Cryptology 05.11.2015

Lecture 13: RSA II; ElGamal; Knapsack

Instructor: Goutam Paul *Scribe: Laltu Sardar*

1 Primality Testing

Primality testing simply is "Given an integer n decide if n is prime or composite". Due to the great invention of public key cryptography at the end of the 20th century, the interest in primality testing has grown rapidly in the past three decades.

The security of this type of cryptographic schemes primarily relies on the difficulty involved in factoring the product of very large primes.

Integer factorization poses many problems, a key one being the testing of numbers for primality. A reliable and fast test for primality would help us in constructing efficient and secure cryptosystem.

Therefore, the mathematics and computer science communities have begun to address the problem of primality testing.

1.1 Deterministic Test

One of the simplest deterministic primality test is checking from $i = 1$ to $n-1$ if $i|n$. Which can be reduced to check from $i = 1$ to $\sqrt{n}$ if $i|n$. But both are *exponential time* algorithm with respect to number of bits.

The AKS Algorithm is the first deterministic *polynomial-time* primality test named after its authors M. Agrawal, N. Kayal, and N. Saxena. In August 2002, this algorithm was presented in the paper "PRIMES is in P", Agrawal et al. (2002).

The AKS primality test is based upon the following theorem:

Theorem 1.1 *An integer $n(\geq 2)$ is prime if and only if the polynomial congruence relation*

$$(x + a)^n \equiv (x^n + a) \bmod n$$

holds for some a coprime to n. Note that x is a free variable in natural numbers.

The theorem is an extension of Fermat's Little Theorem.

1.2 Probabilistic Primality Testing

Probabilistic tests provide provable bounds on the probability of being fooled by a composite number. Many popular primality tests are probabilistic tests.

These tests use, apart from the tested number n, some other numbers a which are chosen at random from some sample space; the usual randomized primality tests never report a prime number as composite, but it is possible for a composite number to be reported as prime.

The probability of error can be reduced by repeating the test with several independently chosen values of a; for two commonly used tests, for any composite n at least half the a's detect n's compositeness, so k repetitions reduce the error probability to at most 2^{-k}, which can be made arbitrarily small by increasing k.

The basic structure of randomized primality tests is as follows:

1. Randomly pick a number a.
2. Check some equality (corresponding to the chosen test) involving a and the given number n. If the equality fails to hold true, then n is a composite number, a is known as a witness for the compositeness, and the test stops.
3. Repeat from step 1 until the required accuracy is achieved.

After one or more iterations, if n is not found to be a composite number, then it can be declared "probably prime".

1.2.1 Fermat's Test

One of the most simplest probabilistic primality test is the "Fermat primality test". This test is based on the following theorem:

Theorem 1.2 *If p is a prime number then for any integer a, not divisible by p*

$$a^{p-1} \equiv 1 \bmod p$$

But If $a^{n-1} \equiv 1 \bmod n$ this does not imply n is a prime. For example, if $n = 341$ and $a = 2$ then $2^{341-1} \equiv 1 \bmod 341$, but 341=11.31 is composite. The algorithm of that test is as follows:

Algorithm 1 Fermat's Primlaity Test

Input: An odd integer N
Output: 'Yes' or 'No'
Steps:

Choose a random integer a such that $2 \leq a \leq N-2$.
if $gcd(a, N) \neq 1$ **then**
    return (No) /*N is composite*/
$s \leftarrow a^{N-1} \bmod n$
if $s \neq 1$ **then**
    return (No) /*N is composite*/
else
    return (Yes)/*N is prime*/

See that here when the algorithm tells composite it is composite but when says prime it may prime with certain probability. To Get the answer with "good" probability we can repeat the algorithm depending on how much "good" we want.

1.2.2 Miller-Rabin Test

Miller-Rabin primality Testing based on finding non-trivial square root of 1 modulo n. Since we basically work with large primes we can assume primes to be odd.

If p is a prime and $p > 2$ then 1 and -1 are the only square roots of 1 modulo p, but the converse is not true in general. So if we found any non trivial square root of 1 modulo n, we can say n must be composite.

let us assume that $n = 2^k.\alpha + 1$, where $gcd(2, \alpha) = 1$ and $k \geq 1$. Thus

$$a^{n-1} \equiv (a^{\alpha} \bmod n)^{2^k} \bmod n$$

Let $b_0 = a^{\alpha} \bmod n$, then

$$b_i = b_{i-1}^2 \bmod n, i = 1, 2, \ldots, k.$$

This implies that $a^{n-1} \bmod n$ may be calculated in $k + 1$ intermediate steps.

