Alin Tomescu, CSE408
Tuesday, March 15th, Lecture #13

Primality, RSA and El Gamal encryption

Previously, in CSE508…

$$\varphi(p^k) = p^k - p^{k-1} = (p-1)p^{k-1}$$

$$\gcd(a,b) = 1 \Rightarrow \varphi(a,b) = \varphi(a)\varphi(b)$$

$$n = p_1^{k_1} p_2^{k_2} \dots p_n^{k_n} \Rightarrow \varphi(n) = \varphi(p_1^{k_1})\varphi(p_2^{k_2}) \dots \varphi(p_n^{k_n})$$

$$n = pq, \text{ where } p \text{ and } q \text{ are primes} \Rightarrow \varphi(n) = (p-1)(q-1)$$

Logarithmic exponentiation

We showed that to compute $a^k \bmod n$, we can reduce $a \bmod n$ and reduce $k \bmod \varphi(n)$, obtaining the following equality:

$$a^k \bmod n = (a \bmod n)^{k \bmod \varphi(n)} \bmod n$$

However, this is still problematic since $k \bmod \varphi(n)$ might be very large requiring us to do a lot of multiplications. **Logarithmic exponentiation** is here to solve that problem.

Let's see how we can compute $a^b \bmod n$, by doing around $\log_2 b$ multiplications.

Let $b = 2^k b_k + 2^{k-1} b_{k-1} + \dots + b_0$, where $b_i =$ i^th bit of b

Then, $a^b = a^{2^k b_k + 2^{k-1} b_{k-1} + \dots + b_0} = a^{2^k b_k} a^{2^{k-1} b_{k-1}} \dots a^{b_0} = \left(a^{2^k}\right)^{b_k} \left(a^{2^{k-1}}\right)^{b_{k-1}} \dots \left(a^{2^0}\right)^{b_0}$

We can compute all the values $a^1, a^2, a^4, a^8, \dots a^{2^k}$ by doing only $k = \log_2 b$ multiplications as follows:

The trick is to start with a^1 and then square repeatedly, obtaining each power along the way.

```
// Let a[i] denote the value of a^(2^i)

int i = 0;
a[i++] = 1;
while(i <= k)
{
       a[i] = a[i-1] * a[i-1]
       i++;
}
```

Note: The values $a^1, a^2, a^4, a^8, \dots a^{2^k}$ can be computed mod n, since it won't affect the final result of $a^b \bmod n$.

Now, that we have $a^1, a^2, a^4, a^8, \dots a^{2^k}$, since $b_i =$ either 0 or 1, then some of the $\left(a^{2^i}\right)^{b_i}$ powers will be equal to $\left(a^{2^i}\right)^0 = 1$ and others will be equal to $\left(a^{2^i}\right)^1 = a^{2^i}$. We can now compute $a^b \bmod n$ by multiplying the a^{2^i} powers for which $b_i = 1$.

This algorithm runs in $O(\log_2 b)$. The exact running time will be $O(\log_2 b)$ multiplications of $O(\log_2 n)$-bit integers.

The pseudocode for $MODEXP(a, b, n)$ can be found below:

```
MODEXP(a, b, n):

r = 1                        // the result of a^b mod n
k = length_in_bits(b)        // the length in bits of b
m = a                        // the current value of a^(2^i)

for i = 0 to k-1
       if b_i = 1
              r = (r * m) mod n
       m = m^2 mod n

return r
```

How do you pick a huge prime?

The prime number theorem

Theorem: The number of primes $\leq n \approx \frac{n}{\ln n}$

Easy prime picking algorithm:

1. Pick a random number
2. Test it for primality
3. Repeat until the test succeeds
    a. The theorem above essentially tells us that **the odds of picking a prime will not be so bad**

Miller-Rabin primality test

For many years, poly-time algorithms for primality testing weren't known until 2002 when the AKS (Agrawal-Kayal-Saxena) primality test was developed.

The **Miller-Rabin primality test** is a primality-testing probabilistic algorithm.

- randomized algorithm
- outputs either: *definitely composite* or *probably prime*
- the probability of error can be made as small as desired

How does it work?

Theorem: There are only two roots of unity in $\mathbb{Z}_p$, where p is prime: the trivial roots 1 and -1.

Proof: p is prime.

