the part $x_1, \dots, x_i$ of the walk that is not cyclic) followed by the **cycle** or **head** (which is the part $x_{i+1}, \dots, x_j$). Drawn appropriately this resembles the shape of the greek letter ρ. The tail and cycle (or head) of such a random walk have expected length $\sqrt{\pi N/8}$ (see Flajolet and Odlyzko [204] for proofs of these, and many other, facts).

The goal is to find integers i and j such that $x_i = x_j$. It might seem that the only approach is to store all the x_i and, for each new value x_j, to check if it appears in the list. This approach would use more memory and time than the baby-step-giant-step algorithm. If one were using a truly random walk then one would have to use this approach. The whole point of using a deterministic walk which eventually becomes cyclic is to enable better methods to find a collision.

Let l_t be the length of the **tail** of the "rho" and l_h be the length of the **cycle** of the "rho". In other words the first collision is

$$x_{l_t + l_h} = x_{l_t}. \tag{14.5}$$

Floyd's cycle finding algorithm[3] is to compare x_i and x_{2i}. Lemma 14.2.3 shows that this will find a collision in at most $l_t + l_h$ steps. The crucial advantage of comparing x_{2i} and x_i is that it only requires storing two group elements. The rho algorithm with Floyd cycle finding is given in Algorithm 16.

Lemma 14.2.3. *Let the notation be as above. Then $x_{2i} = x_i$ if and only if $l_h \mid i$ and $i \ge l_t$. Further, there is some $l_t \le i < l_t + l_h$ such that $x_{2i} = x_i$.*

Proof: If $x_i = x_j$ then we must have $l_h \mid (i - j)$. Hence the first statement of the Lemma is clear. The second statement follows since there is some multiple of l_h between l_t and $l_t + l_h$. □

Exercise 14.2.4. Let $p = 347$, $r = 173$, $g = 3, h = 11 \in \mathbb{F}_p^*$. Let $n_S = 3$. Determine l_t and l_h for the values $(u_1, v_1) = (1, 1)$, $(u_2, v_2) = (13, 17)$. What is the smallest of i for which $x_{2i} = x_i$?

Exercise 14.2.5. Repeat Exercise 14.2.4 for $g = 11, h = 3$ $(u_1, v_1) = (4, 7)$ and $(u_2, v_2) = (23, 5)$.

[3] Apparently this algorithm first appears in print in Knuth [343], but is credited there to Floyd.

Algorithm 16 The rho algorithm

```
INPUT: $g, h \in G$
OUTPUT: $a$ such that $h = g^a$, or $\perp$
Choose randomly the function walk as explained above
$x_1 = g, a_1 = 1, b_1 = 0$
$(x_2, a_2, b_2) = \mathsf{walk}(x_1, a_1, b_1)$
while $(x_1 \neq x_2)$ do
    $(x_1, a_1, b_1) = \mathsf{walk}(x_1, a_1, b_1)$
    $(x_2, a_2, b_2) = \mathsf{walk}(\mathsf{walk}(x_2, a_2, b_2))$
end while
if $b_1 \equiv b_2 \pmod{r}$ then
    return $\perp$
else
    return $(a_2 - a_1)(b_1 - b_2)^{-1} \pmod{r}$
end if
```

The smallest index i such that $x_{2i} = x_i$ is called the **epact**. The expected value of the epact is conjectured to be approximately $0.823\sqrt{\pi r/2}$; see Heuristic 14.2.9.

Example 14.2.6. Let $p = 809$ and consider $g = 89$ which has prime order 101 in $\mathbb{F}_p^*$. Let $h = 799$ which lies in the subgroup generated by g.

Let $n_S = 4$. To define $S(g)$ write g in the range $1 \leq g < 809$, represent this integer in its usual binary expansion and then reduce modulo 4. Choose $(u_1, v_1) = (37, 34), (u_2, v_2) = (71, 69), (u_3, v_3) = (76, 18)$ so that $g_1 = 343, g_2 = 676, g_3 = 627$. One computes the table of values (x_i, a_i, b_i) as follows:

i	x_i	a_i	b_i	$S(x_i)$
1	89	1	0	1
2	594	38	34	2
3	280	8	2	0
4	736	16	4	0
5	475	32	8	3
6	113	7	26	1
7	736	44	60	0

It follows that $l_t = 4$ and $l_h = 3$ and so the first collision detected by Floyd's method is $x_6 = x_{12}$. We leave as an exercise to verify that the discrete logarithm in this case is 50.

Exercise 14.2.7. Let $p = 569$ and let $g = 262$ and $h = 5$ which can be checked to have order 71 modulo p. Use the rho algorithm to compute the discrete logarithm of h to the base g modulo p.

Exercise 14.2.8. One can simplify Definition 14.2.1 and equation (14.4) by replacing g_j by either g^{u_j} or h^{v_j} (independently for each j). Show that this saves one modular addition in each iteration of the algorithm. Explain why this optimisation should not affect the success of the algorithm, as long as the walk uses all values for $S(x_i)$ with roughly equal probability.

Algorithm 16 always terminates, but there are several things that can go wrong:

- The value $(b_1 - b_2)$ may not be invertible modulo r.

Hence, we can only expect to prove that the algorithm succeeds with a certain probability (extremely close to 1).

- The cycle may be very long (as big as r) in which case the algorithm is slower than brute force search.

  Hence, we can only expect to prove an expected running time for the algorithm. We recall that the expected running time in this case is the average, over all choices for the function walk, of the worst-case running time of the algorithm over all problem instances.

Note that the algorithm always halts, but it may fail to output a solution to the DLP. Hence, this is a Monte Carlo algorithm.

It is an open problem to give a rigorous running time analysis for the rho algorithm. Instead it is traditional to make the heuristic assumption that the pseudorandom walk defined above behaves sufficiently close to a random walk. The rest of this section is devoted to showing that the heuristic running time of the rho algorithm with Floyd cycle finding is $(3.093 + o(1))\sqrt{r}$ group operations (asymptotic as $r \to \infty$).

Before stating a precise heuristic we determine an approximation to the expected value of the epact in the case of a truly random walk.[4]

Heuristic 14.2.9. Let x_i be a sequence of elements of a group G of order r obtained as above by iterating a random function $f : G \to G$. Then the expected value of the epact (i.e., the smallest positive integer i such that $x_{2i} = x_i$) is approximately $(\zeta(2)/2)\sqrt{\pi r/2} \approx 0.823\sqrt{\pi r/2}$, where $\zeta(2) = \pi^2/6$ is the value of the Riemann zeta function at 2.

Argument: Fix a specific sequence x_i and let l be the length of the rho, so that x_{l+1} lies in $\{x_1, x_2, \dots, x_l\}$. Since x_{l+1} can be any one of the x_i, the cycle length l_h can be any value $1 \le l_h \le l$ and each possibility happens with probability $1/l$.

The epact is the smallest multiple of l_h which is bigger than $l_t = l - l_h$. Hence, if $l/2 \le l_h \le l$ then the epact is l_h, if $l/3 \le l_h < l/2$ then the epact is $2l_h$. In general, if $l/(k+1) \le l_h < l/k$ then the epact is kl_h. The largest possible value of the epact is $l-1$, which occurs when $l_h = 1$.

The expected value of the epact when the rho has length l is therefore

$$E_l = \sum_{k=1}^{\infty} \sum_{l_h=1}^{l} k l_h P_l(k, l_h)$$

where $P_l(k, l_h)$ is the probability that kl_h is the epact. By the above discussion, $P(k, l_h) = 1/l$ if $l/(k+1) \le l_h < l/k$ or $(k, l_h) = (1, l)$ and zero otherwise. Hence

$$E_l = \tfrac{1}{l} \sum_{k=1}^{l-1} k \sum_{\substack{l/(k+1) \le l_h < l/k \\ \text{or } (k,l_h)=(1,l)}} l_h$$

Approximating the inner sum as $\frac{1}{2}\left((l/k)^2 - (l/(k+1))^2\right)$ gives

$$E_l \approx \tfrac{l}{2} \sum_{k=1}^{\infty} k \left(\tfrac{1}{k^2} - \tfrac{1}{(k+1)^2} \right).$$

[4] I thank John Pollard for showing me this argument.

Now, $k(1/k^2 - 1/(k+1)^2) = 1/k - 1/(k+1) + 1/(k+1)^2$ and

$$\sum_{k=1}^{\infty}(1/k - 1/(k+1)) = 1 \qquad \text{and} \qquad \sum_{k=1}^{\infty} 1/(k+1)^2 = \zeta(2) - 1.$$

Hence $E_l \approx l/2(1 + \zeta(2) - 1)$. It is well-known that $\zeta(2) \approx 1.645$. Finally, write $\Pr(e)$ for the probability the epact is e, $\Pr(l)$ for the probability the rho length is l, and $\Pr(e \mid l)$ for the conditional probability that the epact is e given that the rho has length l. The expectation of e is then

$$\begin{aligned} E(e) &= \sum_{e=1}^{\infty} e \Pr(e) = \sum_{e=1}^{\infty} e \sum_{l=1}^{\infty} \Pr(e \mid l) \Pr(l) \\ &= \sum_{l=1}^{\infty} \Pr(l) \left(\sum_{e=1}^{\infty} e \Pr(e \mid l) \right) \\ &= \sum_{l=1}^{\infty} \Pr(l) E_l \approx (\zeta(2)/2) E(l) \end{aligned}$$

which completes the argument. □

We can now give a heuristic analysis of the running time of the algorithm. We make the following assumption, which we believe is reasonable when r is sufficiently large, $n_S > \log(r)$ and when the function walk is chosen at random (from the set of all walk functions specified in Section 14.2.1).

Heuristic 14.2.10.

1. The expected value of the epact is $(0.823 + o(1))\sqrt{\pi r/2}$.
2. The value $\sum_{i=l_t}^{l_t+l_h-1} v_{S(x_i)} \pmod{r}$ is uniformly distributed in $\mathbb{Z}/r\mathbb{Z}$.

Theorem. *Let the notation be as above and assume Heuristic 14.2.10. Then the rho algorithm with Floyd cycle finding has expected running time of* $(3.093 + o(1))\sqrt{r}$ *group operations. The probability the algorithm fails is negligible.*

Proof: The number of iterations of the main loop in Algorithm 16 is the epact. By Heuristic 14.2.10 the expected value of the epact is $(0.823 + o(1))\sqrt{\pi r/2}$.

Algorithm 16 performs three calls to the function walk in each iteration. Each call to walk results in one group operation and two additions modulo r (we ignore these additions as they cost significantly less than a group operation). Hence the expected number of group operations is $3(0.823 + o(1))\sqrt{\pi r/2} \approx (3.093 + o(1))\sqrt{r}$ as claimed.

The algorithm fails only if $b_{2i} \equiv b_i \pmod{r}$. We have $g^{a_{l_t}} h^{b_{l_t}} = g^{a_{l_t+l_h}} h^{b_{l_t+l_h}}$ from which it follows that $a_{l_t+l_h} = a_{l_t} + u$, $b_{l_t+l_h} = b_{l_t} + v$ where $g^u h^v = 1$. Precisely, $v \equiv b_{l_t+l_h} - b_{l_t} \equiv \sum_{i=l_t}^{l_t+l_h-1} v_{S(x_i)} \pmod{r}$.

Write $i = l_t + i'$ for some $0 \le i' < l_h$ and $b_i = b_{l_t} + w$. Assume $l_h \ge 2$ (the probability that $l_h = 1$ is negligible). Then $2i = l_t + x l_h + i'$ for some integer $1 \le x < (l_t + 2l_h)/l_h < r$ and so $b_{2i} = b_{l_t} + xv + w$. It follows that $b_{2i} \equiv b_i \pmod{r}$ if and only if $r \mid v$.

According to Heuristic 14.2.10 the value v is uniformly distributed in $\mathbb{Z}/r\mathbb{Z}$ and so the probability it is zero is $1/r$, which is a negligible quantity in the input size of the problem.
□

14.2.3 Other Cycle Finding Methods

Floyd cycle finding is not a very efficient way to find cycles. Though any cycle finding method requires computing at least $l_t + l_h$ group operations, Floyd's method needs on average $2.47(l_t + l_h)$ group operations (2.47 is three times the expected value of the epact). Also, the "slower" sequence x_i is visiting group elements which have already been computed during the walk of the "faster" sequence x_{2i}. Brent [98] has given an improved cycle finding method[5] that still only requires storage for two group elements but which requires fewer group operations. Montgomery has given an improvement to Brent's method in [436].

One can do even better by using more storage, as was shown by Sedgewick, Szymanski and Yao [535], Schnorr and Lenstra [527] (also see Teske [603]) and Nivasch [467]. The rho algorithm using Nivasch cycle finding has the optimal expected running time of $\sqrt{\pi r/2} \approx 1.253\sqrt{r}$ group operations and is expected to require polynomial storage.

Finally, a very efficient way to find cycles is to use distinguished points. More importantly, distinguished points allow us to think about the rho method in a different way and this leads to a version of the algorithm that can be parallelised. We discuss this in the next section. Hence, in practice one always uses distinguished points.

14.2.4 Distinguished Points and Pollard Rho

The idea of using distinguished points in search problems apparently goes back to Rivest. The first application of this idea to computing discrete logarithms is by van Oorschot and Wiener [473].

Definition 14.2.11. An element $g \in G$ is a **distinguished point** if its binary representation $b(g)$ satisfies some easily checked property. Denote by $\mathcal{D} \subset G$ the set of distinguished points. The probability $\#\mathcal{D}/\#G$ that a uniformly chosen group element is a distinguished point is denoted θ.

A typical example is the following.

