Chapter 10 Discrete Logarithms

10.1 Discrete Logarithms

In the RSA algorithm, we saw how the difficulty of factoring yields useful cryptosystems. There is another number theory problem, namely discrete logarithms, that has similar applications.

Fix a prime p. Let α and β be nonzero integers mod p and suppose

$$\beta \equiv \alpha^x \pmod{p}$$

The problem of finding x is called the **discrete logarithm problem**. If n is the smallest positive integer such that $\alpha^n \equiv 1 \pmod{p}$, we may assume $0 \leq x < n$, and then we denote

$$x = L_\alpha(\beta)$$

and call it the discrete log of β with respect to α (the prime p is omitted from the notation).

For example, let $p = 11$ and let $\alpha = 2$. Since $2^6 \equiv 9 \ (\text{mod } 11)$, we have $L_2(9) = 6$. Of course, $2^6 \equiv 2^{16} \equiv 2^{26} \equiv 9 \ (\text{mod } 11)$, so we could consider taking any one of 6, 16, 26 as the discrete logarithm. But we fix the value by taking the smallest nonnegative value, namely 6. Note that we could have defined the discrete logarithm in this case to be the congruence class 6 mod 10. In some ways, this would be more natural, but there are applications where it is convenient to have a number, not just a congruence class.

Often, α is taken to be a primitive root mod p, which means that every β is a power of $\alpha \pmod p$. If α is not a primitive root, then the discrete logarithm will not be defined for certain values of β.

Given a prime p, it is fairly easy to find a primitive root in many cases. See Exercise 54 in Chapter 3.

The discrete log behaves in many ways like the usual logarithm. In particular, if α is a primitive root mod p, then

$$L_\alpha(\beta_1\beta_2) \equiv L_\alpha(\beta_1) + L_\alpha(\beta_2) \pmod{p-1}$$

(see Exercise 6).

When p is small, it is easy to compute discrete logs by exhaustive search through all possible exponents. However, when p is large this is not feasible. We give some ways of attacking discrete log problems later. However, it is believed that discrete logs are hard to compute in general. This assumption is the basis of several cryptosystems.

The size of the largest primes for which discrete logs can be computed has usually been approximately the same size as the size of largest integers that could be factored (both of these refer to computations that would work for arbitrary numbers of these sizes; special choices of integers will succumb to special techniques, and thus discrete log computations and factorizations work for much larger specially chosen numbers). Compare Table 10.1 with Table 9.2 in Chapter 9.

Table 10.1 Discrete Log Records

Year	Number of Digits of p
2001	120
2005	130
2007	160
2014	180
2016	232

A function $f(x)$ is called a **one-way function** if $f(x)$ is easy to compute, but, given y, it is computationally infeasible to find x with $f(x) = y$. Modular exponentiation is probably an example of such a function. It is easy to compute $\alpha^x \pmod{p}$, but solving $\alpha^x \equiv \beta$ for x is probably hard. Multiplication of large primes can also be regarded as a (probable) one-way function: It is easy to multiply primes but difficult to factor the result to recover the primes. One-way functions have many cryptographic uses.

10.2 Computing Discrete Logs

In this section, we present some methods for computing discrete logarithms. A method based on the birthday attack is discussed in Subsection 12.1.1.

For simplicity, take α to be a primitive root mod p, so $p-1$ is the smallest positive exponent n such that $\alpha^n \equiv 1 \pmod{p}$. This implies that

$$\alpha^{m_1} \equiv \alpha^{m_2} \pmod{p} \Longleftrightarrow m_1 \equiv m_2 \pmod{p-1}.$$

Assume that

$$\beta \equiv \alpha^x, \quad 0 \le x < p-1.$$

We want to find x.

First, it's easy to determine $x \pmod{2}$. Note that

$$\left(\alpha^{(p-1)/2}\right)^2 \equiv \alpha^{p-1} \equiv 1 \pmod{p},$$

so $\alpha^{(p-1)/2} \equiv \pm 1 \pmod{p}$ (see Exercise 15 in Chapter 3). However, $p-1$ is assumed to be the smallest exponent to yield $+1$, so we must have

$$\alpha^{(p-1)/2} \equiv -1 \pmod{p}.$$

Starting with $\beta \equiv \alpha^x \pmod{p}$, raise both sides to the $(p-1)/2$ power to obtain

$$\beta^{(p-1)/2} \equiv \alpha^{x(p-1)/2} \equiv (-1)^x \pmod{p}.$$

Therefore, if $\beta^{(p-1)/2} \equiv +1$, then x is even; otherwise, x is odd.

Example

Suppose we want to solve $2^x \equiv 9 \pmod{11}$. Since

$$\beta^{(p-1)/2} \equiv 9^5 \equiv 1 \pmod{11},$$

we must have x even. In fact, $x = 6$, as we saw previously.

10.2.1 The Pohlig-Hellman Algorithm

The preceding idea was extended by Pohlig and Hellman to give an algorithm to compute discrete logs when $p-1$ has only small prime factors. Suppose

$$p - 1 = \prod_i q_i^{r_i}$$

is the factorization of $p-1$ into primes. Let q^r be one of the factors. We'll compute $L_\alpha(\beta) \pmod{q^r}$. If this can be done for each $q_i^{r_i}$, the answers can be recombined using the Chinese remainder theorem to find the discrete logarithm.

