Cryptography

2/7/22 Monday

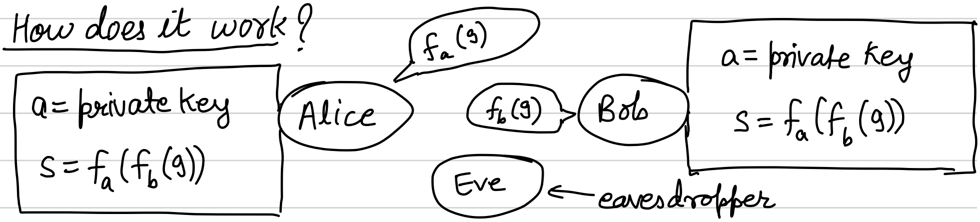
Lecture 1

"Two people who have never met, can exchange a secret in public"

1. Diffie-Hellman Handshake → 1976 – "New direction in cryptography"

↳ two computers can exchange information in the public domain while keeping the information secure

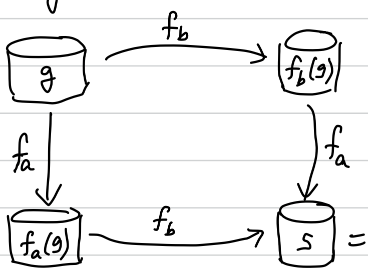
i) How does it work?



- "secret" = a big number; used for encrypting messages
- f_a : Alice's scrambling function
- g : initial seed (publicly agreed on)
- f_b : Bob's scrambling function

(*) If f_a, f_b and g are chosen well, then Eve cannot understand.

Analogy:



$= f_a(f_b(g)) = f_b(f_a(g))$

$\boxed{*}$ Eve cannot remove the pigment from the paint

ii) What we must do to make this precise?

- What are these scrambling functions f_a & f_b? (modular arithmetic §1)
  ↳ "modular exponentiation"
- Prove a theorem:
  – for all choices of a, b (private keys),

$$\underline{f_a(f_b(g)) = f_b(f_a(g))}$$ ← "hints at abstract algebra"

↳ "commutativity", "keep some structure"

no one can prove this

- Argue that Eve cannot extract a from $f_a(g)$ or s.

iii) Comments:

- Diffie & Hellman proposed this in 1976
  – "elliptic curve" version proposed in 1985;
  ↳ now widely used in most applications

2/8/22 Tuesday

LECTURE2

"hard and easy" computations

computing greatest common divisors of huge numbers (§1.2 textbook)

universe is $\approx 2^{60}$ seconds old
$\approx 2^{90}$ nanoseconds old

handy reference:
$2^{10} = 10^3$
$1024 = 1000$
$2^{20} \approx$ million
$2^{30} \approx$ billion

particles in universe $\approx 2^{266}$

$\Rightarrow$ you as "Eve" can never do $2^{90} \cdot 2^{266} \approx 2^{366}$ ops with computer ever because a single CPU in computer can do ≈ 1 "thing" per nanosecond

$\Rightarrow$ modern crypto applications: use number on scale of
$2^{256} \equiv$ "256-bit numbers"
or even $2^{3000} \equiv$ "3000-bit numbers"

throughout the course: understand what computations are "hard" which could mean an algorithm to do it would need to go for example 2^{300} things.

---

for comparision: in this course, we will write "toy models" of cryptographic algorithms that are used in practice & see how much they can do in ≈ 1 sec. on a single computer (can do $\approx 2^{30}$ things)

---

our first computational problem: greatest common divisor (see §1.2)

Definitions: $\mathbb{Z}$ = set of integers $\{\ldots, -3, -2, -1, 0, 1, 2, 3 \ldots\}$

[For $a, b \in \mathbb{Z}$, "a divides b" or "$a \mid b$" means that $\exists$ some $q \in \mathbb{Z}$ such that $b = aq$.
(in other words, $\frac{b}{a}$ is an integer)]

continued...

LECTURE 2 continued...

Given two integers a and b, not both 0, the greatest common divisor of a & b, denoted by $\gcd(a,b)$, is the maximum of the set of integers g that divides both a and b: $\{g \in \mathbb{Z} : g|a \text{ and } g|b\}$

eg. (edge case)... (kinda counter-intuitive at first)

For all $n \in \mathbb{Z}$, $n|0$ because $0 = n \cdot 0$
and $1|n$ because $n = 1 \cdot n$

Therefore, for all positive integers n, $\boxed{\gcd(n,0) = n}$

if $n<0$, $\gcd(n,0) = |n|$

(common divisors of n & 0 are "divisors of n" & n is largest one.)

$\Rightarrow$ Also, $\gcd(0,0)$ is undefined (or ∞, if you will).

A naive algorithm: "trial division"

Given two positive integers a,b: For all integers $g = 1, 2, \ldots, a$ check if g divides a and b. return largest found.