Example 14.2.12. Let E be an elliptic curve over $\mathbb{F}_p$. A point $P \in E(\mathbb{F}_p)$ that is not the point at infinity is represented by an x-coordinate $0 \le x_P < p$ and a y-coordinate $0 \le y_P < p$. Let H be a hash function, whose output is interpreted as being in $\mathbb{Z}_{\ge 0}$.

Fix an integer n_D. Define $\mathcal{D}$ to be the points $P \in E(\mathbb{F}_p)$ such that the n_D least significant bits of $H(x_P)$ are zero. Note that $\mathcal{O}_E \notin \mathcal{D}$. In other words

$$\mathcal{D} = \{P = (x_P, y_P) \in E(\mathbb{F}_p) : H(x_P) \equiv 0 \pmod{2^{n_D}} \text{ where } 0 \le x_P < p\}.$$

Then $\theta \approx 1/2^{n_D}$.

The rho algorithm with distinguished points is as follows. First, choose integers $0 \le a_1, b_1 < r$ uniformly and independently at random, compute the group element $x_1 = g^{a_1}h^{b_1}$ and run the usual deterministic pseudorandom walk until a distinguished point $x_n = g^{a_n}h^{b_n}$ is found. Store (x_n, a_n, b_n) in some easily searched data structure (searchable on x_n). Then choose a fresh randomly chosen group element $x_1 = g^{a_1}h^{b_1}$ and repeat. Eventually two walks will visit the same group element, in which case their paths will continue to the same distinguished point. Once a distinguished group element is found twice then the DLP can be solved with high probability.

Exercise 14.2.13. Write down pseudocode for this algorithm.

[5] This was originally developed to speed up the Pollard rho factoring algorithm.

We stress the most significant difference between this method and the method of the previous section: the previous method had one long walk with a tail and a cycle, whereas the new method has many short walks. Note that this algorithm does not require self-collisions in the walk and so there is no ρ shape anymore; the word “rho” in the name of the algorithm is therefore a historical artifact, not an intuition about how the algorithm works.

Note that, since the group is finite, collisions must eventually occur, and so the algorithm halts. But the algorithm may fail to solve the DLP (with low probability). Hence, this is a Monte Carlo algorithm.

In the analysis we assume that we are sampling group elements (we sometimes call them “points”) uniformly and independently at random. It is important to determine the expected number of steps before landing on a distinguished point.

Lemma 14.2.14. *Let θ be the probability that a randomly chosen group element is a distinguished point. Then*

1. *The probability that one chooses α/θ group elements, none of which are distinguished, is approximately $e^{-\alpha}$ when $1/\theta$ is large.*

2. *The expected number of group elements to choose before getting a distinguished point is $1/\theta$.*

3. *If one has already chosen i group elements, none of which are distinguished, then the expected number of group elements to further choose before getting a distinguished point is $1/\theta$.*

Proof: The probability that i chosen group elements are not distinguished is $(1-\theta)^i$. So the probability of choosing α/θ points, none of which are distinguished, is

$$(1-\theta)^{\alpha/\theta} \le \left(e^{-\theta}\right)^{\alpha/\theta} = e^{-\alpha}.$$

The second statement is the standard formula for the expected value of a geometric random variable, see Example A.14.1.

For the final statement[6], suppose one has already sampled i points without finding a distinguished point. Since the trials are independent, the probability of choosing a further j points which are not distinguished remains $(1-\theta)^j$. Hence the expected number of extra points to be chosen is still $1/\theta$. □

We now make the following assumption. We believe this is reasonable when r is sufficiently large, $n_S > \log(r)$, distinguished points are sufficiently common and specified using a good hash function (and hence, $\mathcal{D}$ is well distributed), $\theta > \log(r)/\sqrt{r}$ and when the function walk is chosen at random.

Heuristic 14.2.15.

1. Walks reach a distinguished point in significantly fewer than $\sqrt{r}$ steps (in other words, there are no cycles in the walks and walks are not excessively longer than $1/\theta$).[7]
2. The expected number of group elements sampled before a collision is $\sqrt{\pi r/2}$.

[6] This is the “apparent paradox” mentioned in footnote 7 of [473].

[7] More realistically, one could assume that only a negligibly small proportion of the walks fall into a cycle before hitting a distinguished point.

Theorem 14.2.16. *Let the notation be as above and assume Heuristic 14.2.15. Then the rho algorithm with distinguished points has expected running time of* $(\sqrt{\pi/2} + o(1))\sqrt{r} \approx (1.253 + o(1))\sqrt{r}$ *group operations. The probability the algorithm fails is negligible.*

Proof: Heuristic 14.2.15 states there are no cycles or "wasted" walks (in the sense that their steps do not contribute to potential collisions). Hence, before the first collision, after N steps of the algorithm we have visited N group elements. By Heuristic 14.2.15, the expected number of group elements to be sampled before the first collision is $\sqrt{\pi r/2}$. The collision is not detected until walks hit a distinguished point, which adds a further $2/\theta$ to the number of steps. Hence, the total number of steps (calls to the function walk) in the algorithm is $\sqrt{\pi r/2} + 2/\theta$. Since $2/\theta < 2\sqrt{r}/\log(r) = o(1)\sqrt{r}$, the result follows.

Let $x = g^{a_i}h^{b_i} = g^{a_j}h^{b_j}$ be the collision. Since the starting values $g^{a_0}h^{b_0}$ are chosen uniformly and independently at random, the values b_i and b_j are uniformly and independently random. It follows that $b_i \equiv b_j \pmod{r}$ with probability $1/r$, which is a negligible quantity in the input size of the problem. □

Exercise 14.2.17. Show that if $\theta = \log(r)/\sqrt{r}$ then the expected storage of the rho algorithm, assuming it takes $O(\sqrt{r})$ steps, is $O(\log(r))$ group elements (which is typically $O(\log(r)^2)$ bits).

Exercise 14.2.18. The algorithm requires storing a triple (x_n, a_n, b_n) for each distinguished point. Give some strategies to reduce the number of bits that need to be stored.

Exercise 14.2.19. Let $G = \langle g_1, g_2 \rangle$ be a group of order r^2 and exponent r. Design a rho algorithm that, on input $h \in G$ outputs (a_1, a_2) such that $h = g_1^{a_1} g_2^{a_2}$. Determine the complexity of this algorithm.

Exercise 14.2.20. Show that the Pollard rho algorithm with distinguished points has better average-case running time than the baby-step-giant-step algorithm (see Exercises 13.3.3 and 13.3.4).

Exercise 14.2.21. Explain why taking $\mathcal{D} = G$ (i.e., all group elements distinguished) leads to an algorithm that is much slower than the baby-step-giant-step algorithm.

Suppose one is given $g, h_1, \dots, h_L$ (where $1 < L < r^{1/4}$) and is asked to find all a_i for $1 \le i \le L$ such that $h_i = g^{a_i}$. Kuhn and Struik [355] propose and analyse a method to solve all L instances of the DLP, using Pollard rho with distinguished points, in roughly $\sqrt{2rL}$ group operations. A crucial trick, attributed to Silverman and Stapleton, is that once the i-th DLP is known one can re-write all distinguished points $g^a h_i^b$ in the form $g^{a'}$. As noted by Hitchcock, Montague, Carter and Dawson [287] one must be careful to choose a random walk function that does not depend on the elements h_i (however, the random starting points do depend on the h_i).

Exercise 14.2.22. Write down pseudocode for the Kuhn-Struik algorithm for solving L instances of the DLP, and explain why the algorithm works.

Section 14.2.5 explains why the rho algorithm with distinguished points can be easily parallelised. That section also discusses a number of practical issues relating to the use of distinguished points.

Cheon, Hong and Kim [133] sped up Pollard rho in $\mathbb{F}_p^*$ by using a "look ahead" strategy; essentially they determine in which partition the next value of the walk lies, without performing a full group operation. A similar idea for elliptic curves has been used by Bos, Kaihara and Kleinjung [89].

14.2.5 Towards a Rigorous Analysis of Pollard Rho

Theorem 14.2.16 is not satisfying since Heuristic 14.2.15 is essentially equivalent to the statement "the rho algorithm has expected running time $(1 + o(1))\sqrt{\pi r/2}$ group operations". The reason for stating the heuristic is to clarify exactly what properties of the pseudorandom walk are required. The reason for believing Heuristic 14.2.15 is that experiments with the rho algorithm (see Section 14.4.3) confirm the estimate for the running time.

Since the algorithm is fundamental to an understanding of elliptic curve cryptography (and torus/trace methods) it is natural to demand a complete and rigorous treatment of it. Such an analysis is not yet known, but in this section we mention some partial results on the problem. The methods used to obtain the results are beyond the scope of this book, so we do not give full details. Note that all existing results are in an idealised model where the selection function S is a random function.

We stress that, in practice, the algorithm behaves as the heuristics predict. Furthermore, from a cryptographic point of view, it is sufficient for the task of determining key sizes to have a lower bound on the running time of the algorithm. Hence, in practice, the absence of proved running time is not necessarily a serious issue.

The main results for the original rho walk (with $n_S = 3$) are due to Horwitz and Venkatesan [294], Miller and Venkatesan [426], and Kim, Montenegro, Peres and Tetali [337, 336]. The basic idea is to define the **rho graph**, which is a directed graph with vertex set $\langle g \rangle$ and an edge from x_1 to x_2 if x_2 is the next step of the walk when at x_1. Fix an integer n. Define the distribution $\mathcal{D}_n$ on $\langle g \rangle$ obtained by choosing uniformly at random $x_1 \in \langle g \rangle$, running the walk for n steps, and recording the final point in the walk. The crucial property to study is the **mixing time** which, informally, is the smallest integer n such that $\mathcal{D}_n$ is "sufficiently close" to the uniform distribution. For these results, the squaring operation in the original walk is crucial. We state the main result of Miller and Venkatesan [426] below.

Theorem 14.2.23. *(Theorem 1.1 of [426]) Fix $\epsilon > 0$. Then the rho algorithm using the original rho walk with $n_S = 3$ finds a collision in $O_\epsilon(\sqrt{r}\log(r)^3)$ group operations with probability at least $1 - \epsilon$, where the probability is taken over all partitions of $\langle g \rangle$ into three sets S_1, S_2 and S_3. The notation O_ϵ means that the implicit constant in the O depends on ϵ.*

Kim, Montenegro, Peres and Tetali improved this result in [336] to the desired $O_\epsilon(\sqrt{r})$ group operations. Note that all these works leave the implied constant in the O unspecified.

Note that the idealised model of S being a random function is not implementable with constant (or even polynomial) storage. Hence, these results cannot be applied to the algorithm presented above, since our selection functions S are very far from uniformly chosen over all possible partitions of the set $\langle g \rangle$. The number of possible partitions of $\langle g \rangle$ into three subsets of equal size is (for convenience suppose that $3 \mid r$)

$$\binom{r}{r/3}\binom{2r/3}{r/3}$$

which, using $\binom{a}{b} \geq (a/b)^b$, is at least $6^{r/3}$. On the other hand, a selection function parameterised by a "key" of $c\log_2(r)$ bits (e.g., a selection function obtained from a keyed hash function) only leads to r^c different partitions.

Sattler and Schnorr [513] and Teske [604] have considered the additive rho walk. One key feature of their work is to discuss the effect of the number of partitions n_S. Sattler and Schnorr show (subject to a conjecture) that if $n_S \geq 8$ then the expected running time for the rho algorithm is $c\sqrt{\pi r/2}$ group operations for an explicit constant c. Teske shows, using results of Hildebrand, that the additive walk should approximate the uniform distribution after fewer than $\sqrt{r}$ steps once $n_S \geq 6$. She recommends using the additive walk with $n_S \geq 20$ and, when this is done, conjectures that the expected cycle length is $\leq 1.3\sqrt{r}$ (compared with the theoretical $\approx 1.2533\sqrt{r}$).

Further motivation for using large n_S is given by Brent and Pollard [99], Arney and Bender [13] and Blackburn and Murphy [59]. They present heuristic arguments that the expected cycle length when using n_S partitions is $\sqrt{c_{n_S}\pi r/2}$ where $c_{n_S} = n_S/(n_S - 1)$. This heuristic is supported by the experimental results of Teske [604]. Let $G = \langle g \rangle$. Their analysis considers the directed graph formed from iterating the function $\mathsf{walk} : G \to G$ (i.e., the graph with vertex set G and an edge from g to $\mathsf{walk}(g)$). Then, for a randomly chosen graph of this type, $n_S/(n_S - 1)$ is the variance of the in-degree for this graph, which is the same as the expected value of $n(x) = \#\{y \in G : y \neq x, \ \mathsf{walk}(y) = \mathsf{walk}(x)\}$.

Finally, when using equivalence classes (see Section 14.4) there are further advantages in taking n_S to be large.

14.3 Distributed Pollard Rho

In this section we explain how the Pollard rho algorithm can be parallelised. Rather than a parallel computing model we consider a **distributed computing** model. In this model there is a **server** and $N_P \geq 1$ **clients** (we also refer to the clients as **processors**). There is no shared storage or direct communication between the clients. Instead, the server can send messages to clients and each client can send messages to the server. In general we prefer to minimise the amount of communication between server and clients.[8]

To solve an instance of the discrete logarithm problem the server will activate a number of clients, providing each with its own individual initial data. The clients will run the rho pseudorandom walk and occasionally send data back to the server. Eventually the server will have collected enough information to solve the problem, in which case it sends all clients a termination instruction. The rho algorithm with distinguished points can very naturally be used in this setting.

The best one can expect for any distributed computation is a linear speedup compared with the serial case (since if the overall total work in the distributed case was less than the serial case then this would lead to a faster algorithm in the serial case). In other words, with N_P clients we hope to achieve a running time proportional to $\sqrt{r}/N_P$.

