6.1.2 The Pollard Rho Discrete Logarithm Algorithm

We previously discussed the Pollard rho algorithm for factoring. There is a corresponding algorithm for finding discrete logarithms. Let $(G, \cdot)$ be a group, written multiplicatively, and let $\alpha \in G$ be an element having order n. We assume that the value of n is known. Let $\beta \in \langle \alpha \rangle$ be the element whose discrete logarithm we want to find. Since α has order n, we have that $\log_\alpha \beta \in \mathbb{Z}_n$.

As with the rho algorithm for factoring, we form a sequence $x_1, x_2, \ldots$ by iteratively applying a random-looking function f. Once we obtain two elements x_i and x_j in the sequence such that $x_i = x_j$ and $i < j$, we can (hopefully) compute $\log_\alpha \beta$. Just as we did in the case of the factoring algorithm, we will seek a collision of the form $x_i = x_{2i}$, in order to save time and memory.

Let $S_1 \cup S_2 \cup S_3$ be a partition of G into three subsets of roughly equal size. We define a function $f : \langle \alpha \rangle \times \mathbb{Z}_n \times \mathbb{Z}_n \to \langle \alpha \rangle \times \mathbb{Z}_n \times \mathbb{Z}_n$ as follows:

$$f(x, a, b) = \begin{cases} (\beta x, a, b+1) & \text{if } x \in S_1 \\ (x^2, 2a, 2b) & \text{if } x \in S_2 \\ (\alpha x, a+1, b) & \text{if } x \in S_3. \end{cases}$$

The triples (x, a, b) that we form must have the property that $x = \alpha^a \beta^b$. We begin with an initial triple having this property, say $(1, 0, 0)$. Observe that $f(x, a, b)$ satisfies the desired property if (x, a, b) does. So we define

$$(x_i, a_i, b_i) = \begin{cases} (1, 0, 0) & \text{if } i = 0 \\ f(x_{i-1}, a_{i-1}, b_{i-1}) & \text{if } i \geq 1. \end{cases}$$

We compare the triples (x_{2i}, a_{2i}, b_{2i}) and (x_i, a_i, b_i) until we find a value of $i \geq 1$ such that $x_{2i} = x_i$. When this occurs, we have that

$$\alpha^{a_{2i}} \beta^{b_{2i}} = \alpha^{a_i} \beta^{b_i}.$$

If we denote $c = \log_\alpha \beta$, then it must be the case that

$$\alpha^{a_{2i} + c b_{2i}} = \alpha^{a_i + c b_i}.$$

Since α has order n, it follows that

$$a_{2i} + c b_{2i} \equiv a_i + c b_i \pmod{n}.$$

This can be rewritten as

$$c(b_{2i} - b_i) \equiv a_i - a_{2i} \pmod{n}.$$

If $\gcd(b_{2i} - b_i, n) = 1$, then we can solve for c as follows:

$$c = (a_i - a_{2i})(b_{2i} - b_i)^{-1} \bmod n.$$

We illustrate the application of the above algorithm with an example. Notice that we take care to ensure that $1 \notin S_2$ (since we would obtain $x_i = (1, 0, 0)$ for all integers $i \geq 0$ if $1 \in S_2$).

Example 6.3 The integer $p = 809$ is prime, and it can be verified that the element $\alpha = 89$ has order $n = 101$ in $\mathbb{Z}_{809}{}^*$. The element $\beta = 618$ is in the subgroup $\langle\alpha\rangle$; we will compute $\log_\alpha \beta$.

Suppose we define the sets S_1, S_2 and S_3 as follows:

$$S_1 = \{x \in \mathbb{Z}_{809} : x \equiv 1 \pmod 3\}$$
$$S_2 = \{x \in \mathbb{Z}_{809} : x \equiv 0 \pmod 3\}$$
$$S_3 = \{x \in \mathbb{Z}_{809} : x \equiv 2 \pmod 3\}.$$

For $i = 1, 2, \ldots$, we obtain triples (x_{2i}, a_{2i}, b_{2i}) and (x_i, a_i, b_i) as follows:

i	(x_{2i}, a_{2i}, b_{2i})	(x_i, a_i, b_i)
1	$(618, 0, 1)$	$(76, 0, 2)$
2	$(76, 0, 2)$	$(113, 0, 4)$
3	$(46, 0, 3)$	$(488, 1, 5)$
4	$(113, 0, 4)$	$(605, 4, 10)$
5	$(349, 1, 4)$	$(422, 5, 11)$
6	$(488, 1, 5)$	$(683, 7, 11)$
7	$(555, 2, 5)$	$(451, 8, 12)$
8	$(605, 4, 10)$	$(344, 9, 13)$
9	$(451, 5, 10)$	$(112, 11, 13)$
10	$(422, 5, 11)$	$(422, 11, 15)$

The first collision in the above list is $x_{10} = x_{20} = 422$. The congruence to be solved is

$$c = (11 - 5)(11 - 15)^{-1} \bmod 101 = 6 \times 25 \bmod 101 = 49.$$

Therefore, $\log_{89} 618 = 49$ in the group $\mathbb{Z}_{809}{}^*$. □

The Pollard rho discrete logarithm algorithm is presented as Algorithm 6.1. In this algorithm, we assume, as usual, that $\alpha \in G$ has order n and $\beta \in \langle\alpha\rangle$. The procedure GPROD computes the product of two elements in the group G (and of course, it depends on the group G).

Algorithm 6.1: POLLARD RHO DISCRETE LOG ALGORITHM(G, n, α, β)

```
external GPROD
procedure f(x, a, b)
 if x ∈ S_1
   then f ← (GPROD(β, x), a, (b + 1) mod n)
   else if x ∈ S_2
   then f ← (GPROD(x, x), 2a mod n, 2b mod n)
   else f ← (GPROD(α, x), (a + 1) mod n, b)
 return (f)

main
 define the partition G = S_1 ∪ S_2 ∪ S_3
 (x, a, b) ← f(1, 0, 0)
 (x', a', b') ← f(x, a, b)
 while x ≠ x'
      ⎧ (x, a, b) ← f(x, a, b)
   do ⎨ (x', a', b') ← f(x', a', b')
      ⎩ (x', a', b') ← f(x', a', b')
 if gcd(b' − b, n) ≠ 1
   then return ("failure")
   else return ((a − a')(b' − b)^{-1} mod n)
```

In the situation where $\gcd(b' - b, n) > 1$, Algorithm 6.1 terminates with the output "failure". The situation may not be so bleak, however. If $\gcd(b' - b, n) = d$, then it is not hard to show that the congruence $c(b' - b) \equiv a - a' \pmod{n}$ has exactly d possible solutions. If d is not too large, then it is relatively straightforward to find the d solutions to the congruence and check to see which one is the correct one.

Algorithm 6.1 can be analyzed in a similar fashion as the Pollard rho factoring algorithm. Under reasonable assumptions concerning the randomness of the function f, we expect to be able to compute discrete logarithms in cyclic groups of order n in $O(\sqrt{n})$ iterations of the algorithm.