Write

$$x = x_0 + x_1 q + x_2 q^2 + \cdots \text{ with } 0 \le x_i \le q-1.$$

We'll determine the coefficients $x_0,\ x_1,\ \ldots x_{r-1}$ successively, and thus obtain $x \bmod q^r$. Note that

$$\begin{aligned} x\left(\frac{p-1}{q}\right) &= x_0\left(\frac{p-1}{q}\right) + (p-1)\left(x_1 + x_2 q + x_3 q^2 + \cdots\right) \\ &= x_0\left(\frac{p-1}{q}\right) + (p-1)n, \end{aligned}$$

where n is an integer. Starting with $\beta \equiv \alpha^x$, raise both sides to the $(p-1)/q$ power to obtain

$$\beta^{(p-1)/q} \equiv \alpha^{x(p-1)/q} \equiv \alpha^{x_0(p-1)/q}(\alpha^{p-1})^n \equiv \alpha^{x_0(p-1)/q} \pmod{p}.$$

The last congruence is a consequence of Fermat's theorem: $\alpha^{p-1} \equiv 1 \pmod{p}$. To find x_0, simply look at the powers

$$\alpha^{k(p-1)/q} \pmod p, \quad k = 0, 1, 2, \ldots, q-1,$$

until one of them yields $\beta^{(p-1)/q}$. Then $x_0 = k$. Note that since $\alpha^{m_1} \equiv \alpha^{m_2} \iff m_1 \equiv m_2 \pmod{p-1}$, and since the exponents $k(p-1)/q$ are distinct mod $p-1$, there is a unique k that yields the answer.

An extension of this idea yields the remaining coefficients. Assume that $q^2 | p-1$. Let

$$\beta_1 \equiv \beta\alpha^{-x_0} \equiv \alpha^{q(x_1+x_2q+\cdots)} \pmod p.$$

Raise both sides to the $(p-1)/q^2$ power to obtain

$$\begin{aligned}\beta_1^{(p-1)/q^2} &\equiv \alpha^{(p-1)(x_1+x_2q+\cdots)/q} \\ &\equiv \alpha^{x_1(p-1)/q}(\alpha^{p-1})^{x_2+x_3q+\cdots} \\ &\equiv \alpha^{x_1(p-1)/q} \pmod p.\end{aligned}$$

The last congruence follows by applying Fermat's theorem. We couldn't calculate $\beta_1^{(p-1)/q^2}$ as $(\beta_1^{p-1})^{1/q^2}$ since fractional exponents cause problems. Note that every exponent we have used is an integer.

To find x_1, simply look at the powers

$$\alpha^{k(p-1)/q} \pmod p, \quad k = 0, 1, 2, \ldots, q-1,$$

until one of them yields $\beta_1^{(p-1)/q^2}$. Then $x_1 = k$.

If $q^3 | p-1$, let $\beta_2 \equiv \beta_1\alpha^{-x_1q}$ and raise both sides to the $(p-1)/q^3$ power to obtain x_2. In this way, we can continue until we find that q^{r+1} doesn't divide $p-1$. Since we cannot use fractional exponents, we must stop. But we have determined $x_0, x_1, \ldots, x_{r-1}$, so we know $x \bmod q^r$.

Repeat the procedure for all the prime factors of $p-1$. This yields x mod $q_i^{r_i}$ for all i. The Chinese remainder theorem allows us to combine these into a congruence for x mod $p-1$. Since $0 \le x < p-1$, this determines x.

Example

Let $p = 41$, $\alpha = 7$, and $\beta = 12$. We want to solve

$$7^x \equiv 12 \pmod{41}.$$

Note that

$$41 - 1 = 2^3 \cdot 5.$$

First, let $q = 2$ and let's find $x \bmod 2^3$. Write $x \equiv x_0 + 2x_1 + 4x_2 \pmod 8$.

To start,

$$\beta^{(p-1)/2} \equiv 12^{20} \equiv 40 \equiv -1 \pmod{41},$$

and

$$\alpha^{(p-1)/2} \equiv 7^{20} \equiv -1 \pmod{41}.$$

Since

$$\beta^{(p-1)/2} \equiv (\alpha^{(p-1)/2})^{x_0} \pmod{41},$$

we have $x_0 = 1$. Next,

$$\beta_1 \equiv \beta\alpha^{-x_0} \equiv 12 \cdot 7^{-1} \equiv 31 \pmod{41}.$$

Also,

$$\beta_1^{(p-1)/2^2} \equiv 31^{10} \equiv 1 \pmod{41}.$$

Since

$$\beta_1^{(p-1)/2^2} \equiv (\alpha^{(p-1)/2})^{x_1} \pmod{41},$$

we have $x_1 = 0$. Continuing, we have

$$\beta_2 \equiv \beta_1\alpha^{-2x_1} \equiv 31 \cdot 7^0 \equiv 31 \pmod{41},$$

and

$$\beta_2^{(p-1)/q^3} \equiv 31^5 \equiv -1 \equiv (\alpha^{(p-1)/2})^{x_2} \pmod{41}.$$

