Discrete Logarithm Problem

Finite Fields

The finite field *GF(q)* exists iff $q = p^e$ for some prime p.

Example: *GF(9)*

$GF(9) = \{a + bi \mid a, b \in Z_3, i^2 = i + 1\}$

$= \{0, 1, 2, i, 1+i, 2+i, 2i, 1+2i, 2+2i\}$

Addition: $(a + bi) + (c + di) := (a + c) + (b + d)i$

Multiplication: $(a + bi)(c + di) = ac + (ad + bc)i + bdi^2$

$= (ac + bd) + (ad + bc + bd)i$

For instance,

$(1 + 2i) + (1 + i) = 2 + 0i = 2$

$(1 + 2i)(1 + i) = (1 + 2) + (1 + 2 + 2)i = 2i$

$(1 + i)^2 = (1 + 1) + (1 + 1 + 1)i = 2$

Finite Fields

Consider the powers of i:

$i, i^2 = i+1, i^3 = i^2 + i = 2i + 1, i^4 = (i^2)^2 = (i+1)^2 = 2,$

$i^5 = (i^4)i = 2i, i^6 = (i^4)(i^2) = 2+2i, i^7 = 2 + i, i^8 = 1$

An element of a field whose powers give all the non-zero elements of that field is called a ***primitive element*** of the field.

If F is a finite field, the set F\{0} of non-zero elements is denoted by **F*** and is a (cyclic) group under multiplication. A primitive element of F is also called a ***generator of F****.

Not all the elements of F* are primitive.

$2, 2^2 = 1$

$i+1, (i+1)^2 = 2, (i+1)^3 = 2i + 2, (i+1)^4 = 1$

Square and Multiply

Let F = GF(q) and take α as a primitive element of F. Any $c \in F^*$ has a unique representation as $c = \alpha^m$, for $0 \le m \le q-1$. c can be computed from α and m with only $\lceil 2 \log_2 q \rceil$ multiplications. The binary representation of m gives the order of the needed multiplications, which consist only of squaring and multiplying by α. For instance, if m = 171 then $171 = 128 + 32 + 8 + 2 + 1 = (10101011)_2$ and the computation of α^{171} is carried out by starting with 1, then, working from the most significant bit down, we square the current value and if there is a 1 in the binary representation we also multiply by α. Thus,

$$\alpha^{171} = ((((((((1)^2\alpha)^2)^2\alpha)^2)^2\alpha)^2)^2\alpha)^2\alpha.$$

Discrete Logarithm Problem

On the other hand, given c and α, finding m is a more difficult proposition and is called the ***discrete logarithm problem***.

If taking a power is of O(t) time, then finding a logarithm is of $O(2^{t/2})$ time. And this can be made prohibitively large if $t = \log_2 q$ is large.

Diffie-Hellman Key Exchange

The difficulty of taking logarithms makes exponentiation in a finite field a one-way function (*not a trapdoor function however*). This can be used in a public key exchange protocol. Public knowledge is p, and α^{m_U} for each user U, while each user keeps secret their value of m_U. To exchange keys without transmission, A looks up B's public key and exponentiates it with his own secret exponent. B does the same to A's public key. Thus, each of them calculates the same key value $\alpha^{m_B m_A} \equiv \alpha^{m_A m_B}$ **mod p**. There does not appear to be any means of obtaining this value without first finding one of the secret exponents ... i.e., solving the discrete logarithm problem for this q. Diffie & Hellman suggest using a value of p which is at least 100 bits long.

Diffie-Hellman Key Exchange

One problem with this protocol is that there is only one key that the two users can use. A variation which permits different keys (called ***session keys***) to be used is:

Public information: prime **p** and generator of Z_p α.

Protocol: A chooses a random secret exponent a, with $0 < a < p-1$.

A computes $\alpha^a \bmod p$ and sends this to B.

B chooses a random secret exponent b, with $0 < b < p-1$.

B computes $\alpha^b \bmod p$ and sends this to A.

A computes the key $K = (\alpha^b)^a \bmod p$ while

B computes the key $K = (\alpha^a)^b \bmod p$.

Man in the Middle Attack

A new problem arises with this variant.

If Oscar is capable of altering the messages between Alice and Bob (an *active* rather than *passive* adversary), then Oscar can change Alice's original α^a to his own α^c and similarly change Bob's original α^b to his own α^c. Now, all communication between Alice and Bob has to go through Oscar in order to be decrypted. That is, a message from Alice is decrypted by Oscar, possibly altered, and then encrypted by Oscar to send to Bob.

