Modern Cryptography

600.442

Lecture #15

Dr. Christopher Pappacena

Fall 2013

Last Time

- Diffie-Hellman Key Exchange

- Discrete Logarithm, CDH, and DDH Problems

- We gave a group generation $\mathcal{G}$ algorithm for which the DDH problem is believed to be hard:

RSA Encryption

The first, and most widely used, public-key encryption scheme is RSA encryption, named after Rivest, Shamir, and Adelman.

We'll talk about RSA before giving a formal definition of public-key encryption.

The construction builds on the number theory we used for Diffie-Hellman.

More Number Theory

RSA encryption uses the group $\mathbb{Z}_N^*$ where $N = pq$ is the product of two primes.

The first question we can ask about $\mathbb{Z}_N^*$, for *any* integer N, is: What is its order?

Since $a \in \mathbb{Z}_N^*$ if and only if $(a, N) = 1$, we know that $|\mathbb{Z}_N^*|$ is equal to the number of integers a in $\{1, \ldots, N-1\}$ with $(a, N) = 1$.

Euler's Phi Function

The integer $|\mathbb{Z}_N^*|$ is denoted by $\phi(N)$ and ϕ is called *Euler's phi function*. For example, if p is prime then $\phi(p) = p - 1$.

Proposition: If $N = pq$ then $\phi(N) = (p-1)(q-1)$.

Proof: An integer is relatively prime to N provided it is not a multiple of p or a multiple of q. There are $q-1$ nonzero multiples of p in $\mathbb{Z}_N$: $p, 2p, \ldots, (q-1)p$. Similarly, there are $p-1$ nonzero multiples of q, namely $q, 2q, \ldots, (p-1)q$. So

$$\phi(N) = (N-1) - (p-1) - (q-1) = pq - (p+q) + 1 = (p-1)(q-1).$$

A General Formula

Theorem: If $N = p_1^{r_1} \times \cdots \times p_n^{r_n}$ is the prime factorization of N, then

$$\phi(N) = \prod_{i=1}^{n} p_i^{r_i - 1}(p_i - 1).$$

An important thing to note is that computing $\phi(N)$ is easy, *provided we know the factorization of* N.

In general, there is no known way to compute $\phi(N)$ which does *not* require knowing the factorization of N.

This fact is critical for RSA encryption as well as several generalizations, including Paillier encryption.

How Do We Find Primes?

The Diffie-Hellman and RSA generation schemes require us to find n-bit primes for some security parameter n. How do we do this?

Prime Number Theorem: Let $\pi(x)$ equal the number of primes $\leq x$. Then $\pi(x) = O(x/\ln x)$.

In fact, much sharper statements about the distribution of primes are known.

For us, we simply note that if you write down an n-bit integer *at random*, it will be prime with probability about $1.44/n$.

So the expected run time for the "random guess" algorithm is $O(n)$ primality tests.

Primality Testing

Given an integer N, how can we tell if it is prime?

Here are a couple of “easy out” tests:

- If N is divisible by a small prime, then N is composite. So simple *trial division* can quickly rule out most numbers.

- If N is prime then we know that $a^{N-1} = 1 \pmod{N}$. So we can choose a random a and check if this identity holds. If not, then N is composite. The integer a is called a *witness* for N.

There are infinitely many composite integers N for which $a^{N-1} = 1 \pmod{N}$ (*Carmichael numbers*). These numbers do not have *any* witnesses.

Strong Witnesses

Write $N - 1 = 2^r u$ for some $r \geq 1$ and odd u. Given $a \in \mathbb{Z}_N^*$, consider the sequence

$$(a^u, a^{2u}, \ldots, a^{2^r u}).$$

Three possibilities for this sequence modulo N:

- $(\pm 1, 1, \ldots, 1)$.
- $(*, \ldots, *, -1, 1, \ldots, 1)$
- $a^u \neq \pm 1 \pmod{N}$ and $a^{2^i u} \neq -1 \pmod{N}$ for $1 \leq i \leq r-1$. In this case a is called a *strong witness* for N.

Strong Witnesses

As the name suggests, if a is a witness then a is a strong witness.

Also, if N is prime then N does not have any strong witnesses: one of the first two cases *must* happen.

Theorem: If N is an odd composite which is not a power of a prime, then at least half of the elements in $\mathbb{Z}_N^*$ are strong witnesses for N.

If we choose t elements of $\mathbb{Z}_N^*$ at random and none of them are strong witnesses for N, then the probability that N is composite is at most 2^{-t}.

The Miller-Rabin Primality Test

Given input N and t:

- If N is even or a perfect power, output "composite".
- Write $N - 1 = 2^r u$ with u odd and $r \geq 1$.
- For $j = 1, \ldots, t$, do:
  - Choose $a \leftarrow \{1, \ldots, N-1\}$. If $(a, N) \neq 1$, return "composite".
  - If a is a strong witness, return "composite".
- Return "prime".

Theorem: The Miller-Rabin test runs in time polynomial in $n = \|N\|$ and t. If N is prime, it returns "prime" with probability 1. If N is composite, it returns "prime" with probability at most 2^{-t}.

To generate a random n-bit prime, we select a random n-bit integer and test it for primality. This algorithm runs in probabilistic polynomial time since we expect to try $O(n)$ integers and each trial is polynomial in n.

In fact, there is a *deterministic* polynomial-time primality testing algorithm (Agrawal, Kayal, and Saxena, 2004).

In practice, the Miller-Rabin test runs faster and produces primes with a high degree of confidence.

Alternative Approach

The advantage to choosing n-bit integers uniformly and testing for primality is that it samples *uniformly* from n-bit primes.