Let $a \in \mathbb{Z}_p$ such that that $a^2 \equiv 1 \pmod{p} \Rightarrow p \mid a^2 - 1 \Rightarrow p \mid (a-1)(a+1) \Rightarrow p \mid (a-1)$ or $p \mid (a+1)$

So $a \equiv 1 \pmod{p}$ or $a \equiv -1 \pmod{p}$

So the there are only two square roots of 1 in $\mathbb{Z}_p$: 1 and -1. QED.

Main point to get across: There are only two roots of unity in $\mathbb{Z}_p$, where p is prime: the trivial roots 1 and -1.

Important consequence: This means that if you can find a number $a \neq \pm 1$ in $\mathbb{Z}_p$ such that $a^2 = 1 \pmod p$ then p is not prime.

Let's think what happens mod a composite number $n = ab$ (in $\mathbb{Z}_{ab}^*$).

Suppose we have a composite number $n = ab$, such that $\gcd(a, b) = 1$, then $\mathbb{Z}_n^* \cong \mathbb{Z}_a^* \times \mathbb{Z}_b^*$.

What are the square roots of $(1,1)$ in $\mathbb{Z}_a^* \times \mathbb{Z}_b^*$? $(1,1), (-1,-1), (1,-1), (-1,1)$.

Therefore, by the isomorphism of the two groups, there are 4 square roots of 1 in $\mathbb{Z}_n^*$.

The **Miller-Rabin test**, assume n is odd and not a perfect power of a prime (since this can be easily tested for).

We let $n - 1 = 2^s \times d$, where d will be odd (so we subtract one from n and dived the result by 2, until we get an odd number d).

```
MillerRabin(n, k) {
        if [n is even]
                return COMPOSITE
        if [n is a prime power n = p^k]
                return COMPOSITE

        repeat k times
                pick random a ∈ Z_n (exclude 1, -1 and 0)
                if [gcd(a,n) ≠ 1]
                        return COMPOSITE

                x = a^d (mod n)
                if [x = 1 (mod n) or x = -1 = n - 1 (mod n)]
                        then do next LOOP

                for r = 1, r ≤ s - 1
                        x = x^2 (mod n)
                        if [x = 1 (mod n)]
                                then return COMPOSITE
                        if [x = -1 = n - 1 (mod n)]
                                then do next LOOP
                return COMPOSITE
        return PRIME
}
```

Description:

- write $n - 1$ in the form $n - 1 = 2^s d$
- choose a random base a and check the value of $a^{n-1} \pmod n$, but…
- perform this computation by first determining $a^d \pmod n$, and then repeatedly squaring to get the sequence:
    - $a^d, a^{2d}, a^{2^2 d}, \ldots, a^{2^{s-1} d}, a^{2^s d} = a^{n-1} \pmod n$
        - every power in this sequence here is computed $\bmod n$ of course
- If $a^{n-1} \neq 1 \pmod n \Leftrightarrow \gcd(a, n) \neq 1$, then n is composite (by the contrapositive of Fermat's little theorem), and we're done
- But if $a^{n-1} = 1 \pmod n$, we conduct a little follow-up test:

- if $a^{n-1} = 1 \pmod{n}$ then somewhere in the preceding sequence, we must have ran into a 1 for the first time.
  - If this happened after the first position (that is, if $a^s \neq 1 \pmod{n}$), and if the preceding value in the list is not $-1 = n-1 \pmod{n}$ then we declare n composite, because we have just found a non-trivial square root of 1 modulo n (a number that is not $\pm 1 \pmod{n}$ but that when squared is equal to $1 \pmod{n}$), and such a root can only exist when n is composite.
    - Note that if the preceding value is -1 we haven't found a non-trivial square root of 1, since we found -1 squaring to 1 which is a trivial square root of 1
  - If this happened on the first position then we haven't found a non-trivial square root of 1, since all the positions after it, being the squares of the first position will also be 1

Another way of thinking about this is by looking at the sequence $a^d, a^{2d}, a^{2^2 d}, \dots, a^{2^{s-1}d}, a^{2^s d} = a^{n-1} \pmod{n}$ and realizing that in the interesting case when $a^{n-1} = 1 \pmod{n}$ in which you can't really tell whether n is prime or not, you have computed a series of squares that eventually yielded a 1, so you definitely have either a trivial or a non-trivial square root of unity in that sequence.

