Quantum algorithms (CO 781, Winter 2008)
Prof. Andrew Childs, University of Waterloo

LECTURE 2: The HSP and Shor's algorithm for discrete log

In this lecture we will discuss the discrete logarithm problem and its relevance to cryptography. We will introduce the general hidden subgroup problem, and show how Shor's algorithm solves a particular instance of it, giving an efficient quantum algorithm for discrete log.

Discrete log Let $G = \langle g \rangle$ be a cyclic group generated by g, written multiplicatively. Given an element $x \in G$, the *discrete logarithm of x in G with respect to g*, denoted $\log_g x$, is the smallest non-negative integer α such that $g^\alpha = x$. The *discrete logarithm problem* is the problem of calculating $\log_g x$.

Here are some simple examples of discrete logarithms:

- For any $G = \langle g \rangle$, $\log_g 1 = 0$
- For $G = \mathbb{Z}_7^\times$, $\log_3 2 = 2$
- For $G = \mathbb{Z}_{541}^\times$, $\log_{126} 282 = 101$

The discrete logarithm seems like a good candidate for a trap-door function. We can efficiently compute g^α, even if α is exponentially large (in $\log |G|$), using repeated squaring. But given x, it is not immediately clear how to compute $\log_g x$, other than by checking exponentially many possibilities. (There are better algorithms, but none is known that works in polynomial time.)

Diffie-Hellman key exchange The apparent hardness of the discrete logarithm problem is the basis of the *Diffie-Hellman key exchange protocol*, the first published public-key cryptographic protocol.

The goal of key exchange is for two distant parties, Alice and Bob, to agree on a secret key using only an insecure public channel. The Diffie-Hellman protocol works as follows:

1. Alice and Bob publicly agree on a large prime p and an integer g of high order. For simplicity, suppose they choose a g for which $\langle g \rangle = \mathbb{Z}_p^\times$ (i.e., a primitive root modulo p). (In general, finding such a g might be hard, but it can be done efficiently given certain restrictions on p.)
2. Alice chooses some $a \in \mathbb{Z}_{p-1}$ uniformly at random. She computes $A := g^a \bmod p$ and sends the result to Bob (keeping a secret).
3. Bob chooses some $b \in \mathbb{Z}_{p-1}$ uniformly at random. He computes $B := g^b \bmod p$ and sends the result to Alice (keeping b secret).
4. Alice computes $K := B^a \bmod p = g^{ab} \bmod p$.
5. Bob computes $A^b \bmod p = g^{ab} \bmod p = K$.

At the end of the protocol, Alice and Bob share a key K, and Eve has only seen p, g, A, and B.

The security of the Diffie-Hellman protocol relies on the assumption that discrete log is hard. Clearly, if Eve can compute discrete logarithms, she can recover a and b, and hence the key. (Note that it is an open question whether, given the ability to break the protocol, Eve can calculate discrete logarithms, thoush some partial results in this direction are known.)

This protocol only provides a means of exchanging a secret key, not of sending private messages. However, very similar ideas can be used to create a public-key cryptosystem (similar in spirit to RSA), known as the ElGamal cryptosystem.

The hidden subgroup problem It turns out that discrete logarithms can be calculated efficiently on a quantum computer, rendering cryptographic protocols such as Diffie-Hellman key exchange insecure. The quantum algorithm for discrete log solves a particular instance of the *hidden subgroup problem* (HSP).

In the general HSP, we are given a black box function $f : G \to S$, where G is a known group and S is a finite set. The function is promised to satisfy

$$\begin{aligned} f(x) = f(y) \text{ if and only if } & x^{-1}y \in H \\ & \text{i.e., } y = xh \text{ for some } h \in H \end{aligned} \tag{1}$$

for some unknown subgroup $H \leq G$. We say that such a function *hides* H. The goal of the HSP is to learn H (say, specified in terms of a generating set) using queries to f.

It's clear that H can in principle be reconstructed if we are given the entire truth table of f. Notice in particular that $f(1) = f(x)$ if and only if $x \in H$: the hiding function is constant on the hidden subgroup, and does not take that value anywhere else.

But the hiding function has a lot more structure as well. If we fix some element $g \in G$ with $g \notin H$, we see that $f(g) = f(x)$ if and only if $x \in gH$, a left coset of H in G with coset representative g. So f is constant on the left cosets of H in G, and distinct on different left cosets.