Therefore, $x_2 = 1$. We have obtained

$$x \equiv x_0 + 2x_1 + 4x_2 \equiv 1 + 4 \equiv 5 \pmod{8}.$$

Now, let $q = 5$ and let's find x mod 5. We have

$$\beta^{(p-1)/5} \equiv 12^8 \equiv 18 \pmod{41}$$

and

$$\alpha^{(p-1)/q} \equiv 7^8 \equiv 37 \pmod{41}.$$

Trying the possible values of k yields

$$37^0 \equiv 1, \quad 37^1 \equiv 37, \quad 37^2 \equiv 16, \quad 37^3 \equiv 18, \quad 37^4 \equiv 10 \pmod{41}.$$

Therefore, 37^3 gives the desired answer, so $x \equiv 3 \pmod{5}$.

Since $x \equiv 5 \pmod{8}$ and $x \equiv 3 \pmod{5}$, we combine these to obtain $x \equiv 13 \pmod{40}$, so $x = 13$. A quick calculation checks that $7^{13} \equiv 12 \pmod{41}$, as desired.

As long as the primes q involved in the preceding algorithm are reasonably small, the calculations can be done quickly. However, when q is large, calculating the numbers $\alpha^{k(p-1)/q}$ for $k = 0, 1, 2, \ldots, q-1$ becomes infeasible, so the algorithm no longer is practical. This means that if we want a discrete logarithm to be hard, we should make sure that $p - 1$ has a large prime factor.

Note that even if $p - 1 = tq$ has a large prime factor q, the algorithm can determine discrete logs mod t if t is composed of small prime factors. For this reason, often β is chosen to be a power of α^t. Then the discrete log is automatically 0 mod t, so the discrete log hides only mod q information, which the algorithm cannot find. If the discrete log x represents a secret (or better, t times a secret), this means that an attacker does not obtain partial information by determining x mod t, since there is no information hidden this way. This idea is used in the Digital Signature Algorithm, which we discuss in Chapter 13.

10.2.2 Baby Step, Giant Step

Eve wants to find x such that $\alpha^x \equiv \beta \pmod{p}$. She does the following. First, she chooses an integer N with $N^2 \geq p-1$, for example $N = \lceil \sqrt{p-1} \rceil$ (where $\lceil x \rceil$ means round x up to the nearest integer). Then she makes two lists:

1. $\alpha^j \pmod{p}$ for $0 \leq j < N$
2. $\beta\alpha^{-Nk} \pmod{p}$ for $0 \leq k < N$

She looks for a match between the two lists. If she finds one, then

$$\alpha^j \equiv \beta\alpha^{-Nk},$$

so $\alpha^{j+Nk} \equiv \beta$. Therefore, $x = j + Nk$ solves the discrete log problem.

Why should there be a match? Since $0 \leq x < p-1 \leq N^2$, we can write x in base N as $x = x_0 + Nx_1$ with $0 \leq x_0, x_1 < N$. In fact, $x_1 = [x/N]$ and $x_0 = x - Nx_1$. Therefore,

$$j = x_0, \quad k = x_1$$

gives the desired match.

The list α^j for $j = 0, 1, 2, \dots$ is the set of “Baby Steps” since the elements of the list are obtained by multiplying by α, while the “Giant Steps” are obtained in the second list by multiplying by α^{-N}. It is, of course, not necessary to compute all of the second list. Each element, as it is computed, can be compared with the first list. As soon as a match is found, the computation stops.

The number of steps in this algorithm is proportional to $N \approx \sqrt{p}$ and it requires storing approximately N numbers. Therefore, the method works for primes p up to 10^{20}, or even slightly larger, but is impractical for very large p.

For an example, see Example 35 in the Computer Appendices.

10.2.3 The Index Calculus

The idea is similar to the method of factoring in Subsection 9.4.1 . Again, we are trying to solve $\beta \equiv \alpha^x \pmod{p}$, where p is a large prime and α is a primitive root.

First, there is a precomputation step. Let B be a bound and let $p_1, p_2, \ldots, p_m,$ be the primes less than B. This set of primes is called our **factor base**. Compute $\alpha^k \pmod{p}$ for several values of k. For each such number, try to write it as a product of the primes less than B. If this is not the case, discard α^k. However, if $\alpha^k \equiv \prod p_i^{a_i} \pmod{p}$, then

$$k \equiv \sum a_i L_\alpha(p_i) \pmod{p-1}.$$

When we obtain enough such relations, we can solve for $L_\alpha(p_i)$ for each i.

Now, for random integers r, compute $\beta\alpha^r \pmod{p}$. For each such number, try to write it as a product of primes less than B. If we succeed, we have $\beta\alpha^r \equiv \prod p_i^{b_i} \pmod{p}$, which means

$$L_\alpha(\beta) \equiv -r + \sum b_i L_\alpha(p_i) \pmod{p-1}.$$

This algorithm is effective if p is of moderate size. This means that p should be chosen to have at least 200 digits, maybe more, if the discrete log problem is to be hard.