The non-trivial roots will only arise when $a^d \neq \pm 1$. When that happens, if as you repeatedly square, you keep getting numbers different that -1 finally getting a 1 then you found a non-trivial square root of 1.

If you kept squaring and you got -1 then obviously after squaring -1 you will get 1, but that's of no use since it's a trivial square root.

Think about it a lot, it might take reading few articles and some textbook chapters to get it ☺

Definition: a is a Miller-Rabin liar for n if n is composite and $MR(a, n)$ outputs prime, instead of composite.

The a's that will tell you a composite number n is prime are called **liars**. The bound on the number of such a's if n is composite is:

$$\Pr[a = MR\ liar\ for\ n] = \frac{1}{4}$$

To overcome this, you repeat the test k times, so the $\Pr[accepting\ a\ composite\ as\ a\ prime] \leq 4^{-k}$

Note that if n is really prime, the algorithm will **always correctly say it's a prime**. However, when n is composite and you pick a's to test it against, you might be unlucky, pick all the a's as liars and the algorithm will tell you n is prime. The probability of that happening is $\leq 4^{-k}$ though.

Running time of $MR(a, n)$ is roughly doing one logarithmic exponentiation, so repeating k times will be $k \log n$.

RSA encryption

RSA is a public-key cryptosystem.

A real-world analogy: Alice wants to receive messages from Bob, and in the real world she buys a lock, gives it to Bob, Bob puts his message in it and sends the lock to Alice. Alice can unlock it with her key and read the message.

Alice:

- pick large primes p and q, compute $n = pq$
- pick e such that $\gcd(e, \varphi(n)) = 1$

- compute d such that $ed = 1 \left(\text{mod } \varphi(n)\right) \Leftrightarrow d = e^{-1} \left(\text{mod } \varphi(n)\right)$
  - use the Extended Euclidian Algorithm to compute d
  - run $EEA\left(e, \varphi(n)\right)$, getting $ex + \varphi(n)y = 1 \Rightarrow \varphi(n) \mid ex - 1 \Rightarrow ex = 1 \text{ mod } \varphi(n)$
    - $d = x$
- send (n, e) to Bob as the public key
- keep (n, d) as the private key

Bob:

- has a message $m \in \mathbb{Z}_n^*$
- encrypts m as $c = m^e \text{ mod } n$
- sends c to Alice

Alice:

- decrypts c by computing $c^d = m^{ed} \text{ mod } n$
- $m^{ed} \text{ mod } n = m^{ed \text{ mod } \varphi(n)} \text{ mod } n = m \text{ mod } n$

Public exponent e can be fixed: Turns out, Alice can pick 3 as e, which will make encryption extremely fast, without loss of security.

- Ron Rivest disagrees about the security of $e = 3$ under some conditions in one of his papers.
- $e = 2^{16} - 1$ seems to be a good choice too.

Why can't the adversary compute d given (n, e)? Because he needs to compute $\varphi(n)$ in order to compute d using EEA, and that is equivalent to factoring n, a hard problem.

RSA attacks

Encrypting short messages using small *e*

Say $e = 3$, and $m < N^{\frac{1}{3}}$ is unknown to the attacker. Then $c = m^3 \text{ mod } N = m^3$ and so $m = \sqrt[3]{c}$

More general attack with small *e*

Let's extend the above attack for any message length of m.

Let $e = 3$. Suppose, three messages are sent to three parties encrypted with public keys $(N_1, 3)$, $(N_2, 3)$ and $(N_3, 3)$

$$c_i = m^3 \text{ mod } N_i$$

We'll assume $\gcd(N_i, N_j) = 1, \forall i, j$ since if that were not the case then you could easily factor one of the N_i's and easily recover m.

Let $N^* = N_1 N_2 N_3$. An extended version of the Chinese Remainder Theorem says there exists a $c < N^*$ such that:

$$c = c_i \text{ mod } N_i, \forall i$$

This c can be computed easily (no clue how) given the public keys and the ciphertexts.

Note that $c = m^3 \text{ mod } N^*$.

Since $m < \min\{N_1, N_2, N_3\}$ we have $m^3 < N^*$. We can now apply the previous attack to get m from c.

$$m = \sqrt[3]{c}$$

Quadratic improvement in recovering m

If $1 \le m < L$ (when interpreting m as an integer), then we can recover m in $\sqrt{L}$ time.