Basically in this algorithm we find either $a^{\alpha} = 1 \mod n$ or if $\exists i \in 0, 1, \ldots, k-1$ such that $a^{2^i \alpha} = -1 \bmod n$. The algorithm is as follows:

Algorithm 2 Miller-Rabin Primlaity Test

Input: An odd integer $n \geq 3$
Output: 'Yes' or 'No'
Steps:

1: write $n - 1 = 2^h t$, where $h \geq 1$ and $gcd(2, t) = 1$
2: choose a random integer a such that $2 \leq a \leq n - 2$.
3: $b \leftarrow a^t \bmod n$
4: **if** $b = 1 \bmod n$ **then**
5: **return** (Yes) /* The number is a prime integer */
6: **for** i=1 **to** h **do**
7: **if** $b = -1 \bmod n$ **then**
8: **return** (Yes) /* The number is a prime integer */
9: **else**
10: $b \leftarrow b^2 \bmod n$
11: $i \leftarrow i + 1$
12: **return** (No) /* The number is a composite integer */

The test is very fast and requires no more than $O(\log_2 n)$ multiplications $(\bmod n)$. Unfortunately, a number which passes the test is not necessarily prime.

Monier (1980) and Rabin (1980) have shown that a composite number passes the test for at most 1/4 of the possible bases a. If N multiple independent tests are performed on a composite number, then the probability that it passes each test is $\frac{1}{4^N}$ or less.

2 ElGamal Cryptosystem

Taher Elgamal first described the ElGamal Cryptosystem in an article published in the proceedings of the CRYPTO '84, a conference on the advances of cryptology.

The original public key system proposed by Diffie and Hellman requires interaction of both parties to calculate a common private key. This poses problems if the cryptosystem should be applied to communication systems where both parties are not able to interact in reasonable time due to delays in transmission or unavailability of the receiving party

Thus ElGamal simplified the Diffie-Hellman key exchange algorithm by introducing a random exponent a. This exponent is an replacement for the private exponent of the receiving entity. Due to this simplification the algorithm can be used to encrypt in one direction, without the necessity of the second party to take actively part. The key advance here is that the algorithm can be used for encryption of electronic messages, which are transmitted by the means of public store-and-forward services.

Before going into the algorithms let us see the following definition,

Definition 2.1 *An element a of a finite field $GF(q)$ is said to be a* ***primitive element*** *of $GF(q)$ if $a^{q-1} = 1 \bmod q$ and $a^i \neq 1 \bmod q$, $\forall i = 2, 3, \ldots, q-2$*

ElGamal encryption consists of three components: the key generation, the encryption algorithm and the decryption algorithm. Algorithms are described bellow.

2.1 Key Generation

The basic requirement for a cryptographic system is at least one key for symmetric algorithms and two keys for asymmetric algorithms.

With ElGamal, only the receiver needs to create a key in advance and publish it. Following our naming scheme from above, we will now follow Bob through his procedure of key generation.

Bob will use the following algorithm to generate his keypair:

Algorithm 3 ElGamal Key Generation

1: Generate a large prime p, consider Z_p^*
2: Find a primitive element α of Z_p^*
3: Chooses an a randomly from $\{1, \ldots, p-1\}$
4: Compute $\beta \leftarrow \alpha^a$
5: Publish $\{ p, \alpha, \beta\}$ as public key and
6: Keep a as private key

The public key now needs to be published using some dedicated keyserver or other means, so that Alice is able to get hold of it.

2.2 Encryption

To encrypt a message M to Bob, Alice first needs to obtain his public key triplet (p, α, β) from a key server or by receiving it from him via unencrypted electronic mail. There is no security issue involved in this transmission, as the only secret part, a, is sent in β.

Since the core assumption of the ElGamal cryptosystem says that it is infeasible to compute the discrete logarithm, this is safe.

For the encryption of the plaintext message x, Alice has to apply the following algorithm:

Algorithm 4 ElGamal Encryption

Input: a message x, p ,α , β
Output: cipher text (y_1, y_2)
Steps:

1: choose a random k from $\{1, \ldots, q-1\}$,
2: $y_1 \leftarrow \alpha^k \bmod p$
3: $y_2 \leftarrow x\beta^k \bmod p$
4: **return** (y_1, y_2)

Even if an attacker would listen to this transmission, and collect β of bob from a keyserver, he would still not be able to derive α^{ak}.

2.3 Decryption

After receiving the encrypted message (y_1, y_2) and the randomized public key β, Bob has to use the dncryption algorithm to be able to read the plaintext x. This algorithm is as follows:

Algorithm 5 ElGamal Decryption

Input: a ciphertext (y_1, y_2) and a private key a
output: The message x
Steps:

1: $x \leftarrow y_2.(y_1^{\alpha})^{-1} \bmod (p)$
2: **return** x

2.4 Proof of correctness:

It can be easily seen that

$$\begin{aligned} Dec_{\alpha}(y_1, y_2) &= y_2.(y_1^{\alpha})^{-1} \bmod (p) \\ &= x\beta^k.(\alpha^k)^{-a} \bmod (p) \\ &= x.\alpha^{ak}.\alpha^{-ak} \bmod (p) \\ &= x \bmod (p) \\ &= x \end{aligned}$$

So Bob can get back the message x from the encrypted ciphertext. Now both of them use x as common symmetric key.