Example

Let $p = 131$ and $\alpha = 2$. Let $B = 10$, so we are working with the primes 2,3,5,7. A calculation yields the following:

$$\begin{aligned} 2^1 &\equiv 2 \pmod{131} \\ 2^8 &\equiv 5^3 \pmod{131} \\ 2^{12} &\equiv 5 \cdot 7 \pmod{131} \\ 2^{14} &\equiv 3^2 \pmod{131} \\ 2^{34} &\equiv 3 \cdot 5^2 \pmod{131}. \end{aligned}$$

Therefore,

$$\begin{aligned} 1 &\equiv L_2(2) \pmod{130} \\ 8 &\equiv 3L_2(5) \pmod{130} \\ 12 &\equiv L_2(5) + L_2(7) \pmod{130} \\ 14 &\equiv 2L_2(3) \pmod{130} \\ 34 &\equiv L_2(3) + 2L_2(5) \pmod{130}. \end{aligned}$$

The second congruence yields $L_2(5) \equiv 46 \pmod{130}$. Substituting this into the third congruence yields $L_2(7) \equiv -34 \equiv 96 \pmod{130}$. The fourth congruence yields only the value of $L_2(3) \pmod{65}$ since $\gcd(2, 130) \neq 1$. This gives two choices for $L_2(3) \pmod{130}$. Of course, we could try them and see which works. Or we could use the fifth congruence to obtain $L_2(3) \equiv 72 \pmod{130}$. This finishes the precomputation step.

Suppose now that we want to find $L_2(37)$. Trying a few randomly chosen exponents yields $37 \cdot 2^{43} \equiv 3 \cdot 5 \cdot 7 \pmod{131}$, so

$$L_2(37) \equiv -43 + L_2(3) + L_2(5) + L_2(7) \equiv 41 \pmod{130}.$$

Therefore, $L_2(37) = 41$.

Of course, once the precomputation has been done, it can be reused for computing several discrete logs for the same prime p.

10.2.4 Computing Discrete Logs Mod 4

When $p \equiv 1 \pmod 4$, the Pohlig-Hellman algorithm computes discrete logs mod 4 quite quickly. What happens when $p \equiv 3 \pmod 4$? The Pohlig-Hellman algorithm won't work, since it would require us to raise numbers to the $(p-1)/4$ power, which would yield the ambiguity of a fractional exponent. The surprising fact is that if we have an algorithm that quickly computes discrete logs mod 4 for a prime $p \equiv 3 \pmod 4$, then we can use it to compute discrete logs mod p quickly. Therefore, it is unlikely that such an algorithm exists.

There is a philosophical reason that we should not expect such an algorithm. A natural point of view is that the discrete log should be regarded as a number mod $p-1$. Therefore, we should be able to obtain information on the discrete log only modulo the power of 2 that appears in $p-1$. When $p \equiv 3 \pmod 4$, this means that asking questions about discrete logs mod 4 is somewhat unnatural. The question is possible only because we normalized the discrete log to be an integer between 0 and $p-2$. For example, $2^6 \equiv 2^{16} \equiv 9 \pmod{11}$. We defined $L_2(9)$ to be 6 in this case; if we had allowed it also to be 16, we would have two values for $L_2(9)$, namely 6 and 16, that are not congruent mod 4. Therefore, from this point of view, we shouldn't even be asking about $L_2(9) \bmod 4$.

We need the following lemma, which is similar to the method for computing square roots mod a prime $p \equiv 3 \pmod 4$ (see Section 3.9).

Lemma

Let $p \equiv 3 \pmod 4$ be prime, let $r \geq 2$, and let y be an integer. Suppose α and γ are two nonzero numbers mod p such that $\gamma \equiv \alpha^{2^r y} \pmod p$. Then

$$\gamma^{(p+1)/4} \equiv \alpha^{2^{r-1}y} \pmod p.$$

Proof.

$$\gamma^{(p+1)/4} \equiv \alpha^{(p+1)2^{r-2}y} \equiv \alpha^{2^{r-1}y}(\alpha^{p-1})^{2^{r-2}y} \equiv \alpha^{2^{r-1}y} \pmod p.$$

The final congruence is because of Fermat's theorem.

Fix the prime $p \equiv 3 \pmod 4$ and let α be a primitive root. Assume we have a machine that, given an input β, gives the output $L_\alpha(\beta) \bmod 4$. As we saw previously, it is easy to compute $L_\alpha(\beta) \bmod 2$. So the new information supplied by the machine is really only the second bit of the discrete log.

Now assume $\alpha^x \equiv \beta \pmod p$. Let $x = x_0 + 2x_1 + 4x_2 + \cdots + 2^n x_n$ be the binary expansion of x. Using the $L_\alpha(\beta) \pmod 4$ machine, we determine x_0 and x_1. Suppose we have determined $x_0, x_1, \ldots, x_{r-1}$ with $r \geq 2$. Let

$$\beta_r \equiv \beta\alpha^{-(x_0+\cdots+2^{r-1}x_{r-1})} \equiv \alpha^{2^r(x_r+2x_{r+1}+\cdots)}.$$

Using the lemma $r-1$ times, we find

$$\beta_r^{((p+1)/4)^{r-1}} \equiv \alpha^{2(x_r+2x_{r+1}+\cdots)} \pmod p.$$

Applying the $L_\alpha \pmod 4$ machine to this equation yields the value of x_r. Proceeding inductively, we obtain all the values $x_0, x_1, \ldots, x_n$. This determines x, as desired.