This man-in-the-middle attack can be thwarted by the use of a signature scheme. After receiving α^a from Alice, Bob sends the pair $(\alpha^b, \text{sig}_B(\alpha^a,\alpha^b))$ from which Alice can verify that α^b has not been altered from what Bob sent. Alice then sends $\text{sig}_A(\alpha^a,\alpha^b)$ to Bob so that he can verify the α^a that Alice originally sent.

El-Gamal Cryptosystem

For a prime p which is intractible (i.e., very large), let α be a generator of Z_p^*. Each user selects a secret element $a \in Z_{p-1}$ and makes public the value $\beta \equiv \alpha^a \bmod p$. Thus, α, β, and p are publicly known. To send a message, Alice randomly selects a secret $k \in Z_{p-1}$ and if x is the message, sends the ordered pair $(\alpha^k, x\,\beta^k) \bmod p$, where β is Bob's β. To decrypt, Bob raises the first component to his secret exponent a, finds the inverse mod p of this number, and multiplies the second component by this inverse to get the message back. This computation is,

$$\mathbf{(x\,\beta^k)\,(\alpha^{ka})^{-1} = x\,\beta^k\,(\beta^k)^{-1} \equiv x \bmod p.}$$

Shank's Algorithm

This algorithm is known as a ***Time-Memory Trade Off***, that is, if you have enough memory at your disposal you can use it to cut down the amount of time it would normally take to solve the problem.

Let p be a prime, α a generator of Z_p^*. We wish to find a, given β where $\beta \equiv \alpha^a \bmod p$. Let $m = \lceil \sqrt{p-1} \rceil$.

1: Compute $\alpha^{mj} \bmod p$ for $0 \le j \le m-1$.

2: Sort the pairs $(j, \alpha^{mj} \bmod p)$ by second coordinate in a list L_1.

3: Compute $\beta\alpha^{-i} \bmod p$ for $0 \le i \le m-1$.

4: Sort the pairs $(i, \beta\alpha^{-i} \bmod p)$ by second coordinate in a list L_2.

5: Find a pair in each list with the same second coordinate, i.e.,

$$(j, y) \in L_1 \text{ and } (i, y) \in L_2.$$

6: $a \equiv mj + i \bmod (p-1)$.

Pohlig-Hellman Algorithm

There are certain cases in which the discrete logarithm problem can be solved in less than $O(\sqrt{q})$ time, for instance when q-1 has only small prime divisors. An algorithm for dealing with this special case was developed in 1978. We first look at a special case:

Suppose that $q - 1 = 2^n$.

Let α be a primitive element in GF(q). Noting that in this case, q is odd, we have $\alpha^{(q-1)/2} = -1$. Let m, $0 \le m \le q-2$, be the exponent of α that we wish to find, i.e. $c = \alpha^m$, and write m in its binary representation: $m = m_0 + m_1 2 + m_2 2^2 + ... + m_{n-1} 2^{n-1}$. Now,

$$c^{\frac{q-1}{2}} = (\alpha^m)^{\frac{q-1}{2}} = (\alpha^{m_0 + m_1 2 + \cdots + m_{n-1} 2^{n-2}})^{\frac{q-1}{2}} = \alpha^{m_0 \frac{q-1}{2}} = \begin{matrix} 1 \text{ if } m_0 = 0 \\ -1 \text{ if } m_0 = 1 \end{matrix}$$

So the evaluation of $c^{(q-1)/2}$ which costs at most $2 \lceil \log_2 q \rceil$ operations, yields m_0.

Pohlig-Hellman Algorithm

We then determine $c_1 = c\alpha^{-m_0}$, and repeat the basic computation again to obtain m_1.

$$c_1^{\frac{q-1}{2}} = \left(\alpha^{m_1 + m_2 2 + \cdots + m_{n-1} 2^{n-2}}\right)^{\frac{q-1}{2}} = \alpha^{m_1 \frac{q-1}{2}} = \begin{matrix} 1 \text{ if } m_1 = 0 \\ -1 \text{ if } m_1 = 1 \end{matrix}$$

This procedure can then be repeated until each of the m_i are obtained. The total number of operations is thus $n\,(2\lceil \log_2 q \rceil + 2) \approx O\,((\log_2 q)^2)$.

The general case is dealt with by repeating the analogue of the special case for each of the prime factors of q-1 and then combining the results using the Chinese Remainder Theorem.