14.3.1 The Algorithm and its Heuristic Analysis

All processors perform the same pseudorandom walk $(x_{i+1}, a_{i+1}, b_{i+1}) = \mathsf{walk}(x_i, a_i, b_i)$ as in Section 14.2.1, but each processor starts from a different random starting point. Whenever a processor hits a distinguished point then it sends the triple (x_i, a_i, b_i) to the server and re-starts its walk at a new random point (x_0, a_0, b_0). If one processor ever visits a point visited by another processor then the walks from that point agree and both walks end at the same distinguished point. When the server receives two triples (x, a, b) and (x, a', b') for the same group element x but with $b \not\equiv b' \pmod{r}$ then it has $g^a h^b = g^{a'} h^{b'}$ and can solve the DLP as in the serial (i.e., non-parallel) case. The server therefore computes the discrete logarithm problem and sends a terminate signal to all processors.

[8] There are numerous examples of such distributed computation over the internet. Two notable examples are the Great Internet Mersenne Primes Search (GIMPS) and the Search for Extraterrestrial Intelligence (SETI). One observes that the former search has been more successful than the latter.

Pseudocode for both server and clients are given by Algorithms 17 and 18. By design, if the algorithm halts then the answer is correct.

Algorithm 17 The distributed rho algorithm: Server side

```
INPUT: $g, h \in G$
OUTPUT: $c$ such that $h = g^c$
 1: Randomly choose a walk function walk$(x, a, b)$
 2: Initialise an easily searched structure $L$ (sorted list, binary tree etc) to be empty
 3: Start all processors with the function walk
 4: while DLP not solved do
 5:     Receive triples $(x, a, b)$ from clients and insert into $L$
 6:     if first coordinate of new triple $(x, a, b)$ matches existing triple $(x, a', b')$ then
 7:         if $b' \not\equiv b \pmod{r}$ then
 8:             Send terminate signal to all clients
 9:             return $(a - a')(b' - b)^{-1} \pmod{r}$
10:         end if
11:     end if
12: end while
```

Algorithm 18 The distributed rho algorithm: Client side

```
INPUT: $g, h \in G$, function walk
 1: while terminate signal not received do
 2:     Choose uniformly at random $0 \le a, b < r$
 3:     Set $x = g^a h^b$
 4:     while $x \notin \mathcal{D}$ do
 5:         $(x, a, b) = $ walk$(x, a, b)$
 6:     end while
 7:     Send $(x, a, b)$ to server
 8: end while
```

We now analyse the performance of this algorithm. To get a clean result we assume that no client ever crashes, that communications between server and client are perfectly reliable, that all clients have the same computational efficiency and are running continuously (in other words, each processor computes the same number of group operations in any given time period).

It is appropriate to ignore the computation performed by the server and instead to focus on the number of group operations performed by each client running Algorithm 18. Each execution of the function walk(x, a, b) involves a single group operation. We must also count the number of group operations performed in line 3 of Algorithm 18; though this term is negligible if walks are long on average (i.e., if $\mathcal{D}$ is a sufficiently small subset of G).

It is an open problem to give a rigorous analysis of the distributed rho method. Hence, we make the following heuristic assumption. We believe this assumption is reasonable when r is sufficiently large, n_S is sufficiently large, $\log(r)/\sqrt{r} < \theta$, the set $\mathcal{D}$ of distinguished points is determined by a good hash function, the number N_P of clients is sufficiently small (e.g., $N_P < \theta\sqrt{\pi r/2}/\log(r)$, see Exercise 14.3.3), the function walk is chosen at random.

Heuristic 14.3.1.

1. The expected number of group elements to be sampled before the same element is sampled twice is $\sqrt{\pi r/2}$.
2. Walks reach a distinguished point in significantly fewer than $\sqrt{r}/N_P$ steps (in other words, there are no cycles in the walks and walks are not excessively long). More realistically, one could assume that only a negligible proportion of the walks fall into a cycle before hitting a distinguished point.

Theorem 14.3.2. *Let the notation be as above, in particular, let N_P be the (fixed, independent of r) number of clients. Let θ the probability that a group element is a distinguished point and suppose $\log(r)/\sqrt{r} < \theta$. Assume Heuristic 14.3.1 and the above assumptions about the the reliability and equal power of the processors hold. Then the expected number of group operations performed by each client of the distributed rho method is $(1 + 2\log(r)\theta)\sqrt{\pi r/2}/N_P + 1/\theta$ group operations. This is $(\sqrt{\pi/2}/N_P + o(1))\sqrt{r}$ group operations when $\theta < 1/\log(r)^2$. The storage requirement on the server is $\theta\sqrt{\pi r/2} + N_P$ points.*

Proof: Heuristic 14.3.1 states that we expect to sample $\sqrt{\pi r/2}$ group elements in total before a collision arises. Since this work is distributed over N_P clients of equal speed it follows that each client is expected to call the function walk about $\sqrt{\pi r/2}/N_P$ times. The total number of group operations is therefore $\sqrt{\pi r/2}/N_P$ plus $2\log(r)\theta\sqrt{\pi r/2}/N_P$ for the work of line 3 of Algorithm 18. The server will not detect the collision until the second client hits a distinguished point, which is expected to take $1/\theta$ further steps by the heuristic (part 3 of Lemma 14.2.14). Hence each client needs to run an expected $\sqrt{\pi r/2}/N_P + 1/\theta$ steps of the walk.

Of course, a collision $g^a h^b = g^{a'} h^{b'}$ can be useless in the sense that $b' \equiv b \pmod{r}$. A collision implies $a' + cb' \equiv a + cb \pmod{r}$ where $h = g^c$; there are r such pairs (a', b') for each pair (a, b). Since each walk starts with uniformly random values (a_0, b_0) it follows that the values (a, b) are uniformly distributed over the r possibilities. Hence the probability of a collision being useless is $1/r$ and the expected number of collisions required is 1.

Each processor runs for $\sqrt{\pi r/2}/N_P + 1/\theta$ steps and therefore is expected to send $\theta\sqrt{\pi r/2}/N_P + 1$ distinguished points in its lifetime. The total number of points to store is therefore $\theta\sqrt{\pi r/2} + N_P$. □

Exercise 14.2.17 shows that the complexity in the case $N_P = 1$ can be taken to be $(1 + o(1))\sqrt{\pi r/2}$ group operations with polynomial storage.

Exercise 14.3.3. When distributing the algorithm it is important to ensure that, with very high probability, each processor finds at least one distinguished point in less than its total expected running time. Show that this will be the case if $1/\theta \le \sqrt{\pi r/2}/\left(N_P \log(r)\right)$.

Schulte-Geers [533] analyses the choice of θ and shows that Heuristics 14.2.15 and 14.3.1 are not valid asymptotically if $\theta = o(1/\sqrt{r})$ as $r \to \infty$ (for example, walks in this situation are more likely to fall into a cycle than to hit a distinguished point). In any case, since each processor only travels a distance of $\sqrt{\pi r/2}/N_P$ it follows we should take $\theta > N_P/\sqrt{r}$. In practice one tends to determine the available storage first (say, c group elements where $c > 10^9$) and to set $\theta = c/\sqrt{\pi r/2}$ so that the total number of distinguished points visited is expected to be c. The results of [533] validate this approach. In particular, it is extremely unlikely that there is a self-collision (and hence a cycle) before hitting a distinguished point.

14.4 Speeding up the Rho Algorithm using Equivalence Classes

Gallant, Lambert and Vanstone [232] and Wiener and Zuccherato [632] showed that one can speed up the rho method in certain cases by defining the pseudorandom walk not on the group $\langle g \rangle$ but on a set of equivalence classes. This is essentially the same thing as working in an algebraic group quotient instead of the algebraic group.

Suppose there is an equivalence relation on $\langle g \rangle$. Denote by $\overline{x}$ the equivalence class of $x \in \langle g \rangle$. Let N_C be the size of a generic equivalence class. We require the following properties:

1. One can define a unique representative $\hat{x}$ of each equivalence class $\overline{x}$.
2. Given (x_i, a_i, b_i) such that $x_i = g^{a_i} h^{b_i}$ then one can efficiently compute $(\hat{x}_i, \hat{a}_i, \hat{b}_i)$ such that $\hat{x}_i = g^{\hat{a}_i} h^{\hat{b}_i}$.

We give some examples in Section 14.4.1 below.

One can implement the rho algorithm on equivalence classes by defining a pseudorandom walk function $\mathsf{walk}(x_i, a_i, b_i)$ as in Definition 14.2.1. More precisely, set $x_1 = g, a_1 = 1, b_1 = 0$ and define the sequence x_i by (this is the "original walk")

$$x_{i+1} = f(x_i) = \begin{cases} \hat{x}_i^2 & \text{if } S(\hat{x}_i) = 0 \\ \hat{x}_i g_j & \text{if } S(\hat{x}_i) = j,\ j \in \{1, \ldots, n_S - 1\} \end{cases} \tag{14.6}$$

where the selection function S and the values $g_j = g^{u_j} h^{v_j}$ are as in Definition 14.2.1. When using distinguished points one defines an equivalence class to be distinguished if the unique equivalence class representative has the distinguished property.

There is a very serious problem with cycles that we do not discuss yet; See Section 14.4.2 for the details.

Exercise 14.4.1. Write down the formulae for updating the values a_i and b_i in the function walk.

Exercise 14.4.2. Write pseudocode for the distributed rho method on equivalence classes.

Theorem 14.4.3. *Let G be a group and $g \in G$ of order r. Suppose there is an equivalence relation on $\langle g \rangle$ as above. Let N_C be the generic size of an equivalence class. Let C_1 be the number of bit operations to perform a group operation in $\langle g \rangle$ and C_2 the number of bit operations to compute a unique equivalence class representative $\hat{x}_i$ (and to compute $\hat{a}_i, \hat{b}_i$).*

Consider the rho algorithm as above (ignoring the possibility of useless cycles, see Section 14.4.2 below). Under a heuristic assumption for equivalence classes analogous to Heuristic 14.2.15 the expected time to solve the discrete logarithm problem is

$$\left(\sqrt{\frac{\pi}{2N_C}} + o(1) \right) \sqrt{r}\ (C_1 + C_2)$$

bit operations. As usual, this becomes $(\sqrt{\pi/2N_C} + o(1))\sqrt{r}/N_P(C_1 + C_2)$ bit operations per client when using N_P processors of equal computational power.

Exercise 14.4.4. Prove this theorem.

Theorem 14.4.3 assumes a perfect random walk. For walks defined on n_S partitions of the set of equivalence classes it is shown in Appendix B of [25] (also see Section 2.2 of [91]) that one predicts a slightly improved constant than the usual factor $c_{n_S} = n_S/(n_S - 1)$ mentioned at the end of Section 14.2.5.

We mention a potential "paradox" with this idea. In general, computing a unique equivalence class representative involves listing all elements of the equivalence class, and hence needs $\tilde{O}(N_C)$ bit operations. Hence, naively, the running time is $\tilde{O}(\sqrt{N_C \pi r/2})$ bit operations, which is worse than doing the rho algorithm without equivalence classes. However, in practice one only uses this method when $C_2 < C_1$, in which case the speedup can be significant.

14.4.1 Examples of Equivalence Classes

We now give some examples of useful equivalence relations on some algebraic groups.

Example 14.4.5. For a group G with efficiently computable inverse (e.g., elliptic curves $E(\mathbb{F}_q)$ or algebraic tori $\mathbb{T}_n$ with $n > 1$ (e.g., see Section 6.3)) one can define the equivalence relation $x \equiv x^{-1}$. We have $N_C = 2$ (though note that some elements, namely the identity and elements of order 2, are equal to their inverse so these classes have size 1). If $x_i = g^{a_i}h^{b_i}$ then clearly $x^{-1} = g^{-a_i}h^{-b_i}$. One defines a unique representative $\hat{x}$ for the equivalence class by, for example, imposing a lexicographical ordering on the binary representation of the elements in the class.

We can generalise this example as follows.

Example 14.4.6. Let G be an algebraic group over $\mathbb{F}_q$ with an automorphism group $\mathrm{Aut}(G)$ of size N_C (see examples in Sections 9.4 and 11.3.3). Suppose that for $g \in G$ of order r one has $\psi(g) \in \langle g \rangle$ for each $\psi \in \mathrm{Aut}(G)$. Furthermore, assume that for each $\psi \in \mathrm{Aut}(G)$ one can efficiently compute the eigenvalue $\lambda_\psi \in \mathbb{Z}$ such that $\psi(g) = g^{\lambda_\psi}$. Then for $x \in G$ one can define $\overline{x} = \{\psi(x) : \psi \in \mathrm{Aut}(G)\}$.

Again, one defines $\hat{x}$ by listing the elements of $\overline{x}$ as bitstrings and choosing the first one under lexicographical ordering.

Another important class of examples comes from orbits under the Frobenius map.

Example 14.4.7. Let G be an algebraic group defined over $\mathbb{F}_q$ but with group considered over $\mathbb{F}_{q^d}$ (for examples see Sections 11.3.2 and 11.3.3). Let π_q be the q-power Frobenius map on $G(\mathbb{F}_{q^d})$. Let $g \in G(\mathbb{F}_{q^d})$ and suppose that $\pi_q(g) = g^\lambda \in \langle g \rangle$ for some known $\lambda \in \mathbb{Z}$.