It is possible to make this algorithm more efficient. See, for example, [Stinson1, page 175].

In conclusion, if we believe that finding discrete logs for $p \equiv 3 \pmod 4$ is hard, then so is computing such discrete logs mod 4.

10.3 Bit Commitment

Alice claims that she has a method to predict the outcome of football games. She wants to sell her method to Bob. Bob asks her to prove her method works by predicting the results of the games that will be played this weekend. “No way,” says Alice. “Then you will simply make your bets and not pay me. If you want me to prove my system works, why don’t I show you my predictions for last week’s games?” Clearly there is a problem here. We’ll show how to resolve it.

Here’s the setup. Alice wants to send a bit b, which is either 0 or 1, to Bob. There are two requirements.

1. Bob cannot determine the value of the bit without Alice’s help.
2. Alice cannot change the bit once she sends it.

One way is for Alice to put the bit in a box, put her lock on it, and send it to Bob. When Bob wants the value of the bit, Alice removes the lock and Bob opens the box. We want to implement this mathematically in such a way that Alice and Bob do not have to be in the same room when the bit is revealed.

Here is a solution. Alice and Bob agree on a large prime $p \equiv 3 \pmod 4$ and a primitive root α. Alice chooses a random number $x < p-1$ whose second bit x_1 is b. She sends $\beta \equiv \alpha^x \pmod p$ to Bob. We assume that Bob cannot compute discrete logs for p. As pointed out in the last section, this means that he cannot compute discrete logs mod 4. In particular, he cannot determine the value of $b = x_1$. When Bob wants to know the value of b, Alice sends him the full value of x, and by looking at $x \bmod 4$, he finds b. Alice cannot send a value of x different than the one already used, since Bob checks that

$\beta \equiv \alpha^x \pmod{p}$, and this equation has a unique solution $x < p - 1$.

Back to football: For each game, Alice sends $b = 1$ if she predicts the home team will win, $b = 0$ if she predicts it will lose. After the game has been played, Alice reveals the bit to Bob, who can see whether her predictions were correct. In this way, Bob cannot profit from the information by receiving it before the game, and Alice cannot change her predictions once the game has been played.

Bit commitment can also be accomplished with many other one-way functions. For example, Alice can take a random 100-bit string, followed by the bit b, followed by another 100-bit string. She applies the one-way function to this string and sends the result to Bob. After the game, she sends the full 201-bit string to Bob, who applies the one-way function and compares with what Alice originally sent.

10.4 Diffie-Hellman Key Exchange

An important problem in cryptography is how to establish keys for use in cryptographic protocols such as DES or AES, especially when the two parties are widely separated. Public key methods such as RSA provide one solution. In the present section, we describe a different method, due to Diffie and Hellman, whose security is very closely related to the difficulty of computing discrete logarithms.

There are several technical implementation issues related to any key distribution scheme. Some of these are discussed in Chapter 15. In the present section, we restrict ourselves to the basic Diffie-Hellman algorithm. For more discussion of some security concerns about implementations of the Diffie-Hellman protocol, see [Adrian et al.].

Here is how Alice and Bob establish a private key K. All of their communications in the following algorithm are over public channels.

1. Either Alice or Bob selects a large prime number p for which the discrete logarithm problem is hard and a primitive root $\alpha \pmod{p}$. Both p and α can be made public.

2. Alice chooses a secret random x with $1 \le x \le p-2$, and Bob selects a secret random y with $1 \le y \le p-2$.

3. Alice sends $\alpha^x \pmod{p}$ to Bob, and Bob sends $\alpha^y \pmod{p}$ to Alice.

4. Using the messages that they each have received, they can each calculate the session key K. Alice calculates K by $K \equiv (\alpha^y)^x \pmod{p}$, and Bob calculates K by $K \equiv (\alpha^x)^y \pmod{p}$.

There is no reason that Alice and Bob need to use all of K as their key for their communications. Now that they have the same number K, they can use some prearranged procedure to produce a key. For example, they could use the middle 56 bits of K to obtain a DES key.

Suppose Eve listens to all the communications between Alice and Bob. She will know α^x and α^y. If she can compute discrete logs, then she can find the discrete log of α^x to obtain x. Then she raises α^y to the power x to obtain $\alpha^{xy} \equiv K$. Once Eve has K, she can use the same procedure as Alice and Bob to extract a communication key. Therefore, if Eve can compute discrete logs, she can break the system.

However, Eve does not necessarily need to compute x or y to find K. What she needs to do is solve the following:

Computational Diffie-Hellman Problem: Let p be prime and let α be a primitive root mod p. Given $\alpha^x \pmod{p}$ and $\alpha^y \pmod{p}$, find $\alpha^{xy} \pmod{p}$.

