Math/CS 467: Factorization and primality testing

John Lesieutre
August 27, 2021

Announcements

- Today: a quick introduction, and some cryptography!

Welcome to Advanced Topics 2!

- Today:
  - Very brief intro
  - Run through the syllabus
  - Then get to work.
  - First topic: Math 467, cryptography.

First up

- The first topic is cryptographic: how can two parties agree on a shared secret key over a public channel?
- One ingredient is being able to compute $a^b \bmod n$ quickly – we worked through this on the worksheet.
- (Here "$a \bmod n$" just means the remainder when a is divided by n.)

Diffie–Hellman key exchange

- Alice and Bob want to establish a secret key that both of them know, but nobody else does, even people who can see their communications.
- This key can then be used to encrypt communications between them.
- (They can use a symmetric-key cipher like one-time pad, AES, Twofish.)

One-time pad

- Encode the message as a binary string
- Encode the key you just created as a binary string
- XOR these together, and send.
- To decode: XOR the key with the encrypted message.

Diffie–Hellman

- A&B agree on a prime p and primitive root g.
- Each one picks an integer x and y, but keeps it secret.
- A computes $A = g^x \pmod{p}$ and B computes $B = g^y \pmod{p}$.
- They trade numbers (in public)

Diffie–Hellman, 2

- A computes $B^x \pmod{p}$, and B computes $A^y \pmod{p}$.
- $B^x = (g^y)^x = (g^x)^y = A^y \pmod{y}$, so they both have the same number. Use as a key.

cont

- What does a listener know?
- p, g, A, and B.
- If they can get x or y from these we're busted.
- Solving for x in $A = g^x \pmod{p}$ is very hard.
- It's called the *discrete logarithm problem* and there's no good algorithm.

"Trapdoor functions"

- Roughly speaking, this is a function that is easy to compute the output for a given input, but hard to find the (unique) input for a given output.
- Discrete log: computing $A = g^x \pmod{p}$ is easy, finding x given $A = g^x \pmod{p}$ is intractible.
- Factorization: computing $N = pq$ is easy, computing p and q from N is hard.

Primitive roots

- For DH to really be secure, you might want g to be a *primitive root* mod p.
- Suppose p is prime. A is a *primitive root* mod p if $a^1, \ldots, a^{p-1}$ give all possible remainders mod p.
- Ex $p = 7$. 2 is not a primitive root:

$$2^1 \equiv 2, 2^2 \equiv 4, 2^3 \equiv 1, 2^4 \equiv 2, 2^5 \equiv 4, 2^6 \equiv 1$$

- Ex $p = 7$. 3 is a primitive root:

$$3^1 \equiv 3, 3^2 \equiv 2, 3^3 \equiv 6, 3^4 \equiv 4, 3^5 \equiv 5, 3^6 \equiv 1.$$

Next algorithm: RSA

- DH lets two people generate a shared (basically random) secret number.
- This could be vulnerable; e.g. a field agent must keep a number secret.
- RSA is an asymmetric encryption algorithm: the field agent can encrypt messages using a public key supplied by headquarters.
- In fact, anyone can send an encrypted message!
- But these can only by decrypted by someone who knows the private key, which the field agent doesn't

A fact

- If b and n are coprime (have no common factors), then there exists an "inverse of b mod n".
- This means a value of a for which $ab \equiv 1 \bmod n$.
- In fact, a can be found quickly using another algorithm, the Euclidean algorithm.
- We'll just take this for granted, though.

Euler ϕ

- Let $\phi(n)$ denote the number of positive integers less than or equal n and relatively prime to n.
- What's $\phi(10)$? $\phi(13)$?
- $\phi(10) = 4$: only $1, 3, 7, 9$ are coprime.
- $\phi(p) = p - 1$.

Euler ϕ

- Theorem: Let n and b be positive and coprime. Then $b^{\phi(n)} \equiv 1 \pmod{n}$.
- This is a version of Fermat's Little Theorem, in the case when n is a prime number.

Proof

- Let $t = \phi(n)$, and let $a_1, \ldots, a_t$ be the coprimes less than n.
- Set $r_i = ba_i \pmod{n}$.
- I claim that $r_i \neq r_j$ if $i \neq j$.
- Indeed, if $r_i = r_j \pmod{n}$, then $ba_i = ba_j \pmod{n}$, and can cancel the bs by multiplying by the inverse.

Pf, cont

- The r_i are also coprime to n
- So we have a set of $\phi(n)$ distinct things less than n and coprime to it.
- The r_i must just be the a_i but reordered!
- $a_1 a_2 \cdots a_t = r_1 r_2 \cdots r_t$

Pf, cont

$$\begin{aligned} a_1 a_2 \cdots a_t &= r_1 r_2 \cdots r_t \pmod{n} \\ r_1 \cdots r_t &= b^t a_1 \cdots a_t \pmod{n} \\ &= b^t r_1 \cdots r_t \pmod{n} \\ b^t &\equiv 1 \pmod{n}. \end{aligned}$$

- That's what we were trying to prove!

RSA

- The person receiving the messages chooses two very large primes p and q, with product n.
- The receiver picks another key e ("encryption")
- And computes a d so that $de \equiv 1 \pmod{\phi(n)}$.
- (This can be done with the Euclidean algorithm very easily; but we're only doing a crash course!)
- Receiver then publicizes the nubers n and e, but keeps d, p, q secret.

Sending a message

- Write message as a number M.
- (It needs to be less than p, q; really less than n is OK unless it happens to be divisible by p, q).
- Encoder computes $E = M^e \pmod{n}$, which you know how to do.

To decrypt

- Receiver gets E and computes $E^d \pmod{n}$.

$$E^d \equiv (M^e)^d \equiv M^{k\phi(n)+1} \equiv 1 \cdot M = M \pmod{n}.$$

- We know de is 1 more than a mult of $\phi(n)$, which is where k comes in.
- We know $M^{\phi(n)} = 1 \pmod{n}$.

What about d and e?

- Why do d and e exist?
- Theorem: given a and m coprime, there exists b so that $ab = 1 \pmod{m}$.
- Pf: By Euclid, find b and c so $ab + mc = 1$. Then $ab = 1 \pmod{w}$.

How to break it?

- It turns out (HW3!) that if you know $n = pq$, then $\phi(n) = (p-1)(q-1)$.
- In fact, knowing $\phi(n)$ lets you find n if you know n is a product of two primes.
- So if you know the factorization you can find d.

In practice

- This is kind of slow, in practice.
- In real life (SSL/TLS – puts the S in HTTPS) for example, what happens is this.
- Client generates a random number, encrypts it using the public key from a server, and then sends it over.

RSA digital signatures

- Suppose you want to "sign" a message M.
- First step: hash it. Digests it down to a short number H.
- Really we'll sign that M has that hash. Assumption is nobody else could come up with another function with the same hash value, because the function is so crazy.
- (In practice: md5, whatever.)

And then

- Signer coputes $\Sigma = H^d \pmod{n}$.
- That's the signatrue.
- We then publish M and Σ.

How to check it?

- Recipient computes hash of M, gets same H (hash function not secret).
- Computes $\Sigma^e \pmod{n}$. As before this should recover H.
- But there's no way somebody could have generated this Σ if they didn't know the private key d!
- (Really it tells us that they generated some message with hash H, and we assume that no other message would generate this.)

If I need to stall...

- Cover the Euclidean algorithm.