Define the equivalence relation on $G(\mathbb{F}_{q^d})$ so that the equivalence class of $x \in G(\mathbb{F}_{q^d})$ is the set $\overline{x} = \{\pi_q^i(x) : 0 \le i < d\}$. We assume that, for elements x of interest, $\overline{x} \subseteq \langle g \rangle$. Then $N_C = d$, though there can be elements defined over proper subfields for which the equivalence class is smaller.

If one uses a normal basis for $\mathbb{F}_{q^d}$ over $\mathbb{F}_q$ then one can efficiently compute the elements $\pi_q^i(x)$ and select a unique representative of each equivalence class using a lexicographical ordering of binary strings.

Example 14.4.8. For some groups (e.g., Koblitz elliptic curves $E/\mathbb{F}_2$ considered as a group over $\mathbb{F}_{2^m}$; see Exercise 9.10.11) we can combine both equivalence classes above. Let m be prime, $\#E(\mathbb{F}_{2^m}) = hr$ for some small cofactor h, and $P \in E(\mathbb{F}_{2^m})$ of order r. Then $\pi_2(P) \in \langle P \rangle$ and we define the equivalence class $\overline{P} = \{\pm\pi_2^i(P) : 0 \le i < m\}$ of size $2m$. Since m is odd, this class can be considered as the orbit of P under the map $-\pi_2$. The distributed rho algorithm on equivalence classes for such curves is expected to require approximately $\sqrt{\pi 2^m/(4m)}$ group operations.

14.4.2 Dealing with Cycles

One problem that can arise is walks that fall into a cycle before they reach a distinguished point. We call these **useless cycles**.

Exercise 14.4.9. Suppose the equivalence relation is such that $x \equiv x^{-1}$. Fix $x_i = \hat{x}_i$ and let $x_{i+1} = \hat{x}_i g$. Suppose $\hat{x}_{i+1} = x_{i+1}^{-1}$ and that $S(\hat{x}_{i+1}) = S(\hat{x}_i)$. Show that $x_{i+2} \equiv x_i$ and so there is a cycle of order 2. Suppose the equivalence classes generically have size N_C. Show, under the assumptions that the function S is perfectly random and that $\hat{x}$ is a randomly chosen element of the equivalence class, that the probability that a randomly chosen x_i leads to a cycle of order 2 is $1/(N_C n_S)$.

A theoretical discussion of cycles was given in [232] and by Duursma, Gaudry and Morain [186]. An obvious way to reduce the probability of cycles is to take n_S to be very large compared with the average length $1/\theta$ of walks. However, as argued by Bos, Kleinjung and Lenstra [91], large values for n_S can lead to slower algorithms (for example, due to the fact that the precomputed steps do not all fit in cache memory). Hence, as Exercise 14.4.9 shows, useless cycles will be regularly encountered in the algorithm. There are several possible ways to deal with this issue. One approach is to use a "look-ahead" technique to avoid falling in 2-cycles. Another approach is to detect small cycles (e.g., by storing a fixed number of previous values of the walk or, at regular intervals, using a cycle-finding algorithm for a small number of steps) and to design a well-defined exit strategy for short cycles; Gallant, Lambert and Vanstone call this **collapsing the cycle**; see Section 6 of [232]. To collapse a cycle one must be able to determine a well-defined element in it; from there one can take a step (different to the steps used in the cycle from that point) or use squaring to exit the cycle. All these methods require small amounts of extra computation and storage, though Bernstein, Lange and Schwabe [56] argue that the additional overhead can be made negligible. We refer to [56, 91] for further discussion of these issues.

Gallant, Lambert and Vanstone [232] presented a different walk that does not, in general, lead to short cycles. Let G be an algebraic group with an efficiently computable endomorphism ψ of order m (i.e., $\psi^m = \psi \circ \cdots \circ \psi$ is the identity map). Let $g \in G$ of order r be such that $\psi(g) = g^\lambda$ so that $\psi(x) = x^\lambda$ for all $x \in \langle g \rangle$. Define the equivalence classes $\overline{x} = \{\psi^j(x) : 0 \le j < m\}$. We define a pseudorandom sequence $x_i = g^{a_i} h^{b_i}$ by using $\hat{x}$ to select an endomorphism $(1 + \psi^j)$ and then acting on x_i with this map. More precisely, j is some function of $\hat{x}$ (e.g., the function S in Section 14.2.1) and

$$x_{i+1} = (1 + \psi^j)x_i = x_i \psi^j(x_i) = x_i^{1+\lambda^j}$$

(the above equation looks more plausible when the group operation is written additively: $x_{i+1} = x_i + \psi^j(x_i) = (1 + \lambda^j)x_i$). One can check that the map is well-defined on equivalence classes and that $x_{i+1} = g^{a_{i+1}} h^{b_{i+1}}$ where $a_{i+1} = (1 + \lambda^j)a_i \pmod{r}$ and $b_{i+1} = (1 + \lambda^j)b_i \pmod{r}$.

We stress that this approach still requires finding a unique representative of each equivalence class in order to define the steps of the walk in a well-defined way. Hence, one can still use distinguished points by defining a class to be distinguished if its representative is distinguished. One suggestion, originally due to Harley, is to use the Hamming weight of the x-coordinate to derive the selection function.

One drawback of the Gallant, Lambert, Vanstone idea is that there is less flexibility in the design of the pseudorandom walk.

Exercise 14.4.10. Generalise the Gallant-Lambert-Vanstone walk to use $(c + \psi^j)$ for any $c \in \mathbb{Z}$. Why do we prefer to only use $c = 1$?

Exercise 14.4.11. Show that taking $n_S = \log(r)$ means the total overhead from handling cycles is $o(\sqrt{r})$, while the additional storage (group elements for the random walks) is $O(\log(r))$ group elements.

Exercise 14.4.11 together with Exercise 14.2.17 shows that (as long as computing equivalence class representatives is fast) one can solve the discrete logarithm problem using equivalence classes of generic size N_C in $(1 + o(1))\sqrt{\pi r/(2N_C)}$ group operations and $O(\log(r))$ group elements storage.

14.4.3 Practical Experience with the Distributed Rho Algorithm

Real computations are not as simple as the idealised analysis above: one doesn't know in advance how many clients will volunteer for the computation; not all clients have the same performance or reliability; clients may decide to withdraw from the computation at any time; the communications between client and server may be unreliable etc. Hence, in practice one needs to choose the distinguished points to be sufficiently common that even the weakest client in the computation can hit a distinguished point within a reasonable time (perhaps after just one or two days). This may mean that the stronger clients are finding many distinguished points every hour.

The largest discrete logarithm problems solved using the distributed rho method are mainly the Certicom challenge elliptic curve discrete logarithm problems. The current records are for the groups $E(\mathbb{F}_p)$ where $p \approx 2^{108} + 2^{107}$ (by a team coordinated by Chris Monico in 2002) and where $p = (2^{128} - 3)/76439 \approx 2^{111} + 2^{110}$ (by Bos, Kaihara and Montgomery in 2009) and for $E(\mathbb{F}_{2^{109}})$ (again by Monico's team in 2004). None of these computations used the equivalence class $\{P, -P\}$.

We briefly summarise the parameters used for these large computations. For the 2002 result the curve $E(\mathbb{F}_p)$ has prime order so $r \approx 2^{108} + 2^{107}$. The number of processors was over 10,000 and they used $\theta = 2^{-29}$. The number of distinguished points found was 68228567 which is roughly 1.32 times the expected number $\theta\sqrt{\pi r/2}$ of points to be collected. Hence, this computation was unlucky in that it ran about 1.3 times longer than the expected time. The computation ran for about 18 months.

The 2004 result is for a curve over $\mathbb{F}_{2^{109}}$ with group order $2r$ where $r \approx 2^{108}$. The computation used roughly 2000 processors, $\theta = 2^{-30}$ and the number of distinguished points found was 16531676. This is about 0.79 times the expected number $\theta\sqrt{\pi 2^{108}/2}$. This computation took about 17 months.

The computation by Bos, Kaihara and Montgomery [90] was innovative in that the work was done using a cluster of 200 computer game consoles. The random walk used $n_S = 16$ and $\theta = 1/2^{24}$. The total number of group operations performed was 8.5×10^{16} (which is 1.02 times the expected value) and 5×10^9 distinguished points were stored.

Exercise 14.4.12. Verify that the parameters above satisfy the requirements that θ is much larger than $1/\sqrt{r}$ and N_P is much smaller than $\theta\sqrt{r}$.

There is a close fit between the actual running time for these examples and the theoretical estimates. This is evidence that the heuristic analysis of the running time is not too far from the performance in practice.

14.5 The Kangaroo Method

This algorithm is designed for the case where the discrete logarithm is known to lie in a short interval. Suppose $g \in G$ has order r and that $h = g^a$ where a lies in a short interval

$b \leq a < b + w$ of width w. We assume that the values of b and w are known. Of course, one can solve this problem using the rho algorithm, but if w is much smaller than the order of g then this will not necessarily be optimal.

The kangaroo method was originally proposed by Pollard [487]. Van Oorschot and Wiener [473] greatly improved it by using distinguished points. We present the improved version in this section.

For simplicity, compute $h' = hg^{-b}$. Then $h' \equiv g^x \pmod{p}$ where $0 \leq x < w$. Hence, there is no loss of generality by assuming that $b = 0$. Thus, from now on our problem is: Given g, h, w to find a such that $h = g^a$ and $0 \leq a < w$.

As with the rho method, the kangaroo method relies on a deterministic pseudorandom walk. The steps in the walk are pictured as the "jumps" of the kangaroo, and the group elements visited are the kangaroo's "footprints". The idea, as explained by Pollard, is to "catch a wild kangaroo using a tame kangaroo". The "tame kangaroo" is a sequence $x_i = g^{a_i}$ where a_i is known. The "wild kangaroo" is a sequence $y_j = hg^{b_j}$ where b_j is known. Eventually, a footprint of the tame kangaroo will be the same as a footprint of the wild kangaroo (this is called the "collision"). After this point, the tame and wild footprints are the same.[9] The tame kangaroo lays "traps" at regular intervals (i.e., at distinguished points) and, eventually, the wild kangaroo falls in one of the traps.[10] More precisely, at the first distinguished point after the collision, one finds a_i and b_j such that $g^{a_i} = hg^{b_j}$ and the DLP is solved as $h = g^{a_i - b_j}$.

There are two main differences between the kangaroo method and the rho algorithm.

- Jumps are "small". This is natural since we want to stay within (or at least, not too far outside) the interval.
- When a kangaroo lands on a distinguished point one **continues** the pseudorandom walk (rather than restarting the walk at a new randomly chosen position).

14.5.1 The Pseudorandom Walk

The pseudorandom walk for the kangaroo method has some significant differences to the rho walk: steps in the walk correspond to known small increments in the exponent (in other words, kangaroos make small jumps of known distance in the exponent). We therefore do not include the squaring operation $x_{i+1} = x_i^2$ (as the jumps would be too big) or multiplication by h (we would not know the length of the jump in the exponent). We now describe the walk precisely.

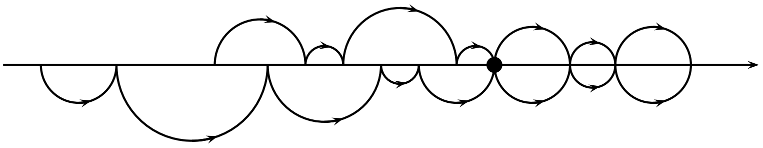
- As in Section 14.2.1 we use a function $S : G \to \{0, \dots, n_S - 1\}$ which partitions G into sets $\mathcal{S}_i = \{g \in G : S(g) = i\}$ of roughly similar size.
- For $0 \leq j < n_S$ choose exponents $1 \leq u_j \leq \sqrt{w}$ Define $m = (\sum_{j=0}^{n_S - 1} u_j)/n_S$ to be the **mean step size**. As explained below we will take $m \approx \sqrt{w}/2$.

  Pollard [487, 488] suggested taking $u_j = 2^j$ as this minimises the chance that two different short sequences of jumps add to the same value. This seems to give good results in practice. An alternative is to choose most of the values u_i to be random and the last few to ensure that m is very close to $c_1\sqrt{w}$.

[9] A collision between two different walks can be drawn in the shape of the letter λ. Hence Pollard also suggested this be called the "lambda method". However, other algorithms (such as the distributed rho method) have collisions between different walks, so this naming is ambiguous. The name "kangaroo method" emphasises the fact that the jumps are small. Hence, as encouraged by Pollard, we do not use the name "lambda method" in this book.

[10] Actually, the wild kangaroo can be in front of the tame kangaroo, in which case it is better to think of each kangaroo trying to catch the other.

Figure 14.1: Kangaroo walk. Tame kangaroo walk pictured above the axis and wild kangaroo walk pictured below. The dot indicates the first collision.



- The pseudorandom walk is a sequence $x_0, x_1, \dots$ of elements of G defined by an initial value x_0 (to be specified later) and the formula

$$x_{i+1} = x_i g_{S(x_i)}.$$

The algorithm is not based on the birthday paradox, but instead on the following observations. Footprints are spaced, on average, distance m apart, so along a region traversed by a kangaroo there is, on average, one footprint in any interval of length m. Now, if a second kangaroo jumps along the same region and if the jumps of the second kangaroo are independent of the jumps from the first kangaroo, then the probability of a collision is roughly $1/m$. Hence, one expects a collision between the two walks after about m steps.

14.5.2 The Kangaroo Algorithm

We need to specify where to start the tame and wild kangaroos, and what the mean step size should be. The wild kangaroo starts at $y_0 = h = g^a$ with $0 \le a < w$. To minimise the distance between the tame and wild kangaroos at the start of the algorithm, we start the tame kangaroo at $x_0 = g^{\lfloor w/2 \rfloor}$, which is the middle of the interval. We take alternate jumps and store the values (x_i, a_i) and (y_i, b_i) as above (i.e., so that $x_i = g^{a_i}$ and $y_i = hg^{b_i}$). Whenever x_i (respectively, y_i) is distinguished we store (x_i, a_i) (resp., (y_i, b_i)) in an easily searched structure. The storage can be reduced by using the ideas of Exercise 14.2.18.

