Discrete Log Based Cryptosystems

Ivan Damgård

November 1, 2004

1 Introduction

This note describes how public-key encryption schemes can be constructed from finite groups with certain properties, and studies some related computational problems that must be hard in order for the cryptosystems to be secure.

2 Preliminaries

Consider the group Z_p^*, where p is a prime. We may think of the prime p as information that *specifies* the group Z_p^*, so that we can work with it on a computer. What we mean by this in general is that knowing the specification of a group, you can

- compute the order
- write down elements in the group, i.e., represent them on our computer as bit strings
- compute the group operation and compute inverses

For Z_p^*, this is clear: if we know p, we know the order of the group $(p-1)$, elements in the group are numbers from 1 to $p-1$, and we can multiply and invert elements modulo p. When in the following, we talk about *generating a group* G, this means running some algorithm to obtain the specification of G. Also, when we say that some algorithm is given G as input, this means it is given the specification.

Any group Z_p^* is cyclic, i.e., there exists some α - a primitive element, or generator - such that $< \alpha >= Z_p^*$. In the following, we wil consider cyclic groups in general, i.e., we are given the specification of some group G and a generator α of G. Throughout, t will be the order of G.

3 Three computational problems

We may now consider the following

The discrete log (DL) problem
Given a group G, generator α, and $\beta \in G$, find integer a, such that $\alpha^a = \beta$.

The DL problem is in many groups notoriously hard, for instance in Z_p^*. A related problem is

The Diffie-Hellman (DH) problem
Given a group G, generator α, and α^a, α^b, where a, b are randomly and independently chosen from Z_t, compute α^{ab}.

Clearly, if we could find a from α^a, we could solve DH by a single exponentiation, so

LEMMA 1 *The DH problem is no harder than the DL problem.*

It is not known if the opposite direction is true in general, but in some groups, the problems are equivalent. Note that the DH problem has a peculiar property, namely if I give you a group element and I claim it solves a DH instance, it is not clear that you can verify that the solution is correct, at least not unless you can solve DL. You would need to decide if, for given $\alpha^a, \alpha^b, \alpha^c$, it holds that $c = ab \bmod t$. This seems to require that you solve DL (although it is NOT clear that this would be necessary). This is the motivation for defining a final related problem. The idea is that you get an instance of the DH problem, plus an extra group element which is either a correct DH solution, or is a random element. You are then supposed to guess which case you are in.

The Decisional Diffie-Hellman (DDH) problem
Given a group G, generator α, and $\alpha^a, \alpha^b, \alpha^c$, where a, b are randomly and

independently chosen from Z_t; and where c is chosen either as $c = ab$, or uniformly random from Z_t. Now guess which of the two cases we are in.

Clearly, if you could solve DH, then you could solve DDH, by computing α^{ab} and comparing this to α^c. So we have:

LEMMA 2 *The DDH problem is no harder than the DH problem.*

There are no types of groups known (other than trivial cases) for which we can show that DDH is equivalent to DH. In fact, there are cases where DDH is known to be easy, but DH is conjectured to be hard. In certain subgroups of Z_p^*, however, DL, DH and DDH all seem to be hard. In the next section, we define more precisely what it means for these problems to be hard.

3.1 Hardness of the problems

There exist several methods by which you can efficently choose a prime p of a particular size (bit length) that you want together with a primitive element. We generalize this notion by assuming that we have a *group generator* $GGen$, i.e., an efficient probabilistic algorithm which takes as input an integer k and outputs a group G and an element $\alpha \in G$ that generates G. The idea is that k controls the size of the group that is generated, and the specification produced allows us to work in G, just as knowing p allows us to work in Z_p^*.

We need this kind of tool to talk about hardness of the DL, DH and DDH problems. The point is that if, for instance, we want to work in groups of the type Z_p^*, it is possible to choose p is such a way that none of our problems are hard. As an example, consider an algorithm that always outputs primes p, where $p-1$ has only small prime factors. There is an algorithm known as Polig-Hellmann that in this case solves DL in Z_p^* efficiently (and hence also DH and DDH). On the other hand, DL does seem to be hard, if we choose *random* primes of length k. So therefore, these problems can only have the kind of hardness we need relative to some algorithm that chooses the group we are in [1].

