MIT 6.875

Foundations of Cryptography
Lecture 9

Lectures 8-10

- Key Agreement and Public-key Encryption:
  Definition and Properties

- Constructions

  1: Diffie-Hellman/El Gamal

  2: Trapdoor Permutations (RSA)

  3: Quadratic Residuosity/Goldwasser-Micali

  4: Learning with Errors/Regev

The Multiplicative Group $\mathbb{Z}_N^*$

$$= \{1 \le x < \mathrm{N} : \gcd(\mathrm{x}, \mathrm{N}) = 1\}$$

Theorem: $\mathbb{Z}_N^*$ is a group under multiplication mod N.

Inverses exist: since $\gcd(\mathrm{x}, \mathrm{N}) = 1$, there exist integers a and b s.t.

$$ax + bN = 1 \text{ (Bezout's identity)}$$

Thus, $ax = 1\ (mod\ N)$ or $a = x^{-1}\ (mod\ N)$.

The Multiplicative Group $\mathbb{Z}_N^*$

$$= \{1 \le x < \mathrm{N}: \gcd(\mathrm{x}, \mathrm{N}) = 1\}$$

Theorem: $\mathbb{Z}_N^*$ is a group under multiplication mod N.

Order of $\mathbb{Z}_N^*$ = Euler's totient function $\varphi(N)$.

$\varphi(P) = P - 1$ if P prime.

$\varphi(N) = (P-1)(Q-1)$ if $N = PQ, P \neq Q$ primes.

$\varphi(N) = \prod P_i^{\alpha_i - 1}(P_i - 1)$ if $N = \prod P_i^{\alpha_i}$.

Theorem [Lagrange, Euler]:

For every a $\in \mathbb{Z}_N^*$, $a^{\varphi(N)} = 1 \; mod \; N$.

Examples

$$\mathbb{Z}_2^* = \{1\}$$

$$\mathbb{Z}_3^* = \{1, 2\}$$

$$\mathbb{Z}_4^* = \{1, 3\}$$

$$\mathbb{Z}_5^* = \{1, 2, 3, 4\}$$

$$\mathbb{Z}_6^* = \{1, 5\}$$

$$\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6, 7\}$$

The Multiplicative Group $\mathbb{Z}_p^*$

$\mathbb{Z}_p^*$: ($\{1, \dots, \mathrm{p} - 1\}$, group operation: $\cdot$ mod p)

- Computing the group operation is easy.
- Computing inverses is easy: Extended Euclid.
- Exponentiation (given $g \in \mathbb{Z}_p^*$ and $x \in \mathbb{Z}_{p-1}$, find g^x mod p) is easy: **Repeated Squaring Algorithm.**
- The discrete logarithm problem (given a generator g and $h \in \mathbb{Z}_p^*$, find $x \in \mathbb{Z}_{p-1}$ s.t. $\mathrm{h} = g^x$ mod p) is **hard**, to the best of our knowledge!

The Discrete Log Assumption

The discrete logarithm problem is: given a generator g and $h \in \mathbb{Z}_p^*$, find $x \in \mathbb{Z}_{p-1}$ s.t. $\mathrm{h} = g^x$ mod p.

Distributions…

1. Is the discrete log problem hard for a random p? Could it be easy for some p?
2. Given p: is the problem hard for all generators g?
3. Given p and g: is the problem hard for all x?

Random Self-Reducibility of DLOG

Theorem: If there is an p.p.t. algorithm A s.t.

$$\Pr[A(p, g, g^x \bmod p) = x] > 1/\text{poly}(\log p)$$

for some p, random generator g of $\mathbb{Z}_p^*$, and random x in $\mathbb{Z}_{p-1}$, then there is a p.p.t. algorithm B s.t.

$$B(p, g, g^x \bmod p) = x$$

for all g and x.

Proof: On the board.

Random Self-Reducibility of DLOG

Theorem: If there is an p.p.t. algorithm A s.t.

$$\Pr[A(p, g, g^x \bmod p) = x] > 1/\mathrm{poly}(\log p)$$

for some p, random generator g of $\mathbb{Z}_p^*$, and random x in $\mathbb{Z}_{p-1}$, then there is a p.p.t. algorithm B s.t.

$$B(p, g, g^x \bmod p) = x$$

for all g and x.

2. Given p: is the problem hard for all generators g?

   … as hard for any generator is it for a random one.

3. Given p and g: is the problem hard for all x?

   … as hard for any x is it for a random one.

Algorithms for Discrete Log

- Baby Step-Giant Step algorithm: time --- and space --- $O(\sqrt{p})$.

- Pohlig-Hellman algorithm: time $O(\sqrt{q})$ where q is the largest prime factor of $p-1$. That is, there are dlog-easy primes.

