6.857
NETWORK AND COMPUTER SECURITY
PROBLEM SET 4

JOHN WANG

Problem 4.1

Problem 4.1.a. We will use an algorithm that uses a hash table in order to take the space and time requirements of the discrete log problem down to $\theta(\sqrt{k})$. This will rely on the assumption that hash tables have expected $\theta(1)$ read and write time (which is a legitimate assumption for reasonable hash functions).

Consider the following algorithm. First, we store g^i for values of i in the range $[0, m = 2^(k/2)]$. Then, for each j in the same range, we check to see if $y(g^{-m})^j$ exists in the table. Formally, we have the following algorithm:

```
def discrete_log(p, g, y):
  m = 2^(k/2)
  g_results = {}
  for i in range(0, m):
    g_results[g^i % p] = i

  k = y
  for j in range(0, m):
    if k % p in g_results:
      i = g_results[k % p]
      return (i,j)
    else:
      k = k*inverse(g^m)
```

Here we see that if any pair (i, j) is such that $g^i(g^{2^(k/2)})^j \equiv g^{i+j(2^{k/2})} \equiv y \pmod p$, then we will find it with this algorithm. This is because all possible combinations of i and j are iterated over in this algorithm, with the first loop iterating over all possible values of i and the second loop iterating over all possible values of j.

If any particular pair (i, j) could be the correct pair for the discrete log problem with equal probability, then the expected number of pairs we must go through is 2^k. However, we only need a runtime of $2^{k/2}(1+1/2)$ because we expect to finish halfway through the second loop. This is because we must complete the first loop in any given call to the *discrete_log* function, but each j will have probability $1/m$ of hitting a correct result.

The total space is given by the number of items in the hash table, which is $2^{k/2}$.

0.1. **Problem 4.1.b.** To solve, this problem, we noticed that we could solve two smaller but related problems. We noticed that computing z^s or z^r was equivalent to computing g^{sa} and h^{rb} respectively. This is because we have $g^a h^b \equiv z \pmod p$ and that $g^a h^b = (g_1^s)^a (g_1^r)^b = g_1^{sa+rb}$. Now, this implies for z^s (and equivalently for z^r by symmetry):

$$
\begin{aligned}
z^s &\equiv (g_1^{sa+rb})^s \pmod p && (1)\\
&\equiv g_0^{2(sa+rb)s} \pmod p && (2)\\
&\equiv g_0^{2sa}(g_0^{2rs})^b \pmod p && (3)\\
&\equiv (g_0^2)^{sa} 1^b \pmod p && (4)\\
&\equiv (g_1^s)^a \pmod p && (5)\\
&\equiv g^{sa} \pmod p && (6)
\end{aligned}
$$

Thus, we can take discrete logarithms for x in the congruence $g^x \equiv z^s \pmod{p}$, since we know x only takes on a total of r values. Because of this, we only need to use an algorithm which iterates over the group with r elements, which is relatively fast. Then, once we have found x, we can obtain a by taking $a \equiv xs^{-1} \equiv sas^{-1} \pmod{r}$. A symmetric argument can be applied to b.

To solve the congruence $g^x \equiv z^s \pmod{p}$, we use a fast discrete log algorithm called baby-step giant-step. The algorithm iterates through all values of g^i for $i \in [0, \sqrt{r}]$, and stores them in a hash table. Then, the algorithm checks if any values of $z^s g^{-j\sqrt{r}}$ are in the hash table for $j \in [0, \sqrt{r}]$. This works because you can decompose $x = jr + i$. This algorithm requires $O(\sqrt{r})$ time and space complexity when computing $g^x \equiv z^s \pmod{p}$ and $O(\sqrt{s})$ time and space complexity when $g^y \equiv z^r \pmod{p}$. Since $r, s \approx \sqrt{p}$, we can see that the total time for our algorithm is $O(\sqrt{s} + \sqrt{r}) = O(\sqrt{\sqrt{p}}) = O(p^{1/4})$. Note we could have used any discrete logarithm algorithm which computes relatively quickly (such as Pollard's Rho or Brent's algorithm), but we decided on the baby-step giant-step because of its ease of implementation.

Our group's number is $z = 872037443554961401$ and our results are given below:

i	a	b
40	44833	308847
48	3972467	2996205
56	97799205	6351201
64	1789544324	1110942352
72	14241181606	27418647169