It is easier in practice to select an n-bit integer k and find the *next prime*. We can use the Sieve of Eratosthenes to eliminate nearby composites and reduce the number of Miller-Rabin tests.

This method runs faster and requires fewer random bits.

We write $p = \text{NextPrime}(k)$ to denote the first prime which is $\geq k$.

Generating RSA Moduli

Define a PPT algorithm GenMod(n) as follows:

- On input n, GenMod(n) finds two random n-bit probable primes p and q using the Miller-Rabin test.

- The parameter t in the Miller-Rabin test is chosen so that 2^{-t} is a negligible function of n.

- GenMod(n) returns (p, q, N).

We can also define GenMod by calling the NextPrime function if we prefer.

The Factoring Experiment

- GenMod(n) is run to produce (p, q, N).

- The adversary $\mathcal{A}$ is given N and returns p', q'.

- $\mathcal{A}$ *succeeds* if $N = p'q'$ and *fails* otherwise. We write $\text{Fac}_{\mathcal{A}}(n) = 1$ if $\mathcal{A}$ succeds.

We say that *factoring is hard* relative to GenMod if $\Pr[\text{Fac}_{\mathcal{A}}(n) = 1]$ is negligible for all PPT adversaries $\mathcal{A}$.

Factoring Is Hard (We Think)

- It is widely believed that factoring is hard relative to GenMod(n).

- As with the hardness of DDH for $\mathcal{G}(n)$, we do not have a *proof* that factoring is hard.

- Unconditional proofs of either of these facts would imply $P \neq NP$.

- These problems (especially factoring) have been studied for a long time, which adds to our confidence that they are hard.

How Hard Is Factoring?

How hard is it to factor $N = pq$, with p and q both $O(\sqrt{N})$?

Selected Methods

Method	Year	Complexity
Trial Division	$-\infty$	$O(\sqrt{N})$
Fermat's Method	1600s	$O(\sqrt[3]{N})$
Pollard Rho	1975	$O(\sqrt[4]{N})$
Continued Fractions	1931, 1975	$L_N(1/2, \sqrt{2})$
Quadratic Sieve	1981	$L_N(1/2, 1)$
Number Field Sieve	1991	$L_N(1/3, \sqrt[3]{64/9})$

RSA Encryption

- Run GenMod(n) to produce (p, q, N).

- Choose e (the *encryption exponent*) relatively prime to $\phi(N)$ and publish (N, e) as the public key.

- Find d (the *decryption exponent*) satisfying $ed = 1 \pmod{\phi(N)}$ using the Euclidean algorithm.

- The private key (also called the secret key) is (p, q, d).

RSA Encryption and Decryption

- To encrypt a message $m \in \mathbb{Z}_N^*$, set $c = m^e \pmod{N}$.

- To decrypt a ciphertext c, set $m = c^d \pmod{N}$.

- This works because

$$(m^e)^d = m^{ed} = m^{ed \bmod \phi(N)} = m^1 = m \pmod{N}.$$

- We can also use the *Chinese Remainder Theorem* to compute $c^d \pmod{p}$ and $c^d \pmod{q}$ and use them to reconstruct $c^d \pmod{N}$. This offers no cryptographic advantage but is cheaper to compute.

Security of RSA

- For RSA Encryption to be secure, it must be hard for an adversary $\mathcal{A}$ to determine m from $m^e \pmod{N}$.

- If $\mathcal{A}$ can determine d, then he can read the message.

- Knowing d allows $\mathcal{A}$ to compute $\phi(N)$ (board).

- Given $\phi(N)$, it is possible to factor N (HW problem).

- On the other hand, factoring N allows $\mathcal{A}$ to compute $\phi(N)$, and then d.

Security of RSA, II

- We conclude that factoring N is equivalent to determining d.

- But, is it possible to recover m from $m^e \pmod{N}$ *without* determining d? Nobody knows.

- For RSA to be secure, it must be difficult to recover m from $m^e \pmod{N}$, regardless of how it is done.

- This is called the *RSA Problem*.

The RSA Experiment

Let GenRSA be a PPT algorithm which, on input n, returns (N, e, d) where $N = pq$ is an RSA modulus.

- Run GenRSA(n) to obtain (N, e, d) and choose $y \leftarrow \mathbb{Z}_N^*$ uniformly.

- $\mathcal{A}$ is given N, e, y, and outputs $x \in \mathbb{Z}_N^*$.

- We say $\mathsf{RSAInv}_{\mathcal{A}}(n) = 1$ if $x^e = y \pmod{N}$ and say that $\mathcal{A}$ *succeeds*. Otherwise $\mathcal{A}$ *fails*.

Definition: We say that *the RSA problem is hard relative to* GenRSA if, for all PPT adversaries $\mathcal{A}$, we have

$$\Pr[\mathsf{RSAInv}_{\mathcal{A}}(n) = 1] = \mathtt{negl}(n)$$

for some negligibe function $\mathtt{negl}$.

As mentioned above, it is widely believed that if GenRSA(n) is instantiated by calling GenMod(n), then the RSA problem is hard for (almost) any choice of e with $(e, \phi(N)) = 1$.

A Cautionary Example

- Generate an RSA modulus N with encryption exponent $e = 3$.
- Let m be a short message. Here "short" means $m < N^{1/3}$.
- Then $c = m^3$; that is, no modular reduction takes place!
- An adversary $\mathcal{A}$ can easily find m by taking a real cube root.

This does not violate the assertion that the RSA problem is hard, even for $e = 3$. (Why not?) It does show that "textbook RSA" can have unintended weaknesses.