When the same distinguished point is visited twice then we have two entries (x, a) and (x, b) in the structure and so either $hg^a = g^b$ or $g^a = hg^b$. The ambiguity is resolved by seeing which of $a - b$ and $b - a$ lies in the interval (or just testing if $h = g^{a-b}$ or not).

As we will explain in Section 14.5.3, the optimal choice for the mean step size is $m = \sqrt{w}/2$.

Exercise 14.5.1. Write this algorithm in pseudocode.

We visualise the algorithm not in the group G but on a line representing exponents. The tame kangaroo starts at $\lfloor w/2 \rfloor$. The wild kangaroo starts somewhere in the interval $[0, w)$. Kangaroo jumps are small steps to the right. See Figure 14.1 for the picture.

Example 14.5.2. Let $g = 3 \in \mathbb{F}_{263}^*$ which has prime order 131. Let $h = 181 \in \langle g \rangle$ and suppose we are told that $h = g^a$ with $0 \le a < w = 53$. The kangaroo method can be used in this case.

Since $\sqrt{w}/2 \approx 3.64$ it is appropriate to take $n_S = 4$ and choose steps $\{1, 2, 4, 8\}$. The mean step size is 3.75. The function $S(x)$ is $x \pmod 4$ (where elements of $\mathbb{F}_{263}^*$ are represented by integers in the set $\{1, \dots, 262\}$).

The tame kangaroo starts at $(x_1, a_1) = (g^{26}, 26) = (26, 26)$. The sequence of points visited in the walk is listed below. A point is distinguished if its representation as an integer is divisible by 3; the distinguished points are written in bold face in the table.

i	0	1	2	3	4
x_i	26	2	**162**	235	**129**
a_i	26	30	34	38	46
$S(x_i)$	2	2	2	3	1
y_i	181	**51**	**75**	2	**162**
b_i	0	2	10	18	22
$S(y_i)$	1	3	3	2	2

The collision is detected when the distinguished point 162 is visited twice. The solution to the discrete logarithm problem is therefore $34 - 22 = 12$.

Exercise 14.5.3. Using the same parameters as Example 14.5.2, solve the DLP for $h = 78$.

14.5.3 Heuristic Analysis of the Kangaroo Method

The analysis of the algorithm does not rely on the birthday paradox; instead, the mean step size is the crucial quantity. We sketch the basic probabilistic argument now. A more precise analysis is given in Section 14.5.6. The following heuristic assumption seems to be reasonable when w is sufficiently large, $n_S > \log(w)$, distinguished points are sufficiently common and specified using a good hash function (and hence are well distributed), $\theta > \log(w)/\sqrt{w}$ and when the function walk is chosen at random.

Heuristic 14.5.4.

1. Walks reach a distinguished point in significantly fewer than $\sqrt{w}$ steps (in other words, there are no cycles in the walks and walks are not excessively longer than $1/\theta$).
2. The footprints of a kangaroo are uniformly distributed in the region over which it has walked with, on average, one footprint in each interval of length m.
3. The footsteps of tame and wild kangaroos are independent of one another before the time when the walks collide.

Theorem 14.5.5. *Let the notation be as above and assume Heuristic 14.5.4. Then the kangaroo algorithm with distinguished points has average case expected running time of $(2 + o(1))\sqrt{w}$ group operations. The probability the algorithm fails is negligible.*

Proof: We don't know whether the discrete logarithm of h is greater or less than $w/2$. So, rather than speaking of "tame" and "wild" kangaroos we will speak of the "front" and "rear" kangaroos. Since one kangaroo starts in the middle of the interval, the distance between the starting point of the rear kangaroo and the starting point of the front kangaroo is between 0 and $w/2$ and is, on average, $w/4$. Hence, on average, $w/(4m)$ jumps are required for the rear kangaro to pass the starting point of the front kangaroo.

After this point, the rear kangaroo is travelling over a region that has already been jumped over by the front kangaroo. By our heuristic assumption, the footprints of the tame kangaroo are uniformly distributed over the region with, on average, one footprint in each interval of length m. Also, the footprints of the wild kangaroo are independent, and with one footprint in each interval of length m. The probability, at each step, that the wild kangaroo does not land on any of the footprints of the tame kangaroo is therefore heuristically $1 - 1/m$. By exactly the same arguments as Lemma 14.2.14 it follows that the expected number of jumps until a collision is m.

Note that there is a miniscule possibility that the walks never meet (this does not require working in an infinite group, it can even happen in a finite group if the "orbits" of the tame and wild walks are disjoint subsets of the group). If this happens then the algorithm never halts. Since the walk function is chosen at random, the probability of this eventuality is negligible. On the other hand, if the algorithm halts then its result is correct. Hence, this is a Las Vegas algorithm.

The overall number of jumps made by the rear kangaroo until the first collision is therefore, on average, $w/(4m) + m$. One can easily check that this is minimised by taking $m = \sqrt{w}/2$. The kangaroo is also expected to perform a further $1/\theta$ steps to the next distinguished point. Since there are two kangaroos the expected total number of group operations performed is $2\sqrt{w} + 2/\theta = (2 + o(1))\sqrt{w}$. □

This result is proved by Montenegro and Tetali [434] under the assumption that S is a random function and that the distinguished points are well-distributed. Pollard [488] shows it is valid when the $o(1)$ is replaced by ϵ for some $0 \le \epsilon < 0.06$.

Note that the expected distance, on average, travelled by a kangaroo is $w/4+m^2 = w/2$ steps. Hence, since the order of the group is greater than w, we do not expect any self-collisions in the kangaroo walk.

We stress that, as with the rho method, the probability of success is considered over the random choice of pseudorandom walk, not over the space of problem instances. Exercise 14.5.6 considers a different way to optimise the expected running time.

Exercise 14.5.6. Show that, with the above choice of m, the expected number of group operations performed for the worst-case of problem instances is $(3 + o(1))\sqrt{w}$. Determine the optimal choice of m to minimise the expected worst-case running time. What is the expected worst-case complexity?

Exercise 14.5.7. A card trick known as **Kruskal's principle** is as follows. Shuffle a deck of 52 playing cards and deal face up in a row. Define the following walk along the row of cards: If the number of the current card is i then step forward i cards (if the card is a King, Queen or Jack then step 5 cards). The magicican runs this walk (in their mind) from the first card and puts a coin on the last card visited by the walk. The magician invites their audience to choose a number j between 1 and 10, then runs the walk from the j-th card. The magician wins if the walk also lands on the card with the coin. Determine the probability of success of this trick.

Exercise 14.5.8. Show how to use the kangaroo method to solve Exercises 13.3.8, 13.3.10 and 13.3.11 of Chapter 13.

Pollard's original proposal did not use distinguished points and the algorithm only had a fixed probability of success. In contrast, the method we have described keeps on running until it succeeds (indeed, if the DLP is insoluble then the algorithm would never terminate). Van Oorschot and Wiener (see page 12 of [473]) have shown that repeating Pollard's original method until it succeeds leads to a method with expected running time of approximately $3.28\sqrt{w}$ group operations.

Exercise 14.5.9. Suppose one is given $g \in G$ of order r, an integer w, and an instance generator for the discrete logarithm problem that outputs $h = g^a \in G$ such that $0 \le a < w$

according to some known distribution on $\{0, 1, \ldots, w-1\}$. Assume that the distribution is symmetric with mean value $w/2$. How should one modify the kangaroo method to take account of this extra information? What is the running time?

14.5.4 Comparison with the Rho Algorithm

We now consider whether one should use the rho or kangaroo algorithm when solving a general discrete logarithm problem (i.e., where the width w of the interval is equal to, or close to, r). If $w = r$ then the rho method requires roughly $1.25\sqrt{r}$ group operations while the kangaroo method requires roughly $2\sqrt{r}$ group operations. The heuristic assumptions underlying both methods are similar, and in practice they work as well as the theory predicts. Hence, it is clear that the rho method is preferable, unless w is much smaller than r.

Exercise 14.5.10. Determine the interval size below which it is preferable to use the kangaroo algorithm over the rho algorithm.

14.5.5 Using Inversion

Galbraith, Ruprai and Pollard [226] showed that one can improve the kangaroo method by exploiting inversion in the group.[11] Suppose one is given g, h, w and told that $h = g^a$ with $0 \leq a < w$. We also require that the order r of g is odd (this will always be the case, due to the Pohlig-Hellman algorithm). Suppose, for simplicity, that w is even. Replacing h by $hg^{-w/2}$ we have $h = g^a$ with $-w/2 \leq a < w/2$. One can perform a version of the kangaroo method with three kangaroos: One tame kangaroo starting from g^u for an appropriate value of u and two wild kangaroos starting from h and h^{-1} respectively.

The algorithm uses the usual kangaroo walk (with mean step size to be determined later) to generate three sequences $(x_i, a_i), (y_i, b_i), (z_i, c_i)$ such that $x_i = g^{a_i}$, $y_i = hg^{b_i}$ and $z_i = h^{-1}g^{c_i}$. The crucial observation is that a collision between any two sequences leads to a solution to the DLP. For example, if $x_i = y_j$ then $h = g^{a_i - b_j}$ and if $y_i = z_j$ then $hg^{b_i} = h^{-1}g^{c_j}$ and so, since g has odd order r, $h = g^{(c_j - b_i)2^{-1} \pmod{r}}$. The algorithm uses distinguished points to detect a collison. We call this the **three-kangaroo algorithm**.

Exercise 14.5.11. Write down pseudocode for the three-kangaroo algorithm using distinguished points.

We now give a brief heuristic analysis of the three-kangaroo algorithm. Without loss of generality we assume $0 \leq a \leq w/2$ (taking negative a simply swaps h and h^{-1}, so does not affect the running time). The distance between the starting points of the tame and wild kangaroos is $2a$. The distance between the starting points of the tame and right-most wild kangaroo is $|a - u|$. The extreme cases (in the sense that the closest pair of kangaroos are as far apart as possible) are when $2a = u - a$ or when $a = w/2$. Making all these cases equal leads to the equation $2a = u - a = w/2 - u$. Calling this distance l it follows that $w/2 = 5l/2$ and $u = 3w/10$. The average distance between the closest pair of kangaroos is then $w/10$ and the closest pair of kangaroos can be thought of as performing the standard kangaroo method in an interval of length $2w/5$. Following the analysis of the standard kangaroo method it is natural to take the mean step size to be $m = \frac{1}{2}\sqrt{2w/5} = \sqrt{w/10} \approx 0.316\sqrt{w}$. The average-case expected number of group operations (only considering the closest pair of kangaroos) would be $\frac{3}{2}2\sqrt{2w/5} \approx 1.897\sqrt{w}$. A more careful analysis takes into account the possibility of collisions between any pair of kangaroos. We refer to [226]

[11] This research actually grew out of writing this chapter. Sometimes it pays to work slowly.

for the details and merely remark that the correct mean step size is $m \approx 0.375\sqrt{w}$ and the average-case expected number of group operations is approximately $1.818\sqrt{w}$.

Exercise 14.5.12. The distance between $-a$ and a is even, so a natural trick is to use jumps of even length. Since we don't know whether a is even or odd, if this is done we don't know whether to start the tame kangaroo at g^u or g^{u+1}. However, one can consider a variant of the algorithm with two wild kangaroos (one starting from h and one from h^{-1}) and two tame kangaroos (one starting from g^u and one from g^{u+1}) and with jumps of even length. This is called the **four-kangaroo algorithm**. Explain why the correct choice for the mean step size is $m = 0.375\sqrt{2w}$ and why the heuristic average-case expected number of group operations is approximately $1.714\sqrt{w} = \frac{2\sqrt{2}}{3}1.818\sqrt{w}$.

Galbraith, Pollard and Ruprai [226] have combined the idea of Exercise 14.5.12 and the Gaudry-Schost algorithm (see Section 14.7) to obtain an algorithm for the discrete logarithm problem in an interval of length w that performs $(1.660 + o(1))\sqrt{w}$ group operations.

14.5.6 Towards a Rigorous Analysis of the Kangaroo Method

Montenegro and Tetali [434] have analysed the kangaroo method using jumps which are powers of 2, under the assumption that the selection function S is random and that the distinguished points are well-distributed. They prove that the average-case expected number of group operations is $(2 + o(1))\sqrt{w}$ group operations. It is beyond the scope of this book to present their methods.

We now present Pollard's analysis of the kangaroo method from his paper [488], though these results have been superseded by [434]. We restrict to the case where the selection function S maps G to $\{0, 1, \dots, n_S - 1\}$ and the kangaroo jumps are taken to be $2^{S(x)}$ (i.e., the set of jumps is $\{1, 2, 4, \dots, 2^{n_S-1}\}$ and the mean of the jumps is $m = (2^{n_S} - 1)/n_S$). We assume $n_S > 2$. Pollard argues in [488] that if one only uses two jumps $\{1, 2^n\}$ (for some n) then the best one can hope for is an algorithm with running time $O(w^{2/3})$ group operations.

Pollard also makes the usual assumption that S is a truly random function.

