MATH 314 - Class Notes

4/20/2017

Scribe: Joshua McGiff

Summary: During this lecture we covered RSA more comprehensively and worked through an example. We also discussed some of the theory behind the algorithm and how to attack it. We concluded by revisiting primality testing with Fermat's theorem and introduced the Miller-Rabin algorithm.

Notes: Include detailed notes from the lecture or class activities. Format your notes nicely using latex such as

RSA recap:

- Asymmetric encryption algorithms utilize trapdoor functions –easy to do one way, but difficult to reverse.
- RSA uses integer factorization as its basis.
- Given two large prime numbers p and q, it is easy to calculate pq=n one way, but difficult to obtain p and q from factoring n

Process Overview:
Alice: Creating a key pair for RSA

1. Alice picks p,q at random and verifies their primality
2. Computes $n = pq$
3. Computes $\varphi(n) = (p-1)(n-1)$
4. Picks e such that $gcd(e, \varphi(n)) = 1$
5. Computes $d = e^{-1}mod(\varphi(n))$

Alice Then publishes her public key: (n, e)
Alice keeps secret her private key: (p, q, $\varphi(n), d$)

bob: Wants to Send a message to Alice
message = m

1. Computes $m^e = cmod(n)$
2. Sends message C to alice
3. Alice decrypts with $c^d \equiv (m^e)^d \equiv m^{ed} \equiv m\ mod(n)$

Example

- Lets first pick two random primes: p = 11, q = 5
- First compute n: $pq = 11 * 5 = 55$
- Compute $\varphi(n)$:$\varphi(n) = (p-1)(q-1) = 10 * 4 = 40$
- Choose e: we pick 7 and verify that gcd(7,55)=1 so 7 is good to use
- Compute Decryption Exponent: $d = e^{-1}(mod\varphi(n))$

$$e^{-1}(mod\varphi(n)) = 1/7(mod40)$$

Euclid's Algorithm:

$$40 = 5(7) + 5$$

$$7 = 1(5) + 2$$

$$5 = 2(2) + 1$$

Reverse Euclid for Inverse:

$$1 = 5 - 2(2)$$

$$1 = 5 - 2(7 - 5)$$

$$1 = 3(5) - 2(7)$$

$$1 = 3(40 - 5(7)) - 2(7)$$

$$1 = 3(40) - 17(7)$$

This gives us $1 \equiv -17(7)(mod40)$ and so $1/7 \equiv -17(mod40) \equiv 23(mod40)$ giving us d=23(mod 40)

Alice can now publish her public key: (55,7) She will hold on to her private key: (11, 5, 23)

<u>suppose Bob wants to send a message to Alice:</u>
m=13

1. Bob computes m^e(mod n) = 13^7(mod 55) = 7
2. Bob sends the ciphertext c=7 to Alice

<u>Alice decrypts:</u>

1. Alice raises the ciphertext to decryption exponent: $7^2 3$(mod 55) = 13
2. Alice successfully recovers the plaintext m=13

Why is RSA secure?

- Suppose Eve is attacking this system –What does she know?
- Public key, (n, e)
- Ciphertext - c
- Eve knows the encryption scheme $c = m^e$(mod n)

Eve would need to compute c^d to recover m. Alice does not know d, how could she compute it?
- To compute d she must know $\varphi(n)$ because $d = e^{-1}$(mod $\varphi(n)$)

- computing $\varphi(n)$ requires her to factor n which is very hard resulting in our security.

What about brute force?
- Eve realizes she cannot compute d
- She wants to try all possible values of m
- To do so she must compute each possible e for $m^e = c$ and check for a solution
- There are n possibilities since we are working mod n meaning with a large enough n value brute force is not feasible

some other notes:
- There may be as of yet undiscovered methods of integer factorization that could be efficient enough to render RSA obsolete// - Quantum computers may be able to compute integer factorization very quickly in the future, also rendering RSA and many other modern cryptosystems obsolete

Timing Attacks
- If you can accurately measure the amount of time encryption takes you can use that to break RSA
- A solution to this used in modern systems is to insert a random delay

Primality Testing
- For RSA we need a way to randomly choose large(100 digit) prime numbers
Method:

1. pick a random 120 digit number
2. check if prime
3. if prime then use, if not go to step 1

How do we check for Primality?
1.) Naive Method:check if n is divisible by any number between 1 and $\sqrt(n)$
$-Ifn = 120digitsthen\sqrt(n)has60digitsmakingthismethodfartooslow$

2.) Fermat's Primality Test
We know from Fermat's theorem that if p is prime then $a^{p-1} \equiv 1(\text{mod p})$
We May use this to check if our candidate numbers, n, are prime

1. pick a base, a, at random where 1¡a¡n-1
2. compute $a^n - 1 \equiv b(\text{mod n})$
3. if $b \equiv 1(\text{mod n})$ then n is probably prime
4. repeat steps 1-3 for several bases n to help avoid psuedo-primes

Remember: There exist composite numbers that will pass fermat's primality test for a given base. These are known as psuedo-primes.

Furthermore, there some composite numbers that will pass fermat's test for any base, these are known as carmichael numbers.

Because of these numbers several other algorithms have been put forward to efficiently test for primality.

Once such algorith to be covered in the future is the Miller-Rabin algorithm.
- Miller-Rabin algorithm has pseudo-primes but no carmicheal numbers.