CS 395T Advanced Cryptography 22-Jan-2009

Lecture 2: Class Introduction

Instructor:Brent Waters *Scribe: Rashid Kaleem*

1 Public Key Encyption

The following scenario describes a public key encryption system in practice.

1. Bob decides to communicate with alice@yahoo.com but fears an attacker.
2. Bob would lookup Alice's public key from a trusted source.
3. Bob would encrypt the email using the public key aquired in the above step.
4. Alice would have the private key which she can use to decrypt the message.

In an identity based encryption, the first step is not required. Today we will review some basic number theory and public key cryptosystems such as the El Gammal and prove it secure under a number theoretic assumption.

2 Groups

A group is a finite set of elements having the following properties

- The order of the group is the number of elements in it.
- The group is closed under an operation (addition, multiplication etc)
- The existence of an Identitiy element I such that $\forall h \in G, I * h = h * I = h$.
- Existence of an inverse for all elements, $\forall h \in G, \exists h^{-1}$ such that $h * h^{-1} = I$
- The operation under which the group is defined be commutative and associative.

For example, the following are groups:

- Z_P^* : for some prime P, is the set of elements $\{1, 2, 3, ...P-1\}$ under the operation multiplication. The order of the group is $P-1$.
- Z_7: consists of $\{1, 2, 3, 4, 5, 6\}$. For instance, $5 * 5 mod 7 = 25 mod 7 = 4$.
  The inverse can be derived similarly, for instance 3^{-1} is represented by 5 since $3 * 5 mod 7 = 15 mod 7 = 1$
- $G = \{1, 2, 4\}$ is a group under the operation multiplication modulo 7.
- $G = \{1, 2, 4, 6\}$ is not a group under the operation multiplication modulo 7 because it does not obey the closure property.
- Elliptic Curve groups. More on this later, but they permit an equal strength cryptosecurity for smaller keys.

2.1 Some usefull facts.

Let G be a group of prime order P, and let $g \in G$

1. $g^a = g * g * g...g$
2. $g^a * g^b = g^{a+b}$. Not g^{ab}.
3. $g^P = 1$, where P is the group order.
4. $g^a = g^{amodP}$.

2.2 Repeated Squaring for Exponentiation.

In practice, $\|G\| \approx 2^{160}$, and to compute g^a for very very large a would require extensive computing. It would take exponential time to compute g^a using regular multiplication a times.
In order to compute g^a more efficiently, we use the following procedure:

1. Detemine $n = loga_2$.
2. Compute $g^{2i} = g^i * g^i$ for $i = 1, 2, 4, ...n$.
3. Let the binary representation of a be $a_n, a_{n-1}, ...a_2, a_1, a_0$.
4. Now use the following to determine g^a :

$$g^a = (g^1)^{a_1} + (g^2)^{a_2} + ... + (g^{2^n})^{a_n}$$

3 Hard Problems

Now we shall consider some hard problems, as well as show how to prove a cryptosystem secure. We will consider three hard problems :

- Discrete Logarithm
- Computational Diffie Hellman
- Decisional Diffie Hellman

Now let us discuss each of these:

3.1 Discrete Logarithm

Given a group G of order P and a generator g, and let a be randomly chosen from Z_P. Now consider $g \in G$. Let $h = g^a$. The discrete logarithm problem T is to determince a from g and g^a.

3.1.1 Assumption

For all poly-time algorithms A,

$$Pr[A(T(G, g, g^a)) = a] = \textit{negligible}$$

But we need to be specific by what we mean negligible and plynomial time algorithm with respect to what. Whenever we have a number theoretic assumption, it is always going to be parameterized with respect to a security parameter λ, so the algorithm A is polynomial time with respect to λ. We also define a function F to be negligile in λ if for all plynomials $g(x)$, $\exists n$ such that $\forall x' > n$

$$f(x') < \frac{1}{g(x')}$$

Hence, as λ becomes larger, the probability of solving the problem becomes smaller.

3.2 Computational Diffie Hellman

Given a group G of order P and a generator g, and let a, b be randomly chosen from Z_P. Now consider $g \in G$. The computational Diffie Hellman problem T is to determince g^{ab} from g, g^a, and g^b.
No polynomial time algorithm can compute g^{ab} without negligible probability.

3.2.1 Proof of Hardness

We can show that a problem A is harder than a problem B if we can show that solving B can be used to solve A as well. In order to show that the Computational Diffie Hellman problem is harder than the Discrete Logarithm problem by showing that a solution to the latter can be used to solve the former.

Suppose that an algorithm (or procedure) A which can solve the Discrete Logarithm problem i.e, $A(g, g^a) = a$ with greater than negligible probability. We can construct an algorithm B that solves the computational Diffie Hellman problem :

Theorem 3.1 *Computational Diffie Hellman is harder than Discrete Logarithm*

Proof. We can construct an algorithm $B(g, g^a, g^b)$ that can compute g^{ab} as follows:

- Invoke $A(g, g^a) = a$, hence determine a.
- Simply compute $(g^b)^a$ to determine g^{ab}.

But the above assumption requires the following conditions:

1. A is a polynomial time algorithm.
2. A solves the Discrete Logarithm problem with greater than negligible probability.
3. B takes polynomial time (and hence invokes A polynomial number of times).
4. Probability of success for B is equal to that of A.

□

3.3 Decisional Diffie Hellman

Given a group G of order P and a generator g, and let a, b be randomly chosen from Z_P, also consider a randomly selected $\beta \in 0, 1$ such that if $\beta = 0$ $\chi = g^{ab}$ else χ is randomly selected from G. The decisional Diffie Hellman problem T is to determince β from g, χ, g^a, and g^b (i.e, is it a valid Diffie Hellman tuple?).

Note that β is uniformly random, so the adversary can also randomly pick his guess for β and get it correct half of the time. So we need to rephrase our definition of the hardness.

For all plynomial time algorithms A, the probability that $A(G, g, g^a, g^b, \chi) = \beta$ is less than $\frac{1}{2} + negligible(\lambda)$, i.e;

$$Pr[A(g, g^a, g^b, \chi) = \beta] < \frac{1}{2} + negligible(\lambda)$$

Theorem 3.2 *Decisional Diffie Hellman is harder than Computational Diffie Hellman*

Proof. If algorithm A breaks computational Diffie Hellman, then there exists an algorithm B that can break decisional Diffie Hellman algorithm as follows:

- B is passed the arguments $g, g^a, g^b and \chi$.
- B invokes A with the arguments g, g^a and g^b which returns χ'.
- B can check if χ' equals χ and outputs 0 else outputs 1.

□