CSL759: Cryptography and Computer Security

Ragesh Jaiswal

CSE, IIT Delhi

The Factoring Problem

The Factoring Problem

- We would like to understand the success of polynomial time algorithms in factoring integers. We formally define this in terms of an experiment:
- Experiment Factor(A, n)
  - Choose two n-bit primes x_1 and x_2 at random.
  - Compute $N = x_1 \cdot x_2$
  - Adversary A is given N and let it output (x_1', x_2').
  - If $(x_1' \cdot x_2' = N)$ then output 1 else output 0.
- How do we randomly generate an n-bit prime number?

The Factoring Problem

- How do we randomly generate an n-bit prime number?
- GRP(1^n)
  - For $i = 1$ to t
    - Randomly pick $p' \in \{0,1\}^{n-1}$
    - $p \leftarrow 1|p'$
    - If (p is prime) then output p
  - Output "fail"
- What is the probability (in terms of t) that the above algorithm outputs a prime number?

The Factoring Problem

- How do we randomly generate an n-bit prime number?
- GRP(1^n)
  - For $i = 1$ to t
    - Randomly pick $p' \in \{0,1\}^{n-1}$
    - $p \leftarrow 1|p'$
    - If (p is prime) then output p
  - Output "fail"
- What is the probability (in terms of t) that the above algorithm outputs a prime number?
- Theorem (Prime Number Theorem): There exists a constant c such that for any $n > 1$, the number of n bit primes is at least $c \cdot \frac{2^{n-1}}{n}$.

The Factoring Problem

- How do we randomly generate an n-bit prime number?
- GRP(1^n)
  - For $i = 1$ to t
    - Randomly pick $p' \in \{0,1\}^{n-1}$
    - $p \leftarrow 1|p'$
    - If (p is prime) then output p
  - Output "fail"
- Problem(Primality Testing): Given an integer $N > 1$, how do we check that it is prime or not?

The Factoring Problem

- How do we randomly generate an n-bit prime number?
- GRP(1^n)
  - For $i = 1$ to t
    - Randomly pick $p' \in \{0,1\}^{n-1}$
    - $p \leftarrow 1|p'$
    - If (p is prime) then output p
  - Output "fail"
- Problem(Primality Testing): Given an integer $N > 1$, how do we check that it is prime or not?
  - There is a randomized algorithm (Miller-Rabin) with one-sided error when the given number is composite. This algorithm runs very fast.
  - There is a polynomial time deterministic algorithm (AKS) too but it runs slower than the randomized algorithm.

Primality Testing

Miller-Rabin

Miller-Rabin Primality Test

- Theorem 1: The Miller-Rabin algorithm takes as input an integer N and a parameter t. If N is prime, then the algorithms outputs "prime" with probability 1. If N is composite, then the algorithm outputs "prime" with probability at most 2^{-t}. Moreover, the algorithm runs in time polynomial in t and the size of N.

The Factoring Problem

- We would like to understand the success of polynomial time algorithms in factoring integers. We formally define this in terms of an experiment:
- GenModulus(1^n)
  - Run GRP(1^n) to obtain p, q. Let $N = p \cdot q$. Return (N, p, q).
- Experiment Factor(A, GRP, n)
  - Run GenModulus(1^n) to obtain (N, p, q).
  - Compute $N = p \cdot q$
  - Adversary A is given N and let it output (p', q').
  - If $(p' \cdot q' = N)$ then output 1 else output 0.
- We say that factoring is hard w.r.t. GenModulus(1^n) if for all PPT algorithms A, there exists a negligible function $negl$ such that

$$\Pr[Factor(A, GRP, n) = 1] \leq negl(n)$$

The RSA Problem

- We would like to understand the success of polynomial time algorithms in solving the RSA problem. We formally define this in terms of an experiment.
- GenRSA(1^n)
  - Run GenModulus(1^n) to obtain (N, p, q).
  - Let $\phi(N) = (p-1) \cdot (q-1)$.
  - Find e such that $\gcd\big(e, \phi(N)\big) = 1$.
  - Compute $d = [e^{-1} \ (mod\ \phi(N))]$.
  - Return (N, e, d)
- Experiment RSA-inv$(A, GenRSA, n)$
  - Run GenRSA(1^n) to obtain (N, e, d).
  - Choose $y \leftarrow Z_N^*$.
  - A is given N, e, y, and outputs $x \in Z_N^*$.
  - If $(x^e \equiv y \ (mod\ N))$, then output 1 else output 0.