So hardness of DL w.r.t. $GGen$ means that if we use $GGen$ to make an instance of the DL problem, then any polynomial time algorithm can solve such an instance with probability essentially 0. More formally:

[1] This is more or less the same reason that is also behind the fact that it is meaningless to talk about security of a cryptosystem without taking the algorithm for generating keys into account

DEFINITION 1 *Consider the following experiment with an algorithm A: run GGen on input k to get G and α. Choose a at random in Z_t, and give A input G, α, α^a. The DL problem is said to be hard (with respect to GGen) if for any polynomial (in k) time algorithm A, the probability that A outputs a is negligible in k.*

We can define hardness of DH and DDH in a similar way:

DEFINITION 2 *Consider the following experiment with an algorithm A: run GGen on input k to get G and α. Choose a, b at random in Z_t, and give A input specification of G, α, α^a, α^b. The DH problem is said to hard (with respect to GGen) if for any polynomial (in k) time algorithm A, the probability that A outputs α^{ab} is negligible in k.*

DEFINITION 3 *Consider the following experiment with an algorithm A: run GGen on input k to get G and α. Choose a, b at random in Z_t. Choose δ to be 0 or 1, if $\delta = 0$, set $c = ab$, else choose c at random from Z_t. Give A input specification of G, α, α^a, α^b, α^c. A then outputs one bit, namely its guess at whether δ is 0 or 1. Now, let $p_{A,0}(k)$ be the probability that A outputs 1 when $\delta = 0$, and $p_{A,1}(k)$ be the probability that A outputs 1 when $\delta = 1$. The* advantage *of A is defined to be*

$$Adv_A(k) = |p_{A,0}(k) - p_{A,1}(k)|$$

The DDH problem is said to be hard (with respect to GGen) if for any polynomial (in k) time algorithm A, $Adv_A(k)$ is negligible in k.

Note that hardness of DDH is defined in much the same way we have defined security of cryptosystems, namely it is hard for the adversary to distinguish the "real" from the "ideal" world with non-negligible advantage.

4 The El Gamal Cryptosystem

The motivation for the DH problem is that Diffie and Hellman in 1977 suggested to use it as a basis for exchanging a secret between two parties A and B that share no secret key in advance. The method for this is very simple, assuming that we already agreed (in public) on a group G and a generator α:

1. A chooses s_A at random in Z_t, B chooses s_B at random in Z_t.
2. A sends $y_A = \alpha^{s_A}$ to B, B sends $y_B = \alpha^{s_B}$ to A.
3. A computes $y_B^{s_A}$ and B computes $y_A^{s_B}$

The point is of course that A and B compute the same value in the last step, since $y_B^{s_A} = y_A^{s_B} = \alpha^{s_A s_B}$. Furthermore, an adversary observing the communication would need to solve an instance of the DH problem to find the shared value. El Gamal suggested a way to turn this idea into a regular public key cryptosystem. What we do is essentially to consider A's first message in the above as a part of his public key, then B's part of the protocol can be modified to be an encryption:

El Gamal cryptosystem (general version)

Key generation On input security parameter k, run $GGen$ on input k to obtain specification of a group G and generator α. Choose a at random from Z_t. Then the public key is the specification of G and $\beta = \alpha^a$, while the secret key is a. The plaintext space is G while the ciphertext space is $G \times G$.

Encryption To encrypt $m \in G$, we choose r at random from Z_t, and the ciphertext is $(\alpha^r, \beta^r m)$.

Decryption To decrypt ciphertext (c, d), compute $c^{-a}d$.

To see that decryption works, simply plug in $(\alpha^r, \beta^r m)$ for (c, d) in the decryption algorithm.

Note that there may be some difficulties in applying this scheme in practice: we have said that the plaintext space is the group G. But in real life, we probably want to encrypt some arbitrary piece of data, given as a bit string. It may not be clear how to consider such a string in a unique way as group element. For Z_p^* this is easy, as long as the string is not too long: just think of it as a binary number. Other cases are less obvious - we return to this later.