Attack: assume $m < 2^l$, so $L = 2^l$ and that the attacker knows l. $\alpha = constant, \frac{1}{2} < \alpha < 1$

- **Input:** Public key (N, e), ciphertext c and parameter l
- **Output:** $m < 2^l$ such that $m^e = c \bmod N$

```
T = 2^(αl)
for r = 1 to T:
    x_i = c/r^e mod N

sort the pairs {(r, x_r)}_{r=1}^T by their second component

for s = 1 to T:
    if s^e mod N = x_r for some r
        return rs mod N
```

Time complexity is dominated by the time taken to sort the $2^{\alpha l}$ pairs. Binary search is used to find whether $\exists r, x_r = s^e \bmod N$.

If m is chosen as a random l-bit integer, it can be shown that with good probability $\exists r, s$ with $1 < r, s < 2^{\alpha l}$ and $m = rs$. The algorithm essentially looks for these r and s values.

Common modulus attack I

Company shares keys to each employee i as $pk_i = (N, e_i)$ and $sk_i = (N, d_i)$

$$e_i d_i = 1 \bmod \phi(n), \forall i$$

So each employee has their e_i, d_i pair which means they can easily factor N, which allows the to obtain the decryption key of all the other employees by computing:

$$d_j = e_j \bmod \phi(n)$$

Common modulus attack II

Suppose m is encrypted and sent to two different employees with public keys (N, e_1) and(N, e_2) where $e_1 \neq e_2$. Further assume that $\gcd(e_1, e_2) = 1$

Eve sees two ciphertexts:

$c_1 = m^{e_1} \bmod N$ and $c_2 = m^{e_2} \bmod N$

$$\gcd(e_1, e_2) = 1 \Rightarrow \exists x, y, e_1 x + e_2 y = 1$$

Eve computes x and y using EEA. Then…

$$c_1^x c_2^y = (m^{e_1} \bmod N)^x (m^{e_2} \bmod N)^y = (m^{e_1 x} \bmod N)(m^{e_2 y} \bmod N) = m^{e_1 x} m^{e_2 y} \bmod N = m^{e_1 x + e_2 y} \bmod N = m$$

Alin Tomescu, CSE408
Tuesday, March 15th, Lecture #13

Weak and strong RSA assumptions

Weak RSA assumption: Given x, e, n (composite n) computing y such that $y^e = x \bmod n$ is really hard.

If I give you a composite n and you can find a d such that $3d = 1 \bmod \varphi(n)$, then you know that $\varphi(n) \mid 3d - 1$. It turns out that given this you can compute $\varphi(n)$, and given $\varphi(n)$ you can factor n.

Equivalence: Given e, n compute d (RSA problem), given n compute $\varphi(n)$, and factoring n are all equivalent problems.

Interesting fact: It is not known whether decrypting RSA is equivalent to factoring n. It is assumed decrypting RSA is a little easier actually.

Strong RSA assumption: Given x, n computing any $y, e \neq \pm 1$ s.t. $y^e = x \bmod n$ is really hard. The attacker is given more freedom here: he can actually choose e.

Note: With public key cryptography, the public key only has to be transmitted to Bob with integrity (Eve should not be able to modify it), no secrecy is needed.

El Gamal

Definition: El Gamal is an encryption system based on Diffie-Hellman.

We have Alice, Bob and global parameters p and g

Alice:

- picks a random a, her secret key
- she sends the public key $g^a \bmod p$ to Bob

Bob:

- To encrypt $m \in \mathbb{Z}_p^*$, Bob picks a random s
- sends $(g^s \bmod p, g^{as} \times m \bmod p)$ to Alice

Alice:

- decrypts by computing:

$$\frac{g^{as} \times m \bmod p}{(g^s \bmod p)^a} = \frac{g^{as} \times m \bmod p}{g^{as} \bmod p} = m \bmod p$$

CCA attack on El Gamal

Eve can always win the CCA game because she can decrypt the received challenge:

Let g and p be the public parameters. Alice has her secret key a and g^a is known by everyone

$$c = (g^s \bmod p, g^{as} \times m \bmod p)$$

Eve computes $c' = (g^s \bmod p, g^{as} \times m \times 2 \bmod p)$

Eve queries the decryption oracle with c' which will gladly decrypt it to $m' = m \times 2 \bmod p$.

Eve can now compute $m' \times 2^{-1} = m \bmod p$ getting m.