The Discrete Log (DLOG) Assumption

W.r.t. a random prime: for every p.p.t. algorithm A, there is a negligible function μ s.t.

$$\Pr\left[\begin{array}{c} p \leftarrow PRIMES_n; g \leftarrow GEN\left(\mathbb{Z}_p^*\right); \\ x \leftarrow \mathbb{Z}_{p-1}: A(p, g, g^x \bmod p) = x \end{array}\right] = \mu(n)$$

One-way Permutation (Family)

$$F(p, g, x) = (p, g, g^x \bmod \mathrm{p})$$

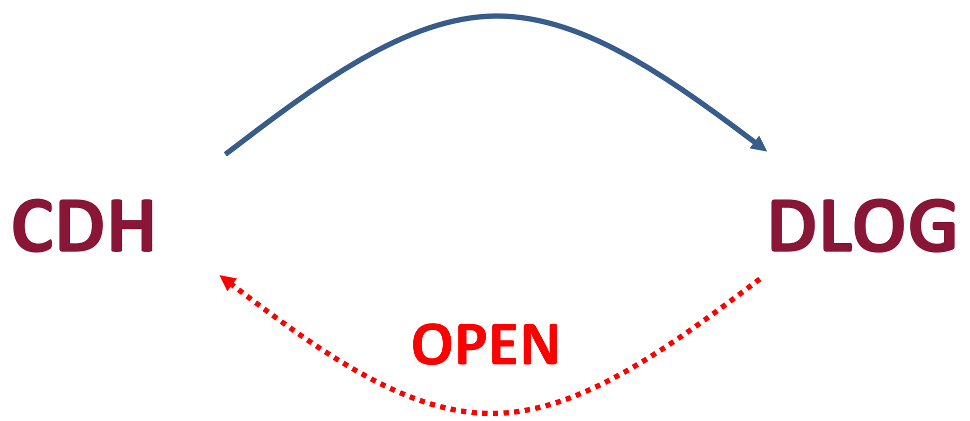
$\mathcal{F}_n = \{F_{n,p,g}\}$ where $F_{n,p,g}(x) = (p, g, g^x \bmod \mathrm{p})$

Theorem: Under the discrete log assumption, F is a one-way permutation (resp. $\mathcal{F}_n$ is a one-way permutation family).

Computational Diffie-Hellman (CDH) Assumption

W.r.t. a random prime: for every p.p.t. algorithm A, there is a negligible function μ s.t.

$$\Pr\left[\begin{array}{c} p \leftarrow PRIMES_n; g \leftarrow GEN\left(\mathbb{Z}_p^*\right); \\ x, y \leftarrow \mathbb{Z}_{p-1}: A(p, g, g^x, g^y) = g^{xy} \end{array}\right] = \mu(n)$$



Diffie-Hellman Key Exchange

p, g: Generator of our group Z_p^*

$g^x \bmod p$ →

← $g^y \bmod p$

Pick a random number $x \in Z_{p-1}$

Pick a random number y $\in Z_{p-1}$

Shared key K = $g^{xy} \bmod p$

= $(g^y)^x \bmod p$

Shared key K = $g^{xy} \bmod p$

= $(g^x)^y \bmod p$

Diffie-Hellman/El Gamal Encryption

- $Gen(1^n)$: Generate an n-bit prime p and a generator g of Z_p^*. Choose a random number $x \in Z_{p-1}$

  Let $pk = (p, g, g^x)$ and let $sk = x$.

- $Enc(pk, m)$ where $m \in Z_p^*$: Generate random $y \in Z_{p-1}$ and output $(g^y, g^{xy} \cdot m)$

- $Dec(sk = x, c)$: Compute g^{xy} using g^y and x and divide the second component to retrieve m.

How to make this really work?

Is this Secure?

How to come up with a prime p

(1) **Prime number theorem**: $\approx 1/n$ fraction of $\boldsymbol{n}$**-bit** numbers are prime.

(2) **Primality tests** [Miller'76, Rabin'80, Agrawal-Kayal-Saxena'02] Can test in time $\mathrm{poly}(n)$ if a given n-bit number is prime.

OPEN: Deterministically come up with an n-bit prime?

How to come up with a generator g

(1) **There are lots of generators**: $\approx 1/\log n$ fraction of $\mathbb{Z}_p^*$ are generators (where p is an n-bit prime).

(2) **Testing if g is a generator**:

<u>Theorem</u>: let $q_1, \ldots, q_k$ be the prime factors of $p-1$. Then, g is a generator of $\mathbb{Z}_p^*$ if and only if

$$g^{(p-1)/q_i} \neq 1 \pmod{p} \text{ for all i.}$$

OPEN: Can you test if g is a generator without knowing the prime factorization of p-1?

