MATH 314 Spring 2018 - Class Notes

11/12/2018

Scribe: Anastasiya Lazarenko

Summary: Discrete logarithm problem, Diffie-Hellman, and Baby Step Giant Step.

Notes:

Factoring n requires $O(e^{\log(n)})$ time. This takes too long to crack RSA with present technology.

There is an algorithm called Shor's algorithm that can factor numbers in polynomial time, but it is made for Quantum computers. Current Quantum computers can only work with numbers that are 2 bits long.

What are some options for public key cryptography?

Need a trapdoor function (easy to do in one direction, but hard to undo), which is the core of the public key.

Trapdoor function for RSA - multiplication/factoring.

Another trapdoor function: Discrete Lograithm problem.
Idea: given a, x, n,
compute $m = a^x \pmod n$ quickly.

Note: Modular exponentiation makes this computation fast – $O(\log(n))$.

What if instead, we know m, a, n, but not x. Is there a way to find x? That is, to solve $m \equiv a^x \pmod n$ for x?
What if this wasn't modulo n?
$y = a^x$
For example, $100 = 2^x$
Use logarithms:
$\ln(a^x) = x\ln(a)$
$x = \frac{\ln(y)}{\ln(a)} = \frac{\ln(100)}{\ln(2)}$

We want to try to do this with only discrete numbers.
Problem: There is no discrete version $\mod n$ of the ln() function.
This seems to be hard.

Example: Solve $6 \equiv 2^x \pmod{()}13$. That is, find an x that satisfies the equation.
Brute force: try values of x.

$x = 1$	$2^1 \equiv 2 \pmod{13}$
$x = 2$	$2^2 \equiv 4 \pmod{13}$
$x = 3$	$2^3 \equiv 8 \pmod{13}$
$x = 4$	$2^4 \equiv 16 \equiv 3 \pmod{13}$
$x = 5$	$2^5 \equiv 32 \equiv 6 \pmod{13}$

If n is big (100 digits), then brute force is not feasible.

We want to use this problem to make a cryptosystem.

How is RSA typically used? Usually, all RSA is used for is to send a key fot AES. Then, the remaining messages are sent using symmetric key cryptography.

Diffie-Hellman Key Exchange:

Uses the discrete logarithm problem to allow two people to securely share a key to be used for symmetric key cryptography.
Alice and Bob can't use it to send a message, but they can pick a secret key.

Steps of Diffie-Hellman:
Alice picks a big prime p.
She also picks a big primitive root $a \pmod{p}$. (Note: *primitive root* means that powers of a produce all residues $\pmod{p}$, which can be computed quickly.)
She tells Bob these numbers (in cleartext). Assume Eve knows these numbers too (not secret).
Alice picks a secret number A, $1 < A < p - 1$.
Bob also picks a secret number B, $1 < B < p - 1$.
Alice computes
$$C_1 \equiv a^A \pmod{p}$$
Bob computes
$$C_2 \equiv a^B \pmod{p}$$
Alice takes C_2 and computes
$$C_2^A \pmod{p}$$
Bob takes C_1 and computes
$$C_1^B \pmod{p}$$
Note:
$$C_2^A = (a^B)^A = a^{BA} = (a^A)^B = C_1^B$$
So, $C_1 = C_2$.
Now both Alice and Bob have the same number $K \equiv C_1^B \equiv C_2^A \pmod{p}$, which they can

use for AES.

Why can't Eve crack this?

Eve knows a, p, C_1, and C_2. She doesn't know A or B. If she knew A or B, she could compute

$$K \equiv C_1^B \equiv C_2^A \pmod{p}$$

How would she compute either A or B?
To find A, she needs to solve $C_1 \equiv a^A \pmod{p}$ for A. (Discrete log problem – hard)
To find B, she needs to solve $C_2 \equiv a^B \pmod{p}$ for A (also the discrete log problem!), which is hard.

Pros of Diffie-Hellman:
- Better against Quantum computers than RSA, although they still have the upper hand.

Cons:
- Can't send a message, have to rely on a symmetric key cryptosystem.

What if Eve is trying to solve the discrete log problem? Brute force is $O(p)$, but we can do better.

Baby Step Giant Step:
Eve wants to solve $y \equiv a^x \pmod{p}$ for x.
She knows y, a, and p.
She computes $N = \lceil \sqrt{p} \rceil$.
Now Eve computes 2 tables:

Baby Steps
$a^i \pmod{p}$
for i in $0 \le i < N$

Giant Steps
$y \cdot a^{-jN} \pmod{p}$
for j in $0 \le i < N$

Note: Each table has N rows. Every step of creating the tables is fast because modular exponentiation is fast.

Eve finds two matching rows in the two tables. From the Baby Steps table, Eve gets a value of $a^i \pmod{p}$. From the Giant Steps table, Eve gets a value of $y \cdot a^{-jN} \pmod{p}$ so that $a^i \equiv y \cdot a^{-jN} \pmod{p}$.

Eve has $a^i \equiv y \cdot a^{-jN} \pmod{p}$ and wants to solve for y.
Multiply both sides by a^{jN}.

$$a^i \cdot a^{jN} \equiv y \cdot a^{-jN} \cdot a^{jN} \pmod{p}$$
$$a^{i+jN} \equiv y \pmod{p}$$

where $i + jN$ is x, which is what Eve was trying to find.

How fast is this?
Each table requires N steps. Checking for matches in the two tables requires N steps (using a computer science hashing algorithm).
This means that the Baby Steps Giant Steps algorithm takes $O(N) = O(\sqrt{p})$ time.

Does Eve always get the answer?
Suppose that x exists and $x < p - 1 < (\sqrt{p})^2 - 1 < N^2$.
Write x in "base" N:
$x = i(N^0) + jN^1 = i + jN$ (because x had to have been a factor of N)
Note: i is 1's place in the table and j is N's place in the table.
If $y \equiv a^x \equiv a^{i+jN} \pmod{p}$, then these are the entries that show up in each table.
Therefore, the coinciding entries must exist and so x exists every time.