Python:

```
def gcd(a, b):
    bestyet = 1
    for g in range(1, a+1):
        if (a%g==0) and (b%g==0):
            bestyet = g
    return bestyet
```

(But if $a,b \sim 2^{1000}$, then this takes $\geq 2^{1000}$ steps but you can only do $\ll 2^{300}$ things)

(if you run exactly this in python, in 1 second, then on one machine it can finish on integers $a,b \approx 2^{36}$. ($< 2^{30}$))

Q. How can we do better? Ans: Euclidean Algorithm

preview:

eg $\gcd(77,42) = \gcd(42, 77-42) = \gcd(42,35)$
$= \gcd(35, 42-35) = \gcd(35,7)$
$= \gcd(7, 35-7*5) = \gcd(7,0)$ ← edge case!
$= \boxed{7}$ Ans

2/9/22 Wednesday

LECTURE 3

"Euclidean algorithm & extended version"

<u>q</u>: How can we compute $\gcd(a,b)$ when a, b are huge?

<u>Lemma</u>: For any $a, b \in \mathbb{Z}$ not both 0, and any other integer q,

$$\gcd(a,b) = \gcd(a-qb, b)$$

"small/helper theorem"

$\boxed{\text{Proof}}$ it suffices to prove

$$\underbrace{\{g \in \mathbb{Z} : g|a \,\&\, g|b\}}_{\text{common divisors of } a \,\&\, b} = \underbrace{\{g \in \mathbb{Z} : g|(a-qb) \,\&\, g|b\}}_{\text{common divisors of } a-qb \,\&\, b}$$

To prove two sets are equal: ① prove elements of left are elements of right "set A $\subseteq$ set B"
② prove the reverse "set B $\subseteq$ set A"

i) Suppose g is a common divisor of a & b.
Then $a = gA$ & $b = gB$ for some integers A & B
Therefore $a - qb = gA - qgB = g(A - qB)$, hence $g | (a-qb)$
Of course, $g|b$ and $g|(a-qb)$, g is common divisor of b & $(a-qb)$

ii) Suppose g is a common divisor of $a-qb$ & b
Then $\exists\, A, B \in \mathbb{Z}$ such that $a - qb = gA$ & $b = gB$
Therefore, $a = gA + qb = g(A + qB)$, hence $g|a$
Since, $g|b$ as well. $\Rightarrow g$ is common divisor of a & b

This shows,

$$\underbrace{\{g \in \mathbb{Z} : g|a \,\&\, g|b\}}_{\text{common divisors of } a \,\&\, b} = \underbrace{\{g \in \mathbb{Z} : g|(a-qb) \,\&\, g|b\}}_{\text{common divisors of } a-qb \,\&\, b}$$

so maximums are the same...

<u>i.e.</u> $\gcd(a,b) = \gcd(a-qb, b)$ ∎

<u>comment</u>: this is similar to row-reduction in linear algebra

→ can use this to quickly compute gcd

LECTURE 3 (continued)

Def^n: For $a,b \in \mathbb{Z}$, with $b>0$, $a\%b$ = the remainder when a is divided by b $= a - \lfloor \frac{a}{b} \rfloor \cdot b$

or "a mod b"
(% is Python syntax)

"floor" of $\frac{a}{b}$, means round down to an integer

Obs: $a\%b = 0$ iff $b|a$

eg $15\%7 = 1$ || $-2\%5 = (-2) - \lfloor \frac{-2}{5} \rfloor (5) = -2 - \lfloor -0.4 \rfloor (5) = -2 - (-1)(5) = 3$

observe: $a\%b = a - qb$ for $q = \lfloor \frac{a}{b} \rfloor$, so $\gcd(a,b) = \gcd(a\%b, b)$ by our lemma
$= \gcd(b, a\%b)$ writing big number first.

eg
$$\begin{aligned}
\gcd(42,25) &= \gcd(25, \underbrace{42\%25}_{=17}) \\
&= \gcd(17, \underbrace{25\%17}_{=8}) \\
&= \gcd(8, \underbrace{17\%8}_{=1}) \\
&= \gcd(1, \underbrace{8\%1}_{=0}) \\
&= \boxed{1}
\end{aligned}$$

A useful organisation:

$42 = r_0$
$25 = r_1$
$42\%25 = 17 = r_2$
$25\%17 = 8 = r_3$
$17\%8 = \boxed{1 = r_4}$
$8\%1 = 0 = r_5$
here $\boxed{t=4}$

gcd(42,25) = gcd of any two adjacent numbers in the list on the left.

Euclidean Algorithm, version 1

Given positive integers a & b, define $r_0 = a$, $r_1 = b$, and $\left[r_{n+1} = r_{n-1} \% r_n\right]$ for $n = 1,2,3\ldots$ until I reach $r_{t+1} = 0$ (for some t)

Then $\gcd(a,b) = \gcd(r_n, r_{n+1})$ for all $n = 0,1,\ldots,t$

In particular, $\gcd(a,b) = \gcd(r_t, r_{t+1}) = \gcd(r_t, 0) = r_t$

Euclidean Algorithm, version 2

Given positive integers a,b, repeatedly replace a,b by $b, a\%b$ (respectively) until I reach $g,0$ and g is the gcd.

Python

```python
def gcd(a,b):
    while b!=0:
        a,b = b, a%b
    return a
```