OPEN: Deterministically come up with a generator?

To Summarize

- Pick a random prime p *together with* the prime factorization of p-1 (How? Adam Kalai 2000 paper)
- Pick a random element of $\mathbb{Z}_p^*$ and test if it is a generator (using theorem from last slide).
- Continue step 2 until you hit a generator.

- We will see another, more commonly used method, soon.

Squares mod P

Let P be prime. $x \in Z_P^*$ is a squares mod P (also called a "quadratic residue") if there is a y $\in Z_P^*$ s.t.

$$x = y^2 \bmod P.$$

Theorem: Exactly half of Z_P^* are squares mod P.

Squares mod P: A Characterization

Claim: Fix any generator g. Then, $x \in \mathbb{Z}_P^*$ is a square iff $DLOG_g(x) \bmod p-1$ is even.

Proof (*if*)
If $x = g^a \bmod P$ and a is even, then $g^{a/2} \bmod P$ is a square root of x.

Proof (*iff*)
If $x = g^a = (g^b)^2 \bmod P$, then $a = 2b \pmod{P-1}$. So, a is even.

Now, an Efficient Characterization…

Claim: x mod P is a square iff $x^{(P-1)/2} = 1 \bmod P$

Proof (*iff*) If $x = y^2 \bmod P, x^{(P-1)/2} = y^{(P-1)} = 1\ mod\ P$.

Proof (*if*) Show that the discrete log of x has to be even and therefore (by previous slide) x is a square.

So, it is easy to detect whether a number mod P is a square.

The Problem

Claim: Given p, g, $g^x \bmod p$ and $g^y \bmod p$, adversary can compute some information about $g^{xy} \bmod p$.

determine if $g^{xy} \bmod p$ is a square mod p.

Corollary: Therefore, additionally given $g^{xy} \cdot m \bmod p$, the adversary can determine whether m is a square mod p, violating "IND-CPA security".

The Problem

Claim: Given p, g, $g^x \bmod p$ and $g^y \bmod p$, adversary can determine if $g^{xy} \bmod p$ is a square mod p.

$g^{xy} \bmod p$ is a square $\Leftrightarrow$ $xy \pmod{p-1}$ is even

$\Leftrightarrow$ xy is even

$\Leftrightarrow$ x is even or y is even

$\Leftrightarrow$ $x\ (mod\ p-1)$ is even or $y \pmod{\mathrm{p}-1}$ is even

$\Leftrightarrow$ $g^x\ mod\ p$ or $g^y\ mod\ p$ is a square

This can be checked in poly time!

Diffie-Hellman/El Gamal Encryption

Claim: Given p, g, $g^x \bmod p$ and $g^y \bmod p$, adversary can determine if $g^{xy} \bmod p$ is a square mod p.

More generally, dangerous to work with groups that have non-trivial subgroups (in our case, the subgroup of all squares mod p)

Lesson: Best to work over a group of prime order. Such groups have no non-trivial subgroups.

An Example: Let $p = 2q + 1$ where q is a prime itself. Then, the group of squares mod p has order $\frac{(p-1)}{2} = q$.

Diffie-Hellman/El Gamal Encryption

- $Gen(1^n)$: Generate an n-bit "safe" prime $p = 2q + 1$ and a generator g of Z_p^* and let $h = g^2 \bmod p$ be a generator of QR_p . Choose a random number $x \in Z_q$.

  Let $pk = (p, h, h^x)$ and let $sk = x$.

- $Enc(pk, m)$ where $m \in QR_p$: Generate random $y \in Z_q$ and output $(g^y, g^{xy} \cdot m)$

- $Dec(sk = x, c)$: Compute g^{xy} using g^y and x and divide the second component to retrieve m.

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption (DDHA):

Hard to distinguish between g^{xy} and a uniformly random group element, given g, g^x and g^y

That is, the following two distributions are computationally indistinguishable:

$$(g, g^x, g^y, g^{xy}) \approx (g, g^x, g^y, u)$$

DH/El Gamal is IND-secure under the DDH assumption on the given group.

Which Group to Use?

(1) QR_P for a safe prime $\mathrm{P} = 2Q + 1$ where Q is prime. The order of the group is Q.

Discrete log can be broken in *sub-exponential* time $2^{\sqrt{\log P \log\log P}}$ (better than $\mathrm{poly}(P)$ but worse than $\mathrm{poly}(\log P)$.)

(2) Elliptic Curve Groups. The set of solutions (x, y) to the equation $y^2 = x^3 + ax + b$ (mod P) together with a very cool group addition law.

Best known Discrete log algorithm: $O(\sqrt{P})$ time!

Much smaller keys: 160-bit P suffices for "80-bit security".