The RSA Problem

- GenRSA(1^n)
  - Run GenModulus(1^n) to obtain (N, p, q).
  - Let $\phi(N) = (p-1) \cdot (q-1)$.
  - Find e such that $\gcd(e, \phi(N)) = 1$.
  - Compute $d = [e^{-1} \ (mod\ \phi(N))]$.
  - Return (N, e, d)
- Experiment RSA-inv$(A, GenRSA, n)$
  - Run GenRSA(1^n) to obtain (N, e, d).
  - Choose $y \leftarrow Z_N^*$.
  - A is given N, e, y, and outputs $x \in Z_N^*$.
  - If $(x^e \equiv y \ (mod\ N))$, then output 1 else output 0.
- We say that the RSA problem is hard relative to GenRSA if for all PPT algorithms A, there exists a negligible function $negl$ such that $\Pr[RSA - inv(A, GenRSA, n) = 1] \leq negl(n)$.

Factoring Vs RSA

Factoring Vs RSA

- Experiment RSA-inv$(A, GenRSA, n)$
  - Run GenRSA(1^n) to obtain (N, e, d).
  - Choose $y \leftarrow Z_N^*$.
  - A is given N, e, y, and outputs $x \in Z_N^*$.
  - If $(x^e \equiv y \ (mod\ N))$, then output 1 else output 0.

- Experiment Factor$(A,$ GRP, $n)$
  - Run GenModulus(1^n) to obtain (N, p, q).
  - Compute $N = p \cdot q$
  - Adversary A is given N and let it output (p', q').
  - If $(p' \cdot q' = N)$ then output 1 else output 0.

- Which is the harder problem?

Factoring Vs RSA

- Experiment RSA-inv$(A, GenRSA, n)$
  - Run GenRSA(1^n) to obtain (N, e, d).
  - Choose $y \leftarrow Z_N^*$.
  - A is given N, e, y, and outputs $x \in Z_N^*$.
  - If $(x^e \equiv y \ (mod\ N))$, then output 1 else output 0.

- Experiment Factor$(A,$ GRP, $n)$
  - Run GenModulus(1^n) to obtain (N, p, q).
  - Compute $N = p \cdot q$
  - Adversary A is given N and let it output (p', q').
  - If $(p' \cdot q' = N)$ then output 1 else output 0.

- Which is the harder problem?
- Claim: If the RSA problem is hard w.r.t. GenRSA, then the factoring problem is hard w.r.t. GenModulus.
- Is the converse also true?
  - Not known.

Cyclic Groups and Diffie-Hellman

Cyclic Groups

- Let G be a finite group of order m.
- For any element $g \in G$, consider the set $< g > = \{g^0, g^1, \dots, \}$.
- Since $g^m = 1$, we know that $< g > = \{g^0, \dots, g^{m-1}\}$.
- Let $i \leq m$ be the smallest integer such that $g^i = 1$. Then $< g > = \{g^0, g^1, \dots, g^{i-1}\}$.
- Lemma 1: $| < g > | = i$.
- Lemma 2: $< g >$ is a subgroup of G.
  - $< g >$ is called the subgroup generated by g.
  - The order of $< g >$ is called the order of g in short.

Cyclic Groups

- Let G be a finite group of order m.
- For any element $g \in G$, consider the set $< g > = \{g^0, g^1, \dots, \}$.
- Since $g^m = 1$, we know that $< g > = \{g^0, \dots, g^{m-1}\}$.
- Let $i \leq m$ be the smallest integer such that $g^i = 1$. Then $< g > = \{g^0, g^1, \dots, g^{i-1}\}$.
- Lemma 1: $| < g > | = i$.
- Lemma 2: $< g >$ is a subgroup of G.
  - $< g >$ is called the subgroup generated by g.
  - The order of $< g >$ is called the order of g in short.
- Lemma 3: Let $g \in G$ be aby element of order i. Then for any integer x, we have $g^x = g^{[x \ (mod \ i)]}$.