3 Knapsack Cryptosystem

knapsack cryptosystem was one of the earliest public key cryptosystems invented by Ralph Merkle and Martin Hellman in 1978. Although this system, and several variants of it, were broken in the early 1980's, it is still worth studying for several reasons, not the least of which is the elegance of its underlying mathematics.

In the Knapsack Cryptosystem based on subset sum problem which is an NP-complete problem. But if the sequence of the set elements are in super increasing manner then it can be find back the original sequence easily. This is the case with the subset sum problem, and this permits the creation of a trapdoor.

Definition 3.1 *A sequence of integers* $(w_1, w_2 \ldots, w_n)$ *is said to form a* ***super increasing knapsack*** *if*

$$w_i < \sum_{j=1}^{i-1} w_j, \forall i = 2, 3, \ldots, n$$

Key Generation, Encryption and Decryption algorithms of this cryptosystem are described bellow.

3.1 Key Generation

In Knapsack cryptosystem the keys are two knapsacks. The public key is a 'hard' knapsack $\mathbf{w}'$, and the private key is an 'easy', or superincreasing, knapsack $\mathbf{w}$, combined with two additional numbers, a multiplier N and a modulus W. The multiplier and modulus can be used to convert the superincreasing knapsack into the hard knapsack.

These same numbers are used to transform the sum of the subset of the hard knapsack into the sum of the subset of the easy knapsack, which is a problem that is solvable in polynomial time.

Algorithm 6 Knapsack Key Generation

1: Choose a super-increasing knapsack $\mathbf{w} = (w_1, w_2 \ldots, w_n)$.
2: Choose a no W such that $W > \sum_{i=1}^{n} w_i$.
3: Select a no N such that $gcd(N, W) = 1$.
4: Compute $w'_i = w_i * N \bmod W$, $\forall i = 2, 3, \ldots, n$
5: Keep W, N and the sequence $\mathbf{w}$ as private key.
6: Publish the sequence $\mathbf{w}' = (w'_1, w'_2 \ldots, w'_n)$ as public key

3.2 Encryption

To encrypt a message, a subset of the hard knapsack $\mathbf{w}'$ is chosen by comparing it with a set of bits (the plaintext) equal in length to the key. Each term in the public key that corresponds to a 1 in the plaintext is an element of the subset w'_i, while terms that corresponding to 0 in the plaintext are ignored. The elements of this subset are added together and the resulting sum is the ciphertext.

Algorithm 7 Knapsack Encryption

Input: a message block $m = (b_1 b_2 \dots b_n)_2$ of length n, $\mathbf{w}'$
Output: ciphertext C
Steps:

1: $C \leftarrow 0$
2: **for** $i = 1$ to n **do**
3: $\quad C \leftarrow C + b_i w'_i$
4: **return** C

3.3 Decryption

Decryption is possible because the multiplier and modulus used to transform the easy knapsack into the public key can also be used to transform the number representing the ciphertext into the sum of the corresponding elements of the superincreasing knapsack. Then, using a simple greedy algorithm, the easy knapsack can be solved using $O(n)$ arithmetic operations, which decrypts the message.

Algorithm 8 Knapsack Decryption

Input: Ciphertext C, N, W and $\mathbf{w}$
Output: $m = (b_1 b_2 \dots b_n)_2$
Steps:

1: $C' \leftarrow CN^{-1} \bmod W$
2: **for** $i = n$ to 1 **do**
3: $\quad$ **if** $C' > w_i$ **then**
4: $\quad\quad$ take $b_i \leftarrow 1$
5: $\quad\quad C' \leftarrow C' - w_i$
6: $\quad$ **else**
7: $\quad\quad$ take $b_i \leftarrow 0$
8: $\quad i \leftarrow i - 1$
9: **return** $(b_1 b_2 \dots b_n)_2$

To prove the correctness of the algorithm It is enough to see that

$$Dec_{N,W}(C) = CN^{-1} \bmod W = \sum_{i=1}^{n} b_i w'_i N^{-1} \bmod W = \sum_{i=1}^{n} b_i w_i \bmod W = \sum_{i=1}^{n} b_i w_i$$

This system was very popular for a while since it is very fast to implement. However, in the early 1980's, Shamir, using Lenstra's fast linear programming algorithm, was able to peel away this disguise and obtain Bob's superincreasing set.

Many other methods for disguising the super-increasing set have been tried, but most of these have also been cracked. One method, known as the Chor-Rivest cryptosystem, uses finite field manipulations and this is still considered to be secure.