It is not known whether or not this problem is easier than computing discrete logs. The reasoning above shows that it is no harder than computing discrete logs. A related problem is the following:

Decision Diffie-Hellman Problem: Let p be prime and let α be a primitive root mod p. Given $\alpha^x \pmod{p}$ and $\alpha^y \pmod{p}$, and $c \not\equiv 0 \pmod{p}$, decide whether or not $c \equiv \alpha^{xy} \pmod{p}$.

In other words, if Eve claims that she has found c with $c \equiv \alpha^{xy} \pmod{p}$, and offers to sell you this information, can you decide whether or not she is telling the truth? Of course, if you can solve the computational Diffie-Hellman problem, then you simply compute $\alpha^{xy} \pmod{p}$ and check whether it is c (and then you can ignore Eve's offer).

Conversely, does a method for solving the decision Diffie-Hellman problem yield a solution to the computational Diffie-Hellman problem? This is not known at present. One obvious method is to choose many values of c and check each value until one equals $\alpha^{xy} \pmod{p}$. But this brute force method takes at least as long as computing discrete logarithms by brute force, so is impractical. There are situations involving elliptic curves, analogous to the present setup, where a fast solution is known for the decision Diffie-Hellman problem but no practical solution is known for the computational Diffie-Hellman problem (see Exercise 8 in Chapter 22).

10.5 The ElGamal Public Key Cryptosystem

In Chapter 9, we studied a public key cryptosystem whose security is based on the difficulty of factoring. It is also possible to design a system whose security relies on the difficulty of computing discrete logarithms. This was done by ElGamal in 1985. This system does not quite fit the definition of a public key cryptosystem given at the end of Chapter 9, since the set of possible plaintexts (integers mod p) is not the same as the set of possible ciphertexts (pairs of integers (r, t) mod p). However, this technical point will not concern us.

Alice wants to send a message m to Bob. Bob chooses a large prime p and a primitive root α. Assume m is an integer with $0 \le m < p$. If m is larger, break it into smaller blocks. Bob also chooses a secret integer b and computes $\beta \equiv \alpha^b \pmod{p}$. The information (p, α, β) is made public and is Bob's public key. Alice does the following:

1. Downloads (p, α, β)
2. Chooses a secret random integer k and computes $r \equiv \alpha^k \pmod{p}$
3. Computes $t \equiv \beta^k m \pmod{p}$
4. Sends the pair (r, t) to Bob

Bob decrypts by computing

$$tr^{-b} \equiv m \pmod{p}.$$

This works because

$$tr^{-b} \equiv \beta^k m(\alpha^k)^{-b} \equiv (\alpha^b)^k m\alpha^{-bk} \equiv m \pmod{p}.$$

If Eve determines b, then she can also decrypt by the same procedure that Bob uses. Therefore, it is important for Bob to keep b secret. The numbers α and β are public, and $\beta \equiv \alpha^b \pmod{p}$. The difficulty of computing discrete logs is what keeps b secure.

Since k is a random integer, β^k is a random nonzero integer mod p. Therefore, $t \equiv \beta^k m \pmod{p}$ is m multiplied by a random integer, and t is random mod p (unless $m = 0$, which should be avoided, of course). Therefore, t gives Eve no information about m. Knowing r does not seem to give Eve enough additional information.

The integer k is difficult to determine from r, since this is again a discrete logarithm problem. However, if Eve finds k, she can then calculate $t\beta^{-k}$, which is m.

It is important that a different random k be used for each message. Suppose Alice encrypts messages m_1 and m_2 for Bob and uses the same value k for each message. Then r will be the same for both messages, so the ciphertexts will be (r, t_1) and (r, t_2). If Eve finds out the plaintext m_1, she can also determine m_2, as follows. Note that

$$t_1/m_1 \equiv \beta^k \equiv t_2/m_2 \pmod{p}.$$

Since Eve knows t_1 and t_2, she computes $m_2 \equiv t_2 m_1/t_1 \pmod{p}$.

In Chapter 21, we'll meet an analog of the ElGamal method that uses elliptic curves.

10.5.1 Security of ElGamal Ciphertexts

Suppose Eve claims to have obtained the plaintext m corresponding to an RSA ciphertext c. It is easy to verify her claim: Compute $m^e \pmod{n}$ and check whether this equal c. Now suppose instead that Eve claims to possess the message m corresponding to an ElGamal encryption (r, t). Can you verify her claim? It turns out that this is as hard as the decision Diffie-Hellman problem from Section 10.4. In this aspect, the ElGamal algorithm is therefore much different than the RSA algorithm (of course, if some randomness is added to an RSA plaintext through OAEP, for example, then RSA encryption has a similar property).

Proposition

A machine that solves Decision Diffie-Hellman problems mod p can be used to decide the validity of mod p ElGamal ciphertexts, and a machine that decides the validity of mod p ElGamal ciphertexts can be used to solve Decision Diffie-Hellman problems mod p.