Cyclic Groups

- Let G be a finite group of order m.
- For any element $g \in G$, consider the set $< g > = \{g^0, g^1, \dots, \}$.
- Since $g^m = 1$, we know that $< g > = \{g^0, \dots, g^{m-1}\}$.
- Let $i \leq m$ be the smallest integer such that $g^i = 1$. Then $< g > = \{g^0, g^1, \dots, g^{i-1}\}$.
- Lemma 1: $| < g > | = i$.
- Lemma 2: $< g >$ is a subgroup of G.
  - $< g >$ is called the subgroup generated by g.
  - The order of $< g >$ is called the order of g in short.
- Lemma 3: Let $g \in G$ be any element of order i. Then for any integer x, we have $g^x = g^{[x \ (mod \ i)]}$.
- Lemma 4: Let i be the order of an element $g \in G$. Then $i | m$.

Cyclic Groups

- Let G be a finite group of order m.
- For any element $g \in G$, consider the set $< g > = \{g^0, g^1, \dots, \}$.
- Since $g^m = 1$, we know that $< g > = \{g^0, \dots, g^{m-1}\}$.
- Let $i \leq m$ be the smallest integer such that $g^i = 1$. Then $< g > = \{g^0, g^1, \dots, g^{i-1}\}$.
- Lemma 1: $| < g > | = i$.
- Lemma 2: $< g >$ is a subgroup of G.
  - $< g >$ is called the subgroup generated by g.
  - The order of $< g >$ is called the order of g in short.
- Lemma 3: Let $g \in G$ be any element of order i. Then for any integer x, we have $g^x = g^{[x \ (mod \ i)]}$.
- Definition (Cyclic group and generator): If there an element $g \in G$ of order m, then G is called a cyclic group and say that g is a generator of G.

Cyclic Groups

- Definition (Cyclic group and generator): If there an element $g \in G$ of order m, then G is called a cyclic group and say that g is a generator of G.
- Lemma 4: Let G be a group of order m. Let i be the order of an element $g \in G$. Then $i|m$.
- Lemma 5: If G is a group of prime order, then G is cyclic. Furthermore, all elements of G except the identity are generators of G.
- Theorem 6: If p is prime, then Z_p^* is cyclic.

Discrete Logarithm Problem

Discerete Logarithm Problem

- Let G be a cyclic group of order q and let g be a generator.
- For every $h \in G$, there exists $x \in Z_q$ such that $g^x = h$.
- x is called the discrete logarithm of h with respect to g. This is denoted as $x = \log_g h$.
- Lemma 1: $\log_g 1 = 0$.
- Lemma 2: $\log_g(h_1 \cdot h_2) = [(\log_g h_1 + \log_g h_2)(mod\ q)]$.
- We now formally define the Discrete Logarithm problem.

Discerete Logarithm Problem

- Let Gen be an algorithm that takes as input 1^n and outputs a cyclic group G of order q such that $|q| = n$ and a generator g for G.
- Experiment Dlog$(A, Gen\ n)$
  - Run $Gen(1^n)$ to obtain (G, q, g).
  - Choose $h \leftarrow G$ uniformly at random.
  - A is given G, q, g, h and let its output be $x \in Z_q$.
  - If $(g^x = h)$, then output 1 else output 0.
- We say that the discrete logarithm problem is hard relative to Gen if for all PPT algorithms A, there exists a negligible function $negl$ such that $\Pr[DLog(A, Gen, n) = 1] \leq negl(n)$.

Discerete Logarithm Problem

- Let Gen be an algorithm that takes as input 1^n and outputs a cyclic group G of order q such that $|q| = n$ and a generator g for G.
- Experiment Dlog$(A, Gen\ n)$
  - Run $Gen(1^n)$ to obtain (G, q, g).
  - Choose $h \leftarrow G$ uniformly at random.
  - A is given G, q, g, h and let its output be $x \in Z_q$.
  - If $(g^x = h)$, then output 1 else output 0.
- We say that the discrete logarithm problem is hard relative to Gen if for all PPT algorithms A, there exists a negligible function $negl$ such that $\Pr[DLog(A, Gen, n) = 1] \leq negl(n)$.
- Is there a Gen w.r.t. which discrete log is hard?