In the above definition of the HSP, we have made an arbitrary choice to multiply by elements of H on the right, which is why the hiding function is constant on left cosets. We could just as well have chosen to multiply by elements of H on the left, in which case the hiding function would be constant on right cosets; the resulting problem would be equivalent. Of course, in the case where G is abelian, we don't need to make such a choice. For reasons that we will see later, this case turns out to be considerably simpler than the general case; and indeed, there is an efficient quantum algorithm for the HSP in any abelian group, whereas there are only a few nonabelian groups for which efficient algorithms are known.

You should be familiar with Simon's problem, the HSP with $G = \mathbb{Z}_2^n$ and $H = \{0, s\}$ for some $s \in \mathbb{Z}_2^n$. There is a very simple quantum algorithm for this problem, yet one can prove that any classical algorithm for finding s must query the hiding function exponentially many times (in n). The gist of the argument is that, since the set S is unstructured, we can do no better than querying random group elements so long as we do not know two elements x, y for which $f(x) = f(y)$. But by the birthday problem, we are unlikely to see such a collision until we make $\Omega(\sqrt{|G|/|H|})$ random queries.

A similar argument applies to any HSP with a large number of trivially intersecting subgroups. More precisely, we have

Theorem. *Suppose that G has a set $\mathcal{H}$ of N subgroups whose only common element is the identity. Then a classical computer must make $\Omega(\sqrt{N})$ queries to solve the HSP.*

Proof. Suppose the oracle does not a priori hide a particular subgroup, but instead behaves adversarially, as follows. On the ℓth query, the algorithm queries g_ℓ, which we assume to be different from $g_1, \ldots, g_{\ell-1}$ without loss of generality. If there is any subgroup $H \in \mathcal{H}$ for which $g_k \notin g_j H$ for

all $1 \leq j < k \leq \ell$ (i.e., there is some consistent way the oracle could assign g_ℓ to an as-yet-unqueried coset of a hidden subgroup from $\mathcal{H}$), then the oracle simply outputs ℓ; otherwise the oracle conceeds defeat and outputs a generating set for some $H \in \mathcal{H}$ consistent with its answers so far (which must exist, by construction).

The goal of the algorithm is to force the oracle to conceed, and we want to lower bound the number of queries required. (Given an algorithm for the HSP in G, there is clearly an algorithm that forces this oracle to conceed using only one more query.) Now consider an algorithm that queries the oracle t times before forcing the oracle to conceed. This algorithm simply sees a fixed sequence of responses $1, 2, \ldots, t$, so for the first t queries, the algorithm cannot be adaptive. But observe that, regardless of which t group elements are queried, there are at most $\binom{t}{2}$ values of $g_k g_j^{-1}$, whereas there are N possible subgroups in $\mathcal{H}$. Thus, to satisfy the N conditions that for all $H \in \mathcal{H}$, there is some pair j, k such that $g_k g_j^{-1} \in H$, we must have $\binom{t}{2} \geq N$, i.e., $t = \Omega(\sqrt{N})$. $\square$

Note that there are cases where a classical algorithm *can* find the hidden subgroup with a polynomial number of queries. In particular, since a classical computer can easily test whether a certain subgroup is indeed the hidden one, the HSP is easy for a group with only a polynomial number of subgroups. Thus, for example, a classical computer can easily solve the HSP in $\mathbb{Z}_p$ for p prime (since it has only 2 subgroups) and in $\mathbb{Z}_{2^n}$ (since it has only $n+1$ subgroups).

Shor's algorithm Now we will see how Shor's algorithm can be used to calculate discrete logarithms. This is a nice example because it's simpler than the factoring algorithm, but the problem it solves is actually at least as hard: factoring N can be reduced to calculating discrete log in $\mathbb{Z}_N^\times$. (Unfortunately, this does not by itself give a quantum algorithm for factoring, because Shor's algorithm for discrete log in G requires us to know the order of G—but computing $|\mathbb{Z}_N^\times| = \phi(N)$ is as hard as factoring N.)

Recall that we are given some element x of a cyclic group $G = \langle g \rangle$, we would like to calculate $\log_g x$, the smallest integer α such that $g^\alpha = x$. For simplicity, let us assume that the order of G, $N := |G|$, is known. (For example, if $G = \mathbb{Z}_p^\times$, then we know $N = p - 1$. In general, if we do not know N, we can learn it using Shor's period-finding algorithm, but we will not discuss that here.) We can also assume that $x \neq g$ (i.e., $\log_g x \neq 1$), since it is easy to check whether this is the case.