As always we visualise the kangaroos in terms of their exponents, and so we study a pseudorandom walk on $\mathbb{Z}$. The tame kangaroo starts at w. The wild kangaroo starts somewhere in $[0, w)$. We begin the analysis when the wild kangaroo first lands at a point $\geq w$. Let $w + i$ be the first wild kangaroo footprint $\geq w$. Define $q(i)$ to be the probability (over all possible starting positions for the wild kangaroo) that this first footstep is at $w + i$. Clearly $q(i) = 0$ when $i \geq 2^{n_S-1}$. The wild kangaroo footprints are chosen uniformly at random with mean m, hence $q(0) = 1/m$. For $i > 0$ then only jumps of length $> i$ could be useful, so the probability is

$$q(i) = \#\{0 \leq j < n_S : 2^j > i\}/mn_S.$$

To summarise $q(1) = (n_S - 1)/mn_S, q(2) = (n_S - 2)/mn_S$ and for $i > 2$, $q(i) = (n_S - 1 - \lfloor \log_2(i) \rfloor)/mn_S$.

We now want to analyse how many further steps the wild kangaroo makes before landing on a footprint of the tame kangaroo. We abstract the problem to the following: Suppose the front kangaroo is at i and the rear kangaroo is at 0 and run the pseudorandom walk. Define $F(i)$ to be the expected number of steps made by the front kangaroo to the collision and $B(i)$ the expected number of steps made by the rear kangaroo to the collision.

We can extend the functions to F and B to $i = 0$ by taking a truly random and independent step from $\{1, 2, 4, \dots, 2^{n_S-1}\}$ (i.e., not using the deterministic pseudorandom walk function).

We can now obtain formulae relating the functions $F(i)$ and $B(i)$. Consider one jump by the rear kangaroo. Suppose the jump has distance s where $s < i$. Then the rear kangaroo remains the rear kangaroo, but the front kangaroo is now only $i - s$ ahead. If $F(i-s) = n_1$ and $B(i-s) = n_2$ then we have $F(i) = n_1$ and $B(i) = 1 + n_2$. On the other hand, suppose the jump has distance $s \geq i$. Then the front and rear kangaroo swap roles and the front kangaroo is now $s - i$ ahead. We have $B(i) = 1 + F(s-i)$ and $F(i) = B(s-i)$. Since the steps are chosen uniformly with probability $1/n_S$ we get

$$F(i) = \tfrac{1}{n_S}\left(\sum_{j=0,2^j<i}^{n_S-1} F(i-2^j) + \sum_{j=0,2^j\geq i}^{n_S-1} B(2^j - i) \right)$$

and

$$B(i) = 1 + \tfrac{1}{n_S}\left(\sum_{j=0,2^j<i}^{n_S-1} B(i-2^j) + \sum_{j=0,2^j\geq i}^{n_S-1} F(2^j - i) \right)$$

Pollard then considers the expected value of the number of steps of the wild kangaroo to a collision, namely

$$\sum_{i=1}^{2^{(n_S-1)}-1} q(i)F(i)$$

which we write as $mC(n_S)$ for some $C(n_S) \in \mathbb{R}$. In [488] one finds numerical data for $C(n_S)$ which suggest that it is between 1 and 1.06 when $n_S \geq 12$. Pollard also conjectures that $\lim_{n_S\to\infty} C(n_S) = 1$.

Given an interval of size w one chooses n_S such that the mean $m = (2^{n_S} - 1)/n_S$ is as close as possible to $\sqrt{w}/2$. One runs the tame Kangaroo, starting at w, for $mC(n_S)$ steps and sets the trap. The wild kangaroo is expected to need $w/2m$ steps to pass the start of the tame kangaroo followed by $mC(n_S)$ steps to fall into the trap. Hence, the expected number of group operations for the kangaroo algorithm (for a random function S) is

$$w/2m + 2mC(n_S).$$

Taking $m = \sqrt{w}/2$ gives expected running time

$$(1 + C(n_S))\sqrt{w}$$

group operations.

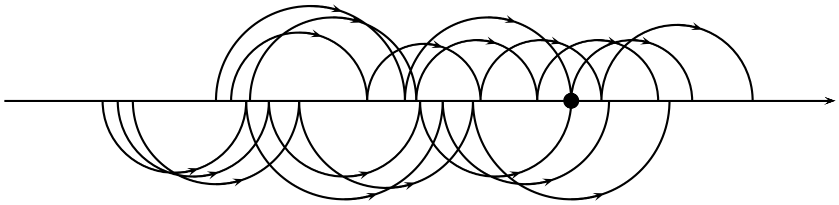
In practice one would slightly adjust the jumps $\{1, 2, 4, \dots, 2^{n_S-1}\}$ (while hoping that this does not significantly change the value of $C(n_S)$) to arrange that $m = \sqrt{w/C(n_S)}/2$.

14.6 Distributed Kangaroo Algorithm

Let N_P be the number of processors or clients. A naive way to parallelise the the kangaroo algorithm is to divide the interval $[0, w)$ into N_P sub-intervals of size w/N_P and then run the kangaroo algorithm in parallel on each sub-interval. This gives an algorithm with running time $O(\sqrt{w/N_P})$ group operations per client, which is not a linear speedup.

Since we are using distinguished points one should be able to do better. But the kangaroo method is not as straightforward to parallelise as the rho method (a good exercise is to stop reading now and think about it for a few minutes). The solution is to use a herd of $N_P/2$ tame kangaroos and a herd of $N_P/2$ wild kangaroos. These are super-kangaroos in the sense that they take much bigger jumps (roughly $N_P/2$ times

Figure 14.2: Distributed kangaroo walk (van Oorschot and Wiener version). The herd of tame kangaroos is pictured above the axis and the herd of wild kangaroos is pictured below. The dot marks the collision.



longer) than in the serial case. The goal is to have a collision between one of the wild kangaroos and one of the tame kangaroos. We imagine that both herds are setting traps, each trying to catch a kangaroo from the other herd (regrettably, they may sometimes catch one of their own kind).

When a kangaroo lands on a distinguished point one continues the pseudorandom walk (rather than restarting the walk at a new randomly chosen position). In other words, the herds march ever onwards with an occasional individual hitting a distinguished point and sending information back to the server. See Figure 14.2 for a picture of the herds in action.

There are two versions of the distributed algorithm, one by van Oorschot and Wiener [473] and another by Pollard [488]. The difference is how they handle the possibility of collisions between kangaroos of the same herd. The former has a mechanism to deal with this, which we will explain later. The latter paper elegantly ensures that there will not be collisions between individuals of the same herd.

14.6.1 Van Oorschot and Wiener Version

We first present the algorithm of van Oorschot and Wiener. The herd of tame kangaroos starts around the midpoint of the interval $[0, w)$, and the kangaroos are spaced a (small) distance s apart (as always, we describe kangaroos by their exponent). Similarly, the wild kangaroos start near $a = \log_g(h)$, again spaced a distance s apart. As we will explain later, the mean step size of the jumps should be $m \approx N_P\sqrt{w}/4$.

Here $\mathsf{walk}(x_i, a_i)$ is the function which returns $x_{i+1} = x_i g_{S(x_i)}$ and $a_{i+1} = a_i + u_{S(x_i)}$. Each client has a variable type which takes the value 'tame' or 'wild'.

If there is a collision between two kangaroos of the same herd then it will eventually be detected when the second one lands on the same distinguished point as the first. In [473] it is suggested that in this case the server should instruct the second kangaroo to take a jump of random length so that it no longer follows the path of the front kangaroo. Note that Teske [606] has shown that the expected number of collisions within the same herd is 2, so this issue can probably be ignored in practice.

We now give a very brief heuristic analysis of the running time. The following assumption seems to be reasonable when w is sufficiently large, n_S is sufficiently large, $\log(w)/\sqrt{w} < \theta$, the set $\mathcal{D}$ of distinguished points is determined by a good hash function, the number N_P of clients is sufficiently small (e.g., $N_P < \theta\sqrt{\pi r/2}/\log(r)$, see Exercise 14.3.3), the spacing s is independent of the steps in the random walk and is sufficiently large, the function walk is chosen at random.

Algorithm 19 The distributed kangaroo algorithm (van Oorschot and Wiener version): Server side

```
INPUT: $g, h \in G$, interval length $w$, number of clients $N_P$
OUTPUT: $a$ such that $h = g^a$
Choose $n_S$, a random function $S : G \to \{0, \dots, n_S - 1\}$, $m = N_P\sqrt{w}/4$, jumps
    $\{u_0, \dots, u_{n_S-1}\}$ with mean $m$, spacing $s$
for $i = 1$ to $N_P/2$ do                              ▷ Start $N_P/2$ tame kangaroo clients
    Set $a_i = \lfloor w/2 \rfloor + is$
    Initiate client on $(g^{a_i}, a_i, \text{'tame'})$ with function walk
end for
for $j = 1$ to $N_P/2$ do                              ▷ Start $N_P/2$ wild kangaroo clients
    Set $a_j = js$
    Initiate client on $(hg^{a_j}, a_j, \text{'wild'})$ with function walk
end for
Initialise an easily sorted structure $L$ (sorted list, binary tree etc) to be empty
while DLP not solved do
    Receive triples $(x_i, a_i, \mathsf{type}_i)$ from clients and insert into $L$
    if first coordinate of new triple $(x, a_2, \mathsf{type}_2)$ matches existing triple $(x, a_1, \mathsf{type}_1)$
        then
        if $\mathsf{type}_2 = \mathsf{type}_1$ then
            Send message to the sender of $(x, a_2, \mathsf{type}_2)$ to take a random jump
        else
            Send terminate signal to all clients
            if $\mathsf{type}_1$ ='tame' then
                return $(a_1 - a_2) \pmod{r}$
            else
                return $(a_2 - a_1) \pmod{r}$
            end if
        end if
    end if
end while
```

Algorithm 20 The distributed kangaroo algorithm (van Oorschot and Wiener version): Client side

```
INPUT: $(x_1, a_1, \mathsf{type}) \in G \times \mathbb{Z}/r\mathbb{Z}$, function walk
while terminate signal not received do
    $(x_1, a_1) = \mathsf{walk}(x_1, a_1)$
    if $x_1 \in \mathcal{D}$ then
        Send $(x_1, a_1, \mathsf{type})$ to server
        if Receive jump instruction then
            Choose random $1 < u < 2m$ (where $m$ is the mean step size)
            Set $a_1 = a_1 + u$, $x_1 = x_1 g^u$
        end if
    end if
end while
```

Heuristic 14.6.1.

1. Walks reach a distinguished point in significantly fewer than $\sqrt{w}$ steps (in other words, there are no cycles in the walks and walks are not excessively longer than $1/\theta$).
2. When two kangaroos with mean step size m walk over the same interval, the expected number of group elements sampled before a collision is m.
3. Walks of kangaroos in the same herd are independent.[12]

Theorem 14.6.2. *Let N_P be the number of clients (fixed, independent of w). Assume Heuristic 14.6.1 and that all clients are reliable and have the same computing power. The average-case expected number of group operations performed by the distributed kangaroo method for each client is $(2 + o(1))\sqrt{w}/N_P$.*

Proof: Since we don't know where the wild kangaroo is, we speak of the front herd and the rear herd. The distance (in the exponent) between the front herd and the rear herd is, on average, $w/4$. So it takes $w/(4m)$ steps for the rear herd to reach the starting point of the front herd.

We now consider the footsteps of the rear herd in the region already visited by the front herd of kangaroos. Assuming the $N_P/2$ kangaroos of the front herd are independent, the region already covered by these kangaroos is expected to have $N_P/2$ footprints in each interval of length m. Hence, under our heuristic assumptions, the probability that a random footprint of one of the rear kangaroos lands on a footprint of one of the front kangaroos is $N_P/(2m)$. Since there are $N_P/2$ rear kangaroos, all mutually independent, the probability of one of the rear kangaroos landing on a tame footprint is $N_P^2/(4m)$. By the heuristic assumption, the expected number of footprints to be made before a collision occurs is $4m/N_P^2$.

Finally, the collision will not be detected until a distinguished point is visited. Hence, one expects a further $1/\theta$ steps to be made.

The expected number of group operations made by each client in the average case is therefore $w/(4m) + 4m/N_P^2 + 1/\theta$. Ignoring the $1/\theta$ term, this expression is minimised by taking $m = N_P\sqrt{w}/4$. The result follows. □

The remarks made in Section 14.3.1 about parallelisation (for example, Exercise 14.3.3) apply equally for the distributed kangaroo algorithm.

Exercise 14.6.3. The above analysis is optimised for the average-case running time. Determine the mean step size to optimise the worst-case expected running time. Show that the heuristic optimal running time is $(3 + o(1))\sqrt{w}/N_P$ group operations.

Exercise 14.6.4. Give distributed versions of the three-kangaroo and four-kangaroo algorithms of Section 14.5.5.

14.6.2 Pollard Version

Pollard's version reduces the computation to essentially a collection of serial versions, but in a clever way so that a linear speed-up is still obtained. One merit of this approach is

[12] This assumption is very strong, and indeed is false in general (since there is a chance that walks collide). The assumption is used for only two purposes. First, to "amplify" the second assumption in the heuristic from any pair of kangaroos to the level of herds. Second, to allow us to ignore collisions between kangaroos in the same herd (Teske, in Section 7 of [606], has argued that such collisions are rare). One could replace the assumption of independence by these two consequences.

that the analysis of the serial kangaroo algorithm can be applied; we no longer need the strong heuristic assumption that kangaroos in the same herd are mutually independent.

Let N_P be the number of processors and suppose we can write $N_P = U + V$ where $\gcd(U, V) = 1$ and $U, V \approx N_P/2$. The number of tame kangaroos is U and the number of wild kangaroos is V. The (super) kangaroos perform the usual pseudorandom walk with steps $\{UVu_0, \dots, UVu_{n-1}\}$ having mean $m \approx N_P\sqrt{w}/4$ (this is UV times the mean step size for solving the DLP in an interval of length $w/UV \approx 4w/N_P^2$). As usual we choose either $u_j \approx 2^j$ or else random values between 0 and $2m/UV$.