Discerete Logarithm Problem

Index Calculus

Let p be a prime and $G = \mathbf{Z}_p^*$. Then there is an algorithm that finds discrete logs in G in time

$$e^{1.92(\ln p)^{1/3}(\ln \ln p)^{2/3}}$$

This is sub-exponential, and quite a bit less than

$$\sqrt{p} = e^{(\ln p)/2}$$

Note: The actual running time is $e^{1.92(\ln q)^{1/3}(\ln \ln q)^{2/3}}$ where q is the largest prime factor of $p-1$, but we chose p so that $q \approx p$, for example $p - 1 = 2q$ for q a prime.

Discerete Logarithm Problem

Elliptic Curve Groups

Let G be a prime-order group of points over an elliptic curve. Then the best known algorithm to compute discrete logs takes time

$$O(\sqrt{p})$$

where $p = |G|$.

Discerete Logarithm Problem

Comparison

Say we want 80-bits of security, meaning discrete log computation by the best known algorithm should take time 2^{80}. Then

- If we work in $\mathbf{Z}_p^*$ (p a prime) we need to set $|\mathbf{Z}_p^*| = p - 1 \approx 2^{1024}$
- But if we work on an elliptic curve group of prime order p then it suffices to set $p \approx 2^{160}$.

Why?

$$e^{1.92(\ln 2^{1024})^{1/3}(\ln \ln 2^{1024})^{2/3}} \approx \sqrt{2^{160}} = 2^{80}$$

Discerete Logarithm Problem

Why are Smaller Groups Preferable?

Group Size	Cost of Exponentiation
2^{160}	1
2^{1024}	260

Exponentiation takes time cubic in $\log|G|$ where G is the group.

Encryption and decryption will be 260 times faster in the smaller group!

Diffie-Hellman Problems

Diffie-Hellman Problems

- Let G be a cyclic group with generator g.
- Let $DH_g(h_1, h_2) = g^{\log_g h_1 \cdot \log_g h_2}$, that is if $h_1 = g^x$ and $h_2 = g^y$, then $DH_g(h_1, h_2) = g^{x \cdot y}$.
- Computational Diffie-Hellman(CDH): Compute $DH_g(h_1, h_2)$ for randomly chosen h_1, h_2.
- Claim 1: The CDH problem is not harder than Discrete Log problem.

Diffie-Hellman Problems

- Let G be a cyclic group with generator g.
- Let $DH_g(h_1, h_2) = g^{\log_g h_1 \cdot \log_g h_2}$, that is if $h_1 = g^x$ and $h_2 = g^y$, then $DH_g(h_1, h_2) = g^{x \cdot y}$.
- Computational Diffie-Hellman(CDH): Compute $DH_g(h_1, h_2)$ for randomly chosen h_1, h_2.
- Claim 1: The CDH problem is not harder than Discrete Log problem.
- Decisional Diffie-Hellman(DDH): Distinguish $DH_g(h_1, h_2)$ from randomly chosen $h' \in G$.
  - We say that the DDH problem is hard relative to Gen if for all PPT algorithms A, there exists a negligible function $negl$ such that
    $|\Pr[A(G, q, g, g^x, g^y, g^z) = 1] - \Pr[A(G, q, g, g^x, g^y, g^{xy}) = 1]| \leq negl(n)$.

Diffie-Hellman Problems

- DL → CDH → DDH
- Is there a Gen w.r.t. which DDH problem is hard?
  - Yes. We will see this shortly.

Status

Problem	Group	
	$\mathbf{Z}_p^*$	EC
DL	hard	harder
CDH	hard	harder
DDH	easy	harder

hard: best known algorithm takes time $e^{1.92(\ln p)^{1/3}(\ln \ln p)^{2/3}}$

harder: best known algorithm takes time $\sqrt{p}$, where p is the prime order of the group.

easy: There is a polynomial time algorithm.

Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange

Run Gen to obtain (G, q, g)
Pick $x \in Z_q$ randomly and let
$h_1 = g^x$.