The discrete log problem can be cast as an abelian HSP in the (additive) group $\mathbb{Z}_N \times \mathbb{Z}_N$. Define a function $f : \mathbb{Z}_N \times \mathbb{Z}_N \to G$ as follows:

$$f(\alpha, \beta) = x^\alpha g^{-\beta}. \tag{2}$$

Since $f(\alpha, \beta) = g^{\alpha \log_g x - \beta}$, f is constant on the lines

$$\{(\alpha, \beta) \in \mathbb{Z}_N^2 : \alpha \log_g x - \beta = \text{constant}\}. \tag{3}$$

In other words, it hides the subgroup

$$H = \{(0,0), (1, \log_g x), (2, 2\log_g x), \ldots, (N-1, (N-1)\log_g x)\}. \tag{4}$$

The cosets of H in $\mathbb{Z}_N \times \mathbb{Z}_N$ are of the form $(\gamma, \delta) + H$ with $\gamma, \delta \in \mathbb{Z}_N$. In particular, consider the set of cosets of the form

$$(0, \delta) + H = \{(\alpha, \delta + \alpha \log_g x) : \alpha \in \mathbb{Z}_N\} \tag{5}$$

varying over all $\delta \in \mathbb{Z}_N$ gives a complete set of cosets (so the set $\{0\} \times \mathbb{Z}_N$ is a complete set of coset representatives, i.e., a *transversal of H in $\mathbb{Z}_N \times \mathbb{Z}_N$*).

Shor's algorithm for finding H proceeds as follows. We start from uniform superposition over $\mathbb{Z}_N \times \mathbb{Z}_N$ and compute the hiding function:

$$|\mathbb{Z}_N \times \mathbb{Z}_N\rangle := \frac{1}{N} \sum_{\alpha,\beta \in \mathbb{Z}_N} |\alpha, \beta\rangle \mapsto \frac{1}{N} \sum_{\alpha,\beta \in \mathbb{Z}_N} |\alpha, \beta, f(\alpha, \beta)\rangle. \tag{6}$$

Next we measure the third register. (This step is not essential—in principle, measurements can always be deferred to the end of a quantum algorithm—but it is conceptually helpful.) The post-measurement state is a superposition over group elements consistent with the observed function value, which by definition is some coset of H. In particular, if the measurement outcome is $g^{-\delta}$, we are left with the *coset state* corresponding to $(0, \delta) + H$, namely

$$\mapsto |(0, \delta) + H\rangle := \frac{1}{\sqrt{N}} \sum_{\alpha \in \mathbb{Z}_N} |\alpha, \delta + \alpha \log_g x\rangle \tag{7}$$

Note that the measurement outcome is apparently unhelpful: each possible value occurs with equal probablity; and we cannot obtain δ from $g^{-\delta}$ unless we know how to take discrete logarithms. However, we can exploit the symmetry of the quantum state by performing a QFT over $\mathbb{Z}_N \times \mathbb{Z}_N$:

$$\mapsto \frac{1}{N^{3/2}} \sum_{\alpha,\mu,\nu \in \mathbb{Z}_N} \omega_N^{\mu\alpha + \nu(\delta + \alpha \log_g x)} |\mu, \nu\rangle \tag{8}$$

$$= \frac{1}{N^{3/2}} \sum_{\mu,\nu \in \mathbb{Z}_N} \omega_N^{\nu\delta} \sum_{\alpha \in \mathbb{Z}_N} \omega_N^{\alpha(\mu + \nu \log_g x)} |\mu, \nu\rangle, \tag{9}$$

and using the identity $\sum_{\alpha \in \mathbb{Z}_N} \omega_n^{\alpha\beta} = n\delta_{\beta,0}$, we have

$$= \frac{1}{\sqrt{N}} \sum_{\nu \in \mathbb{Z}_N} \omega_N^{\nu\delta} |-\nu \log_g x, \nu\rangle. \tag{10}$$

Now suppose we measure this state in the computational basis. Then we obtain some pair $(-\nu \log_g x, \nu)$ for uniformly random $\nu \in \mathbb{Z}_N$. If ν has a multiplicative inverse modulo N, we can divide the first register by $-\nu$ to get the desired answer. If ν does not have a multiplicative inverse, we simply repeat the entire procedure again. The probability of success for each independent attempt is $\phi(N)/N = \Omega(1/\log\log N)$, so we don't have to repeat the procedure many times before we find an invertible ν.

This algorithm can be carried out for any cyclic group G so long as we have a unique representation of the group elements, and we are able to efficiently compute products in G. (We need to be able to compute high powers of a group element, but recall that this can be done quickly by repeated squaring.) In the next lecture, we will talk about the application of this algorithm to another kind of group relevant to cryptography, namely the group corresponding to an *elliptic curve* over a finite field.