The U tame kangaroos start at

$$g^{\lfloor w/2 \rfloor + iV}$$

for $0 \le i < U$. The V wild kangaroos start at hg^{jU} for $0 \le j < V$. Each kangaroo then uses the pseudorandom walk to generate a sequence of values (x_n, a_n) where $x_n = g^{a_n}$ or $x_n = hg^{a_n}$. Whenever a distinguished point is hit the kangaroo sends data to the server and continues the same walk.

Lemma 14.6.5. *Suppose the walks do not cover the whole group, i.e., $0 \le a_n < r$. Then there is no collision between two tame kangaroos or two wild kangaroos. There is a unique pair of tame and wild kangaroos who can collide.*

Proof: Each element of the sequence generated by the ith tame kangaroo is of the form

$$g^{\lfloor w/2 \rfloor + iV + lUV}$$

for some $l \in \mathbb{Z}$. To have a collision between two different tame kangaroos one would need

$$\lfloor w/2 \rfloor + i_1V + l_1UV = \lfloor w/2 \rfloor + i_2V + l_2UV$$

and reducing modulo U implies $i_1 \equiv i_2 \pmod{U}$ which is a contradiction. To summarise, the values a_n for the tame kangaroos all lie in disjoint equivalence classes modulo U. A similar argument shows that wild kangaroos do not collide.

Finally, if $h = g^a$ then $i = (\lfloor w/2 \rfloor - a)V^{-1} \pmod{U}$ and $j = (a - \lfloor w/2 \rfloor)U^{-1} \pmod{V}$ are the unique pair of indices such that the ith tame kangaroo and the jth wild kangaroo can collide. □

The analysis of the algorithm therefore reduces to the serial case, since we have one tame kangaroo and one wild kangaroo who can collide. This makes the heuristic analysis simple and immediate.

Theorem 14.6.6. *Let the notation be as above. Assume Heuristic 14.5.4 and that all clients are reliable and have the same computational power. Then the average-case expected running time for each client is $(1 + o(1))\sqrt{w/UV} = (2 + o(1))\sqrt{w}/N_P$ group operations.*

Proof: The action is now constrained to an equivalence class modulo UV, so the clients behave like the serial kangaroo method in an interval of size w/UV (see Exercise 14.5.8 for reducing a DLP in a congruence class to a DLP in a smaller interval). The mean step size is therefore $m \approx UV\sqrt{w/UV}/2 \approx N_P\sqrt{w}/4$. Applying Theorem 14.5.5 gives the result. □

14.6.3 Comparison of the Two Versions

Both versions of the distributed kangaroo method have the same heuristic running time of $(2 + o(1))\sqrt{w}/N_P$ group operations.[13] So which is to be preferred in practice? The

[13] Though the analysis by van Oorschot and Wiener needs the stronger assumption that the kangaroos in the same herd are mutually independent.

answer depends on the context of the computation. For genuine parallel computation in a closed system (e.g., using special-purpose hardware) then either could be used.

In distributed environments then both methods have drawbacks. For example, the van Oorschot-Wiener method needs a communication from server to client in response to uploads of distinguished point information (the "take a random jump" instruction); though Teske [606] has remarked that this can probably be ignored.

More significantly, both methods require knowing the number N_P of processors at the start of the computation, since this value is used to specify the mean step size. This causes problems if a large number of new clients join the computation after it has begun.

With the van Oorschot and Wiener method, if further clients want to join the computation after it has begun, then they can be easily added (half the new clients tame and half wild) by starting them at further shifts from the original starting points of the herds. With Pollard's method it is less clear how to add new clients. Even worse, since only one pair of "lucky" clients has the potential to solve the problem, if either of them crashes or withdraws from the computation then the problem will not be solved. As mentioned in Section 14.4.3 these are serious issues which do arise in practice.

On the other hand, these issues can be resolved by over-estimating N_P and by issuing clients with fresh problem instances once they have produced sufficiently many distinguished points from their current instance. Note that this also requires communication from server to client.

14.7 The Gaudry-Schost Algorithm

Gaudry and Schost [249] give a different approach to solving discrete logarithm problems using pseudorandom walks. As we see in Exercise 14.7.6, this method is slower than the rho method when applied to the whole group. However, the approach leads to low-storage algorithms for the multi-dimensional discrete logarithm problems (see Definition 13.5.1); and the discrete logarithm problem in an interval using equivalence classes. This is interesting since, for both problems, it is not known how to adapt the rho or kangaroo methods to give a low-memory algorithm with the desired running time.

The basic idea of the Gaudry-Schost algorithm is as follows. One has pseudorandom walks in two (or more) subsets of the group such that a collision between walks of different types leads to a solution to the discrete logarithm problem. The sets are smaller than the whole group, but they must overlap (otherwise, there is no chance of a collision). Typically, one of the sets is called a "tame set" and the other a "wild set". The pseudorandom walks are deterministic, so that when two walks collide they continue along the same path until they hit a distinguished point and stop. Data from distinguished points is held in an easily searched database held by the server. After reaching a distinguished point, the walks re-start at a freshly chosen point.

14.7.1 Two-Dimensional Discrete Logarithm Problem

Suppose we are given $g_1, g_2, h \in G$ and $N \in \mathbb{N}$ (where we assume N is even) and asked to find integers $0 \le a_1, a_2 < N$ such that $h = g_1^{a_1} g_2^{a_2}$. Note that the size of the solution space is N^2, so we seek a low-storage algorithm with number of group operations proportional to N. The basic Gaudry-Schost algorithm for this problem is as follows.

Define the tame set

$$T = \{(x, y) \in \mathbb{Z}^2 : 0 \le x, y < N\}$$

and the wild set

$$W = (a_1 - N/2, a_2 - N/2) + T = \{(a_1 - N/2 + x, a_2 - N/2 + y) \in \mathbb{Z}^2 : 0 \le x, y < N\}.$$

In other words, T and W are $N \times N$ boxes centered on $(N/2-1, N/2-1)$ and (a_1, a_2) respectively. It follows that $\#W = \#T = N^2$ and if $(a_1, a_2) = (N/2-1, N/2-1)$ then $T = W$, otherwise $T \cap W$ is a proper non-empty subset of T.

Define a pseudorandom walk as follows: First choose $n_S > \log(N)$ random pairs of integers $-M < m_i, n_i < M$ where M is an integer to be chosen later (typically, $M \approx N/(1000\log(N))$) and precompute elements of the form $w_i = g_1^{m_i} g_2^{n_i}$ for $0 \le i < n_S$. Then choose a selection function $S : G \to \{0, 1, \dots, n_S - 1\}$. The walk is given by the function

$$\mathsf{walk}(g, x, y) = (g w_{S(g)}, x + m_{S(g)}, y + n_{S(g)}).$$

Tame walks are started at $(g_1^x g_2^y, x, y)$ for random elements $(x, y) \in T$ and wild walks are started at $(h g_1^{x-N/2+1} g_2^{y-N/2+1}, x - N/2 + 1, y - N/2 + 1)$ for random elements $(x, y) \in T$. Walks proceed by iterating the function walk until a distinguished element of G is visited; at which time the data (g, x, y), together with the type of walk, is stored in a central database. When a distinguished point is visited, the walk is re-started at a uniformly chosen group element (this is like the rho method, but different from the behaviour of kangaroos). Once two walks of different types visit the same distinguished group element we have a collision of the form

$$g_1^x g_2^y = h g_1^{x'} g_2^{y'}$$

and the two-dimensional DLP is solved.

Exercise 14.7.1. Write pseudocode, for both the client and server, for the distributed Gaudry-Schost algorithm.

Exercise 14.7.2. Explain why the algorithm can be modified to omit storing the the type of walk in the database. Show that the methods of Exercise 14.2.18 to reduce storage can also be used in the Gaudry-Schost algorithm.

Exercise 14.7.3. What modifications are required to solve the problem $h = g_1^{a_1} g_2^{a_2}$ such that $0 \le a_1 < N_1$ and $0 \le a_2 < N_2$ for $0 < N_1 < N_2$?

An important practical consideration is that walks will sometimes go outside the tame or wild regions. One might think that this issue can be solved by simply taking the values x and y into account and altering the walk when close to the boundary, but then the crucial property of the walk function (that once two walks collide, they follow the same path) would be lost. By taking distinguished points to be quite common (i.e., increasing the storage) and making M relatively small one can minimise the impact of this problem. Hence, we ignore it in our analysis.

We now briefly explain the heuristic complexity of the algorithm. The key observation is that a collision can only occur in the region where the two sets overlap. Let $A = T \cap W$. If one samples uniformly at random in A, alternately writing elements down on a "tame" and "wild" list, the expected number of samples until the two lists have an element in common is $\sqrt{\pi \# A} + O(1)$ (see, for example, Selivanov [536] or [223]).

The following heuristic assumption seems to be reasonable when N is sufficiently large, $n_S > \log(N)$, distinguished points are sufficiently common and specified using a good hash function (and hence are well-distributed), $\theta > \log(N)/N$, walks are sufficiently "local" that they do not go outside T (respectively, W) but also not too local, and when the function walk is chosen at random.

Heuristic 14.7.4.

1. Walks reach a distinguished point in significantly fewer than N steps (in other words, there are no cycles in the walks and walks are not excessively longer than $1/\theta$).
2. Walks are uniformly distributed in T (respectively, W).

Theorem 14.7.5. *Let the notation be as above, and assume Heuristic 14.7.4. Then the average-case expected number of group operations performed by the Gaudry-Schost algorithm is* $(\sqrt{\pi}(2(2-\sqrt{2}))^2 + o(1))N \approx (2.43 + o(1))N$.

Proof: We first compute $\#(T \cap W)$. When $(a_1, a_2) = (N/2, N/2)$ then $W = T$ and so $\#(T \cap W) = N^2$. In all other cases the intersection is less. The extreme case is when $(a_1, a_2) = (0, 0)$ (similar cases are $(a_1, a_2) = (N-1, N-1)$ etc). Then $W = \{(x, y) \in \mathbb{Z}^2 : -N/2 \le x, y < N/2\}$ and $\#(T \cap W) = N^2/4$. By symmetry it suffices to consider the case $0 \le a_1, a_2 < N/2$ in which case we have $\#(T \cap W) \approx (N/2 + a_1)(N/2 + a_2)$ (here we are approximating the number of integer points in a set by its area).

Let $A = T \cap W$. To sample $\sqrt{\pi \# A}$ elements in A it is necessary to sample $\#T/\#A$ elements in T and W. Hence, the number of group elements to be selected overall is

$$\frac{\#T}{\#A}\big(\sqrt{\pi \# A} + O(1)\big) \;=\; (\#T + o(1))\sqrt{\pi}(\#A)^{-1/2}.$$

The average-case number of group operations is

$$(N^2 + o(1))\sqrt{\pi}\big(\tfrac{2}{N}\big)^2 \int_0^{N/2} \int_0^{N/2} (N-x)^{-1/2}(N-y)^{-1/2} dx dy.$$

Note that

$$\int_0^{N/2} (N-x)^{-1/2} dx \;=\; \sqrt{N}(2-\sqrt{2}).$$

The average-case expected number of group operations is therefore

$$\big(\sqrt{\pi}(2(2-\sqrt{2}))^2 + o(1)\big)N$$

as stated. □

The Gaudry-Schost algorithm has a number of parameters that can be adjusted (such as the type of walks, the sizes of the tame and wild regions etc). This gives it a lot of flexibility and makes it suitable for a wide range of variants of the DLP. Indeed, Galbraith and Ruprai [227] have improved the running time to $(2.36 + o(1))N$ group operations by using smaller tame and wild sets (also, the wild set is a different shape). One drawback is that it is hard to fine-tune all these parameters to get an implementation that achieves the theoretically optimal running time.

Exercise 14.7.6. Determine the complexity of the Gaudry-Schost algorithm for the standard DLP in G, when one takes $T = W = G$.

Exercise 14.7.7. Generalise the Gaudry-Schost algorithm to the n-dimensional DLP (see Definition 13.5.1). What is the heuristic average-case expected number of group operations?

14.7.2 Discrete Logarithm Problem in an Interval using Equivalence Classes

Galbraith and Ruprai [228] used the Gaudry-Schost algorithm to solve the DLP in an interval of length $N < r$ faster than is possible using the kangaroo method when the group has an efficiently computable inverse (e.g., elliptic curves or tori). First, shift the discrete logarithm problem so that it is of the form $h = g^a$ with $-N/2 < a \leq N/2$. Define the equivalence relation $u \equiv u^{-1}$ for $u \in G$ as in Section 14.4 and determine a rule that leads to a unique representative of each equivalence class. Design a pseudorandom walk on the set of equivalence classes. The tame set is the set of equivalence classes coming from elements of the form g^x with $-N/2 < x \leq N/2$. Note that the tame set has $1 + N/2$ elements and every equivalence class $\{g^x, g^{-x}\}$ arises in two ways, except the singleton class $\{1\}$ and the class $\{-N/2, N/2\}$.

A natural choice for the wild set is the set of equivalence classes coming from elements of the form hg^x with $-N/2 < x \leq N/2$. Note that the size of the wild set now depends on the discrete logarithm problem: if $h = g^0 = 1$ then the wild set has $1 + N/2$ elements while if $h = g^{N/2}$ then the wild set has N elements. Even more confusingly, sampling from the wild set by uniformly choosing x does not, in general, lead to uniform sampling from the wild set. This is because the equivalence class $\{hg^x, (hg^x)^{-1}\}$ can arise in either one or two ways, depending on h. To analyse the algorithm it is necessary to use a non-uniform version of the birthday paradox (see, for example, Galbraith and Holmes [223]). The main result of [228] is an algorithm that solves the DLP in heuristic average-case expected $(1.36 + o(1))\sqrt{N}$ group operations.