G, q, g, h_1

h_2

Pick $y \in Z_q$ randomly and let $h_2 = g^y$.

$K_A = h_2^x$

$K_B = h_1^y$

$$K = K_A = K_B = g^{xy}$$

Note that the adversary sees g^x and g^y but does not know g^{xy}

Diffie-Hellman Key Exchange

Run Gen to obtain (G, q, g)
Pick $x \in Z_q$ randomly and let
$h_1 = g^x$.

G, q, g, h_1

h_2

Pick y $\in Z_q$ randomly and let $h_2 = g^y$.

$K_A = h_2^x$

$K_B = h_1^y$

$$K = K_A = K_B = g^{xy}$$

- Note that the adversary sees g^x and g^y but does not know g^{xy}.
- When do we call a key exchange protocol secure?

Diffie-Hellman Key Exchange

Run Gen to obtain (G, q, g)
Pick $x \in Z_q$ randomly and let
$h_1 = g^x$.

G, q, g, h_1

h_2

Pick $y \in Z_q$ randomly and let $h_2 = g^y$.

$K_A = h_2^x$

$K_B = h_1^y$

$$K = K_A = K_B = g^{xy}$$

- Note that the adversary sees g^x and g^y but does not know g^{xy}.
- When do we call a key exchange protocol secure?
  - When an adversary knows *nothing* about K.

Security: Key Exchange Protocols

- Experiment $KE_{A,\Pi}(n)$
  - Two parties execute the key exchange protocol Π. This execution of the protocol results in a transcript $trans$ containing all the messages sent by the parties, and a key k that is output by each of the parties.
  - A random bit $b \in \{0,1\}$ is chosen. If $b = 0$, then choose $k' \leftarrow \{0,1\}^n$ uniformly at random, and if $b = 1$, set $k' = k$.
  - Adversary A is given $trans$ and k' and let it output bit b'.
  - If $(b = b')$, then output 1 else output 0.
- Definition(Security): A key-exchange protocol Π is secure in the presence of an eavesdropper if for every PPT adversary A there exists a negligible function $negl$ such that

$$\Pr[KE_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n).$$

Security: Key Exchange Protocols

- Experiment $KE_{A,\Pi}(n)$
  - Two parties execute the key exchange protocol Π. This execution of the protocol results in a transcript $trans$ containing all the messages sent by the parties, and a key k that is output by each of the parties.
  - A random bit $b \in \{0,1\}$ is chosen. If $b = 0$, then choose $k' \leftarrow \{0,1\}^n$ uniformly at random, and if $b = 1$, set $k' = k$.
  - Adversary A is given $trans$ and k' and let it output bit b'.
  - If $(b = b')$, then output 1 else output 0.
- Definition(Security): A key-exchange protocol Π is secure in the presence of an eavesdropper if for every PPT adversary A there exists a negligible function $negl$ such that

$$\Pr[KE_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n).$$

- Theorem(informal): If the DDH problem is hard w.r.t. Gen, then the Diffie-Hellman key exchange protocol is secure as per the above notion of security.

Diffie-Hellman Key Exchange

Run Gen to obtain (G, q, g)
Pick $x \in Z_q$ randomly and let $h_1 = g^x$.

G, q, g, h_1

h_2

Pick $y \in Z_q$ randomly and let $h_2 = g^y$.

$K_A = h_2^x$

$K_B = h_1^y$

$$K = K_A = K_B = g^{xy}$$

- So can we safely deploy Diffie-Hellman key exchange protocol?

Diffie-Hellman Key Exchange

Run Gen to obtain (G, q, g)
Pick $x \in Z_q$ randomly and let $h_1 = g^x$.

G, q, g, h_1

h_2

Pick $y \in Z_q$ randomly and let $h_2 = g^y$.

$K_A = h_2^x$

$K_B = h_1^y$

$$K = K_A = K_B = g^{xy}$$

- So can we safely deploy Diffie-Hellman key exchange protocol?
- No there is a simple attack on the protocol called *man-in-the-middle-attack*.

Diffie-Hellman Key Exchange

Run Gen to obtain (G, q, g)
Pick $x \in Z_q$ randomly and let $h_1 = g^x$.