Proof. Suppose first that you have a machine M_1 that can decide whether an ElGamal decryption is correct. In other words, when given the inputs $p, \alpha, \beta, (r, t), m$, the machine outputs “yes” if m is the decryption of (r, t) and outputs “no” otherwise. Let’s use this machine to solve the decision Diffie-Hellman problem. Suppose you are given α^x and α^y, and you want to decide whether or not $c \equiv \alpha^{xy}$. Let $\beta = \alpha^x$ and $r = \alpha^y \pmod{p}$. Moreover, let $t = c$ and $m = 1$. Input

$$p, \quad \alpha, \quad \beta, \quad (\alpha^x, \alpha^{xy}), \quad 1$$

into M_1. Note that in the present setup, x is the secret integer b and α^y takes the place of the $r \equiv \alpha^k$. The correct decryption of $(r, t) = (\alpha^y, \alpha^{xy})$ is $tr^{-b} \equiv cr^{-x} \equiv c\alpha^{-xy}$. Therefore, M_1 outputs “yes” exactly when $m = 1$ is the same as $c\alpha^{-xy} \pmod{p}$, namely when $c \equiv \alpha^{xy} \pmod{p}$. This solves the decision Diffie-Hellman problem.

Conversely, suppose you have a machine M_2 that can solve the decision Diffie-Hellman problem. This means that if you give M_2 inputs p, α, α^x, α^y, c, then M_2 outputs “yes” if $c \equiv \alpha^{xy}$ and outputs “no” if not. Let m be the claimed decryption of the ElGamal ciphertext (r, t). Input $\beta \equiv \alpha^b$ as α^x, so $x = b$, and input $r \equiv \alpha^k$ as α^y so $y = k$. Input $tm^{-1} \pmod{p}$ as c. Note that m is the correct plaintext for the ciphertext (r, t) if and only if $m \equiv tr^{-a} \equiv t\alpha^{-xy}$, which happens if and only if $tm^{-1} \equiv \alpha^{xy}$. Therefore, m is the correct plaintext if and only if $c \equiv tm^{-1}$ is the solution to the Diffie-Hellman problem. Therefore, with these inputs, M_2 outputs “yes” exactly when m is the correct plaintext.

The reasoning just used can also be used to show that solving the computational Diffie-Hellman problem is equivalent to breaking the ElGamal system:

Proposition

A machine that solves computational Diffie-Hellman problems mod p can be used to decrypt mod p ElGamal ciphertexts, and a machine that decrypts mod p ElGamal ciphertexts can be used to solve computational Diffie-Hellman problems mod p.

Proof. If we have a machine M_3 that can decrypt all ElGamal ciphertexts, then input $\beta \equiv \alpha^x$ (so $a = x$) and $r \equiv \alpha^y$. Take any nonzero value for t. Then M_3 outputs $m \equiv tr^{-a} \equiv t\alpha^{-xy}$. Therefore, $tm^{-1} \pmod{p}$ yields the solution α^{xy} to the computational Diffie-Hellman problem.

Conversely, suppose we have a machine M_4 that can solve computational Diffie-Hellman problems. If we have an ElGamal ciphertext (r, t), then we input $\alpha^x = \alpha^a \equiv \beta$ and $\alpha^y \equiv \alpha^k \equiv r$. Then M_4 outputs $\alpha^{xy} \equiv \alpha^{ak}$. Since $m \equiv tr^{-a} \equiv t\alpha^{-ak}$, we obtain the plaintext m.

10.6 Exercises

1.
   1. Let $p = 13$. Compute $L_2(3)$.
   2. Show that $L_2(11) = 7$.

2.
   1. Let $p = 17$. Compute $L_3(2)$.
   2. Show that $L_3(15) = 6$.

3.
   1. Compute $6^5 \pmod{11}$.
   2. Let $p = 11$. Then 2 is a primitive root. Suppose $2^x \equiv 6 \pmod{11}$. Without finding the value of x, determine whether x is even or odd.

4. Let $p = 19$. Then 2 is a primitive root. Use the Pohlig-Hellman method to compute $L_2(14)$.

5. It can be shown that 5 is a primitive root for the prime 1223. You want to solve the discrete logarithm problem $5^x \equiv 3 \pmod{1223}$. Given that $3^{611} \equiv 1 \pmod{1223}$, determine whether x is even or odd.

6.
   1. Let α be a primitive root mod p. Show that

      $$L_\alpha(\beta_1\beta_2) \equiv L_\alpha(\beta_1) + L_\alpha(\beta_2) \pmod{p-1}.$$

      (Hint: You need the proposition in Section 3.7.)
   2. More generally, let α be arbitrary. Show that

      $$L_\alpha(\beta_1\beta_2) \equiv L_\alpha(\beta_1) + L_\alpha(\beta_2) \pmod{\text{ord}_p(\alpha)},$$

      where $\text{ord}_p(\alpha)$ is defined in Exercise 53 in Chapter 3.

7. Let $p = 101$, so 2 is a primitive root. It can be shown that $L_2(3) = 69$ and $L_2(5) = 24$.
   1. Using the fact that $24 = 2^3 \cdot 3$, evaluate $L_2(24)$.
   2. Using the fact that $5^3 \equiv 24 \pmod{101}$, evaluate $L_2(24)$.

8. The number 12347 is prime. Suppose Eve discovers that $2^{10000} \cdot 79 \equiv 2^{5431} \pmod{12347}$. Find an integer k with $0 < k < 12347$ such that $2^k \equiv 79 \pmod{12347}$.