14.8 Parallel Collision Search in Other Contexts

Van Oorschot and Wiener [473] propose a general method, motivated by Pollard's rho algorithm, for finding collisions of functions using distinguished points and parallelisation. They give applications to cryptanalysis of hash functions and block ciphers that are beyond the scope of this book. But they also give applications of their method for algebraic meet-in-the-middle attacks, so we briefly give the details here.

First we sketch the parallel collision search method. Let $f : \mathcal{S} \to \mathcal{S}$ be a function mapping some set $\mathcal{S}$ of size N to itself. Define a set $\mathcal{D}$ of distinguished points in $\mathcal{S}$. Each client chooses a random starting point $x_1 \in \mathcal{S}$, iterates $x_{n+1} = f(x_n)$ until it hits a distinguished point, and sends (x_1, x_n, n) to the server. The client then restarts with a new random starting point. Eventually the server gets two triples (x_1, x, n) and (x_1', x, n') for the same distinguished point. As long as we don't have a "Robin Hood"[14] (i.e., one walk is a subsequence of another) the server can use the values (x_1, n) and (x_1', n') to efficiently find a collision $f(x) = f(y)$ with $x \neq y$. The expected running time for each client is $\sqrt{\pi N/2}/N_P + 1/\theta$, using the notation of this chapter. The storage requirement depends on the choice of θ.

We now consider the application to meet-in-the-middle attacks. A general meet-in-the-middle attack has two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ and functions $f_i : \mathcal{S}_i \to \mathcal{R}$ for $i = 1, 2$. The goal is to find $a_1 \in \mathcal{S}_1$ and $a_2 \in \mathcal{S}_2$ such that $f_1(a_1) = f_2(a_2)$. The standard solution (as in baby-step-giant-step) is to compute and store all $(f_1(a_1), a_1)$ in an easily searched structure and then test for each $a_2 \in \mathcal{S}_2$ whether $f_2(a_2)$ is in the structure. The running time is $\#\mathcal{S}_1 + \#\mathcal{S}_2$ function evaluations and the storage is proportional to $\#\mathcal{S}_1$.

[14] Robin Hood is a character of English folklore who is expert in archery. His prowess allows him to shoot a second arrow on exactly the same trajectory as the first, so that the second arrow splits the first. Chinese readers may substitute the name Houyi.

The idea of [473] is to phrase this as a collision search problem for a single function f. For simplicity we assume that $\#\mathcal{S}_1 = \#\mathcal{S}_2 = N$. We write $I = \{0, 1, \dots, N-1\}$ and assume one can construct bijective functions $\sigma_i : I \to \mathcal{S}_i$ for $i = 1, 2$. One defines a surjective map

$$\rho : \mathcal{R} \to I \times \{1, 2\}$$

and a set $\mathcal{S} = I \times \{1, 2\}$. Finally, define $f : \mathcal{S} \to \mathcal{S}$ as $f(x, i) = \rho(f_i(\sigma_i(x)))$. Clearly, the desired collision $f_1(a_1) = f_2(a_2)$ can arise from $f(\sigma_1^{-1}(a_1), 1) = f(\sigma_2^{-1}(a_2), 2)$, but collisions can also arise in other ways (for example, due to collisions in ρ). Indeed, since $\#\mathcal{S} = 2N$ one expects there to be roughly $2N$ pairs $(a_1, a_2) \in \mathcal{S}^2$ such that $a_1 \neq a_2$ but $f(a_1) = f(a_2)$. In many applications there is only one collision (van Oorschot and Wiener call it the "golden collision") that actually leads to a solution of the problem. It is therefore necessary to analyse the algorithm carefully to determine the expected time until the problem is solved.

Let N_P be the number of clients and let N_M be the total number of group elements that can be stored on the server. Van Oorschot and Wiener give a heuristic argument that the algorithm finds a useful collision after $2.5\sqrt{(2N)^3/N_M}/N_P$ group operations per client. This is taking $\theta = 2.25\sqrt{N_M/2N}$ for the probability of a distinguished point. We refer to [473] for the details.

14.8.1 The Low Hamming Weight DLP

Recall the low Hamming weight DLP: Given g, h, n, w find x of bit-length n and Hamming weight w such that $h = g^x$. The number of values for x is $M = \binom{n}{w}$ and there is a naive low storage algorithm running in time $\tilde{O}(M)$. We stress that the symbol w here means the Hamming weight; rather than its meaning earlier in this chapter.

Section 13.6 gave baby-step-giant-step algorithms for the low Hamming weight DLP that perform $O(\sqrt{w}\binom{n/2}{w/2})$ group operations. Hence these methods require time and space roughly proportional to $\sqrt{wM}$.

To solve the low Hamming weight DLP using parallel collision search one sets $\mathcal{R} = \langle g \rangle$ and $\mathcal{S}_1, \mathcal{S}_2$ to be sets of integers of binary length $n/2$ and Hamming weight roughly $w/2$. Define the functions $f_1(a) = g^a$ and $f_2(a) = hg^{-2^{n/2}a}$ so that a collision $f_1(a_1) = f_2(a_2)$ solves the problem. Note that there is a unique choice of (a_1, a_2) such that $f_1(a_1) = f_2(a_2)$ but when one uses the construction of van Oorschot and Wiener to get a single function f then there will be many useless collisions in f. We have $N = \#\mathcal{S}_1 = \#\mathcal{S}_2 \approx \binom{n/2}{w/2} \approx \sqrt{M}$ and so get an algorithm whose number of group operations is proportional to $N^{3/2} = M^{3/4}$ yet requires low storage. This is a significant improvement over the naive low-storage method, but still slower than baby-step-giant-step.

Exercise 14.8.1. Write this algorithm in pseudocode and give a more careful analysis of the running time.

It remains an open problem to give a low memory algorithm for the low Hamming weight DLP with complexity proportional to $\sqrt{wM}$ as with the BSGS methods.

14.9 Pollard Rho Factoring Method

This algorithm was proposed in [486] and was the first algorithm invented by Pollard that exploited pseudorandom walks. As more powerful factoring algorithms exist, we keep the presentation brief. For further details see Section 5.6.2 of Stinson [592] or Section 5.2.1 of Crandall and Pomerance [162].

Let N be a composite integer to be factored and let $p \mid N$ be a prime (usually p is the smallest prime divisor of N). We try to find a relation that holds modulo p but not modulo other primes dividing N.

The basic idea of the rho factoring algorithm is to consider the pseudorandom walk $x_1 = 2$ and

$$x_{i+1} = f(x_i) \pmod{N}$$

where the usual choice for $f(x)$ is $x^2 + 1$ (or $f(x) = x^2 + a$ for some small integer a). Consider the values $x_i \pmod{p}$ where $p \mid N$. The sequence $x_i \pmod{p}$ is a pseudorandom sequence of residues modulo p, and so after about $\sqrt{\pi p/2}$ steps we expect there to be indicies i and j such that $x_i \equiv x_j \pmod{p}$. We call this a **collision**. If $x_i \not\equiv x_j \pmod{N}$ then we can split N as $\gcd(x_i - x_j, N)$.

Example 14.9.1. Let $p = 11$. Then the rho iteration modulo p is

$$2,\ 5,\ 4,\ 6,\ 4,\ 6,\ 4,\ \ldots$$

Let $p = 19$. Then the sequence is

$$2,\ 5,\ 7,\ 12,\ 12,\ 12,\ \ldots$$

As with the discrete logarithm algorithms, the walk is deterministic in the sense that a collision leads to a cycle. Let l_t be the length of the tail and l_h be the length of the cycle. Then the first collision is

$$x_{l_t + l_h} \equiv x_{l_t} \pmod{p}.$$

We can use Floyd's cycle finding algorithm to detect the collision. The details are given in Algorithm 21. Note that it is not efficient to compute the gcd in line 5 of the algorithm for each iteration; Pollard [486] gave a solution to reduce the number of gcd computations and Brent [98] gave another.

Algorithm 21 The rho algorithm for factoring

```
INPUT: $N$
OUTPUT: A factor of $N$
1: $x_1 = 2$, $x_2 = f(x_1) \pmod{N}$
2: repeat
3:     $x_1 = f(x_1) \pmod{N}$
4:     $x_2 = f(f(x_2)) \pmod{N}$
5:     $d = \gcd(x_2 - x_1, N)$
6: until $1 < d < N$
7: return $d$
```

We now briefly discuss the complexity of the algorithm. Note that the "algorithm" may not terminate, for example if the length of the cycle and tail are the same for all $p \mid N$ then the gcd will always be either 1 or N. In practice one would stop the algorithm after a certain number of steps and repeat with a different choice of x_1 and/or $f(x)$. Even if it terminates, the length of the cycle of the rho may be very large. Hence, the usual approach is to make the heuristic assumption that the rho pseudorandom walk behaves like a random walk. To have meaningful heuristics one should analyse the algorithm when the function $f(x)$ is randomly chosen from a large set of possible functions.

Note that the rho method is more general than the $p - 1$ method (see Section 12.3), since a random $p \mid N$ is not very likely to be $\sqrt{p}$-smooth.

Theorem 14.9.2. *Let N be composite, not a prime power and not "too smooth". Assume that the Pollard rho walk modulo p behaves like a pseudorandom walk for all $p \mid N$. Then the rho algorithm factors N in $O(N^{1/4}\log(N)^2)$ bit operations.*

Proof: (Sketch) Let p be a prime dividing N such that $p \le \sqrt{N}$. Define the values l_t and l_h corresponding to the sequence $x_i \pmod{p}$. If the walk behaves sufficiently like a random walk then, by the birthday paradox, we will have $l_h, l_t \approx \sqrt{\pi p/8}$. Similarly, for some other prime $q \mid N$ one expects that the walk modulo q has different values l_h and l_t. Hence, after $O(\sqrt{p})$ iterations of the loop one expects to split N. □

Bach [21] has given a rigorous analysis of the rho factoring algorithm. He proves that if $0 \le x, y < N$ are chosen randomly and the iteration is $x_1 = x$, $x_{i+1} = x_i^2 + y$, then the probability of finding the smallest prime factor p of N after k steps is at least $k(k-1)/2p + O(p^{-3/2})$ as p goes to infinity, where the constant in the O depends on k. Bach's method cannot be used to analyse the rho algorithm for discrete logarithms.

Example 14.9.3. Let $N = 144493$. The values (x_i, x_{2i}) for $i = 1, 2, \dots, 7$ are

$$(2,5),(5,677),(26,9120),(677,81496),(24851,144003),(9120,117992),(90926,94594)$$

and one can check that $\gcd(x_{14} - x_7, N) = 131$.

The reason for this can be seen by considering the values x_i modulo $p = 131$. The sequence of values starts

$$2,\ 5,\ 26,\ 22,\ 92,\ 81,\ 12,\ 14,\ 66,\ 34,\ 109,\ 92$$

and we see that $x_{12} = x_5 = 92$. The tail has length $l_t = 5$ and the head has length $l_h = 7$. Clearly, $x_{14} \equiv x_7 \pmod{p}$.

Exercise 14.9.4. Factor the number 576229 using the rho algorithm.

Exercise 14.9.5. The rho algorithm usually uses the function $f(x) = x^2 + 1$. Why do you think this function is used? Why are the functions $f(x) = x^2$ and $f(x) = x^2 - 2$ less suitable?

Exercise 14.9.6. Show that if N is known to have a prime factor $p \equiv 1 \pmod{m}$ for $m > 2$ then it is preferable to use the polynomial $f(x) = x^m + 1$.

Exercise 14.9.7. Floyd's and Brent's cycle finding methods are both useful for the rho factoring algorithm. Explain why one cannot use the other cycle finding methods listed in Section 14.2.2 (Sedgewick-Szymanski-Yao, Schnorr-Lenstra, Nivasch, distinguished points) for the rho factoring method.

14.10 Pollard Kangaroo Factoring

One can also use the kangaroo method to obtain a factoring algorithm. This is a much more direct application of the discrete logarithm algorithm we have already presented. Let $N = pq$ be a product of two n-bit primes. Then $\sqrt{N} < p + q < 3\sqrt{N}$. Let $g \in \mathbb{Z}_N^*$ be chosen at random. Since $g^{\varphi(N)/2} \equiv 1 \pmod{N}$ we have

$$g^{(N+1)/2} \equiv g^x \pmod{N}$$

for $x = (p+q)/2$. In other words, we have a discrete logarithm problem in $\mathbb{Z}_N^*$ an interval of width $\sqrt{N}$. Using the standard kangaroo algorithm in the group $\mathbb{Z}_N^*$ one expects to find x (and hence split N) in time $\tilde{O}(N^{1/4})$.

Exercise 14.10.1. The above analysis was for integers N which are a product of two primes of very similar size. Let N now be a general composite integer and let $p \mid N$ be the smallest prime dividing N. Then $p < \sqrt{N}$. Choose $g \in \mathbb{Z}_N^*$ and let $h = g^N \pmod{N}$. Then $h \equiv g^x \pmod{p}$ for some $1 \le x < p$. It is natural to try to use the kangaroo method to find x in time $O(\sqrt{p}\log(N)^2)$. If x were found then $g^{N-x} \equiv 1 \pmod{p}$ and so one can split N as $\gcd(g^{N-x} - 1 \pmod{N}, N)$. However, it seems to be impossible to construct an algorithm based on this idea. Explain why.