G, q, g, h_1

h_2

Pick $y \in Z_q$ randomly and let $h_2 = g^y$.

$K_A = h_2^x$

$K_B = h_1^y$

$$K = K_A = K_B = g^{xy}$$

- So can we safely deploy Diffie-Hellman key exchange protocol?
- No there is a simple attack on the protocol called *man-in-the-middle-attack*.
- So is our security notion weak?
  - No it is a strong notion but for *passive* adversaries only.

Diffie-Hellman Key Exchange: MITM attack

Run Gen to obtain (G, q, g)

G, q, g, g^x

$g^{y'}$

$G, q, g, g^{x'}$

g^y

Alice uses $g^{xy'}$

Malcolm uses
$g^{xy'}$ with Alice
$g^{x'y}$ with Bob

Bob uses $g^{x'y}$

Diffie-Hellman Key Exchange: MITM attack

Run Gen to obtain (G, q, g)

G, q, g, g^x

$g^{y'}$

$G, q, g, g^{x'}$

g^y

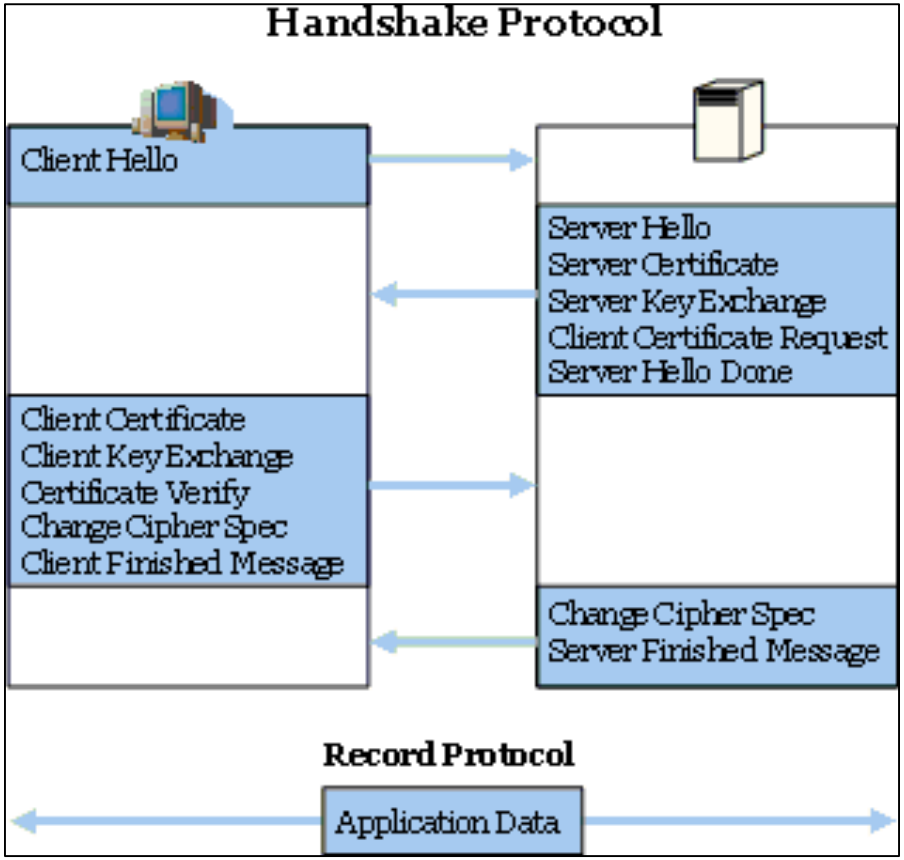
Alice uses $g^{xy'}$

Malcolm uses
$g^{xy'}$ with Alice
$g^{x'y}$ with Bob

Bob uses $g^{x'y}$

- So, do we actually use Diffile-Hallman key exchange protocol in practice.
  - Yes but in a more sophisticated form.

Case study: TLS handshake protocol



- Server Certificate: $[pk,\ Sign_{sk_G}(pk)]$
  - pk is the public key of the sever.
  - sk_G is the secret key of the trusted authority.

End

Slides 26-29 and 33 use Mihir Bellare's slides.