Given a ciphertext $(\alpha^r, \beta^r m)$, it is clear that you can compute m if and only if you can compute $\beta^r = \alpha^{ar}$, so we have:

LEMMA 3 *The problem of decrypting an El Gamal ciphertext (without the secret key) is equivalent to solving the DH problem.*

However, this does not say much about the semantic security of El Gamal: even if it is hard to compute the entire plaintext, we may be able to compute partial information about it. However, if DDH is hard, then even you are given $\alpha, \alpha^a, \alpha^r$, it is infeasible to distinguish α^{ra} from a random element. This means that for all you care, the last part of the ciphertext that determines m (namely $\alpha^{ra}m$), might as well have been a random element – which would contain no information on m. So in this case, we can expect to have semantic security. More formally:

THEOREM 1 *If the DDH problem is hard (w.r.t. GGen), then the El Gamal cryptosystem is semantically secure.*

Exercise 1 Prove this result. Hint: do it by contradiction. Assume that there exists some adversary *Adv* that can play the semantic security game and have advantage at least ϵ. Use him to construct an algorithm that solves DDH with advantage at least ϵ. Then, if *Adv* is polynomial time and ϵ is not negligible, DDH cannot be hard, and so such an *Adv* cannot exist.

There are also results about El Gamal similar to what is known for RSA:
Exercise 2 RSA is multiplicative, i.e. the product of two encryptions is an encryption of the product of the messages. Formulate and prove a similar property for EL Gamal encryption. Use this to prove the following: given an algorithm A which on input a public El Gamal key $pk = (G, \alpha, \beta)$ and a random ciphertext, outputs the correct plaintext with probability ϵ. Construct an algorithm which on input pk and *any fixed* ciphertext (c, d), decrypts it correctly with probability ϵ.

5 Some Example Groups

5.1 Z_p^*

These are probably the most well-known examples. For this kind of group, a natural candidate for the *GGen* algorithm would be: On input k, choose a random k-bit prime p and an arbitrary generator $\alpha \in Z_p^*$. Choosing a random k-bit prime is something that is well-known from RSA key generation. As for finding a generator, it turns out that there are so many generators, that it will work fine to simply choose random candidates for α until we find a generator. Moreover, if we know the factorization of $p-1$, we can recognize a generator when we see one:

LEMMA 4 *$\alpha \in Z_p^*$ is a generator if and only if $\alpha^{(p-1)/q} \neq 1$ for every prime q that divides $p-1$.*

Fortunately, there are methods for building random primes p in such a way that we also know the factorization of $p-1$. Details of this are out of scope for this note.

Unfortunately, however, if we use $G = Z_p^*$ in El Gamal, we will *never* get a semantically secure cryptosystem. To see this, we need the following basic fact:

LEMMA 5 *Let α be a generator of Z_p^*. Then $(\alpha^i)^{(p-1)/2} \bmod p = 1$ if and only if i is even, and is -1 otherwise.*

PROOF. Clearly $((\alpha^i)^{(p-1)/2})^2 = (\alpha^i)^{p-1} = 1 \bmod p$, which implies that $(\alpha^i)^{(p-1)/2}$ is 1 or -1 – this follows since Z_p is a field, and so the quadratic equation $X^2 = 1 \bmod p$ can have at most 2 solutions. If $i = 2j$, then $(\alpha^i)^{(p-1)/2} = (\alpha^{2j})^{(p-1)/2} = (\alpha^j)^{(p-1)} = 1 \bmod p$. This gives $(p-1)/2$ elements of Z_p that are solutions to $X^{(p-1)/2} = 1 \bmod p$. Again since Z_p is a field, there can be no more, so for all the odd values of i, we must have $(\alpha^i)^{(p-1)/2} \bmod p = -1$. △

This implies:

LEMMA 6 *In the group Z_p^*, the DDH problem is never hard*

PROOF. The adversary is given $\alpha, \alpha^a, \alpha^b, \alpha^c$. By the above, he can easily compute parity of a, b, i.e., whether a, b are even or odd, and hence the parity of ab. He can also find the parity of c. Of course, if $c = ab$, these two parities always match, whereas if c is randomly and indpendently chosen, they will match with probability 1/2. So the adversary compares the parities of ab and c and if they are equal he guesses that he is in case $ab = c$, otherwise he guesses that c was random. Clearly, his advantage is $1 - 1/2 = 1/2$ which certainly is not negligible. △

5.2 Prime Order Subgroups of Z_p^*

To obtain semantic security, we need to use, instead of Z_p^*, a subgroup of large prime order. The reason for this is that the problem we spotted before with the DDH problem really comes from the fact that the order of the group we had $(p-1)$ was divisible by a very small prime, namely 2. This means

that by raising elements to the power $(p-1)/2$, we can "squeeze" things into a very small subgroup (consisting of 1 and -1) where DDH is not hard to handle.

Now, for every prime q that divides $p-1$, there is exactly one subgroup G of order q, and if α_0 generates all of Z_p^*, then $\alpha = \alpha_0^{(p-1)/q}$ generates this subgroup. This follows, sinply from the fact that there are q different multiples $i\frac{p-1}{q}$ of $\frac{p-1}{q}$ between 0 and $p-1$, and so G simply consists of the elements $\{\alpha^1, \alpha^2, ..., \alpha^q\} = \{\alpha_0^{(p-1)/q}, \alpha_0^{2(p-1)/q}, ..., \alpha_0^{(p-1)}\}$.

This leads to the following algorithm for generating G and α:

1. On input k, use known methods to generate a k-bit prime p with known factorization of $p-1$, and generator α_0 of Z_p^*.

2. Let q be the largest prime factor in $p-1$. Let G be the subgroup of order q, and set $\alpha = \alpha_0^{(p-1)/q}$. Output p, q, α.

Then we can use general El Gamal, with these choices of G, α. There is one problem, however: in practice, we want to encrypt some bit string *str*. The plaintext space is really G, and it consists of only some of the numbers in Z_p^*. So we cannot just interpret *str* as a binary number modulo p, most of the time, it will not be in G. The simplest way to solve this problem is to generate primes p of special form, so called *safe primes*. These are primes of form $p = 2q + 1$, where q is also prime. With such a prime, the following encoding enables us to encrypt any number in Z_q. It provides a way to map efficiently and 1-1 from Z_q to G:

- On input $x \in Z_q$, set $y = x + 1$ and compute $y^{(p-1)/2} \bmod p$. If this is 1, output y, else output $-y \bmod p$.

When we decrypt, we get an element in G, and we can then bring this back to Z_q as follows:

- On input $g \in G \subset Z_p^*$, test if $g \leq q$. If so, set $y = g$, else set $y = -g \bmod p$. Output $y - 1$.

Exercise 3 Prove that when $p = 2q + 1$, the subgroup G of order q in Z_p^* consists of all even powers of α. Use this and Lemma 5 to prove that the above encoding method works, i.e., the mapping introduced is 1-1, and maps to the subgroup G.

5.3 Other Examples

As G, we can also use the multiplicative group in some finite field, or (more interesting) Elliptice Curve groups. The interesting point about them is that only generic algorithms are known to solve DL in these groups. Such groups are typically constructed from a prime p, and it is known that the group you get has order approximately p. This means that a generic algorithm will need to do approximately $\sqrt{p}$ group operations to solve DL (or DH), which implies that p should be a 200-300 bit prime with current state of the art. This compares favorably with the 1000 or more bits one would need if we used Z_p^* directly, so that keys can be shorter, and (in some cases) the encryption/decryption faster. On the other hand, the key generation is considerably more complex.

An interesting special case of elliptic curve groups are those constructed from so called supersingular curves. The groups we get from these curves are special in that, while the DL and DH problems are believed to be hard in these groups, when their order are around 2^{1000}, the DDH problem is easy. It turns out to be possible to use this property for many interesting cryptographic constructions, although of course standard El-Gamal encryption is no good here.