9. Suppose you know that

$$3^6 \equiv 44 \pmod{137}, \quad 3^{10} \equiv 2 \pmod{137}.$$

Find a value of x with $0 \le x \le 135$ such that $3^x \equiv 11 \ (\text{mod } 137)$.

10. Let p be a large prime and suppose $\alpha^{10^{18}} \equiv 1 \ (\text{mod } p)$. Suppose $\beta \equiv \alpha^k \ (\text{mod } p)$ for some integer k.

    1. Explain why we may assume that $0 \le k < 10^{18}$.

    2. Describe a BabyStep, Giant Step method to find k. (Hint: One list can contain numbers of the form $\beta\alpha^{-10^9 j}$.)

11. 1. Suppose you have a random 500-digit prime p. Suppose some people want to store passwords, written as numbers. If x is the password, then the number $2^x \ (\text{mod } p)$ is stored in a file. When y is given as a password, the number $2^y \ (\text{mod } p)$ is compared with the entry for the user in the file. Suppose someone gains access to the file. Why is it hard to deduce the passwords?

    2. Suppose p is instead chosen to be a five-digit prime. Why would the system in part (a) not be secure?

12. Let's reconsider Exercise 55 in Chapter 3 from the point of view of the Pohlig-Hellman algorithm. The only prime q is 2. For k as in that exercise, write $k = x_0 + 2x_1 + \cdots + 2^{15}x_{15}$.

    1. Show that the Pohlig-Hellman algorithm yields

    $$x_0 = x_1 = \cdots = x_{10} = 0$$

    and

    $$2 = \beta = \beta_1 = \cdots = \beta_{11}.$$

    2. Use the Pohlig-Hellman algorithm to compute k.

13. In the Diffie-Hellman Key Exchange protocol, suppose the prime is $p = 17$ and the primitive root is $\alpha = 3$. Alice's secret is $a = 3$ and Bob's secret is $b = 5$. Describe what Alice and Bob send each other and determine the shared secret that they obtain.

14. In the Diffie-Hellman Key Exchange protocol, Alice thinks she can trick Eve by choosing her secret to be $a = 0$. How will Eve recognize that Alice made this choice?

15. In the Diffie-Hellman key exchange protocol, Alice and Bob choose a primitive root α for a large prime p. Alice sends $x_1 \equiv \alpha^a \ (\text{mod } p)$ to Bob, and Bob sends $x_2 \equiv \alpha^b \ (\text{mod } p)$ to Alice. Suppose Eve bribes Bob to tell her the values of b and x_2.

However, he neglects to tell her the value of α. Suppose $\gcd(b, p-1) = 1$. Show how Eve can determine α from the knowledge of p, x_2, and b.

16. In the ElGamal cryptosystem, Alice and Bob use $p = 17$ and $\alpha = 3$. Bob chooses his secret to be $a = 6$, so $\beta = 15$. Alice sends the ciphertext $(r, t) = (7, 6)$. Determine the plaintext m.

17. Consider the following Baby Step, Giant Step attack on RSA, with public modulus n. Eve knows a plaintext m and a ciphertext c with $\gcd(c, n) = 1$. She chooses $N^2 \geq n$ and makes two lists: The first is $c^j \pmod{n}$ for $0 \leq j < N$. The second is $mc^{-Nk} \pmod{n}$ for $0 \leq k < N$.

    1. Why is there always a match between the two lists, and how does a match allow Eve to find the decryption exponent d?

    2. Your answer to (a) is probably partly false. What you have really found is an exponent d such that $c^d \equiv m \pmod{n}$. Give an example of a plaintext–ciphertext pair where the d you find is not the encryption exponent. (However, usually d is very close to being the correct decryption exponent.)

    3. Why is this not a useful attack on RSA? (Hint: How long are the lists compared to the time needed to factor n by trial division?)

18. Alice and Bob are using the ElGamal public key cryptosystem, but have set it up so that only Alice, Bob, and a few close associates (not Eve) know Bob's public key. Suppose Alice is sending the message *dismiss Eve* to Bob, but Eve intercepts the message and prevents Bob from receiving it. How can Eve change the message to *promote Eve* before sending it to Bob?

10.7 Computer Problems

1. Let $p = 53047$. Verify that $L_3(8576) = 1234$.
2. Let $p = 31$. Evaluate $L_3(24)$.
3. Let $p = 3989$. Then 2 is a primitive root mod p.
    1. Show that $L_2(3925) = 2000$ and $L_2(1046) = 3000$.
    2. Compute $L_2(3925 \cdot 1046)$. (Note: The answer should be less than 3988.)
4. Let $p = 1201$.
    1. Show that $11^{1200/q} \not\equiv 1 \pmod{1201}$ for $q = 2, 3, 5$.
    2. Use method of Exercise 54 in Chapter 3 plus the result of part (a) to show that 11 is a primitive root mod 1201.
    3. Use the Pohlig-Hellman algorithm to find $L_{11}(2)$.
    4. Use the Baby Step, Giant Step method to find $L_{11}(2)$.