Math 1580: Cryptography *Lecture Notes*

E. Larson

Spring 2022

These are lecture notes for Math 1580: Cryptography taught at Brown University by Eric Larson in the Spring of 2022.

Notes last updated February 14, 2022.

Contents

§0 January 26, 2022

§0.1 Course Logistics

§0.2 Introduction

§0.3 Simple Substitution Ciphers

§0.4 Divisibility

§1 January 28, 2022

§1.1 Greatest Common Divisors

§1.2 Euclidean Algorithm

§1.3 Linear Combinations

§2 January 31, 2022

§2.1 Linear Combinations *continued*

Recall from last time that we proposed that

$$\text{greatest common divisor} \leq \text{least linear combination}.$$

Example 2.1

$\gcd(2024, 748) = 44$ because we have

$$\begin{aligned} 2024 &= 748 \cdot 2 + 528 \\ 748 &= 528 \cdot 1 + 220 \\ 528 &= 220 \cdot 2 + 88 \\ 220 &= 88 \cdot 2 + \boxed{44} \leftarrow \gcd(2024, 748) \\ 88 &= 44 \cdot 2 + 0 \end{aligned}$$

We determine which linear combinations or 2024 znd 748 we can create:

$$\begin{aligned}
2024 &= 1 \cdot 2024 + 0 \cdot 748 \\
748 &= 0 \cdot 2024 + 1 \cdot 748 \\
528 &= 1 \cdot 2024 + (-2) \cdot 748 \\
220 &= 748 - 1 \cdot 528 \\
&= 748 - 1 \cdot (1 \cdot 2024 + (-2) \cdot 748) \\
&= -1 \cdot 2024 + 3 \cdot 748 \\
88 &= 528 - 2 \cdot 220 \\
&= \underbrace{[1 \cdot 2024 + (-2) \cdot 748]}_{528} - 2 \cdot \underbrace{[-1 \cdot 2024 + 3 \cdot 748]}_{220} \\
&= 3 \cdot 2024 - 8 \cdot 748 \\
44 &= 220 - 2 \cdot 88 \\
&= [-1 \cdot 2024 + 3 \cdot 748] - 2 \cdot [3 \cdot 2024 - 8 \cdot 748] \\
&= -7 \cdot 2024 + 19 \cdot 748
\end{aligned}$$

Following this example, we have shown that every common divisor of a and b can be written as a linear combination of a and b, and since the greatest common divisor has to be less than the least linear combination (as shown last time), the greatest common divisor *is* the least linear combination[1].

We realize that there is a *recurrence* happening here. If we call every set of coefficients x, y and z, w for a and b respectively, such that

$$\begin{aligned}
a &= x \cdot a_0 + y \cdot b_0 \\
b &= z \cdot a_0 + y \cdot b_0
\end{aligned}$$

where a_0 and b_0 are the original numbers, we can use a sliding window approach[2] again to determine the next set of x, y, z, w, a, b.

Recall from last time we had

$$\begin{aligned}
a' &= b \\
b' &= a \mod b
\end{aligned}$$

We can extend this algorithm for our new coefficients:

$$\begin{aligned}
x' &= z \\
y' &= w \\
z' &= w - \left\lfloor \frac{a}{b} \right\rfloor \cdot z \\
w' &= y - \left\lfloor \frac{a}{b} \right\rfloor \cdot w
\end{aligned}$$

[1] Assume for contradiction that the gcd were any less, then that would also be a linear combination. $\lightning$

[2] Updating our iterators on every loop by sliding our window of coefficients down.

where $\left\lfloor \frac{a}{b} \right\rfloor$ are the quotients from our Euclidean Algorithm. Note that initially, we have

$$a = 1 \cdot a_0 + 0 \cdot b_0$$
$$b = 0 \cdot a_0 + 1 \cdot b_0$$

so we have initial values of $x = 1, y = 0, z = 0, w = 0$.

so our code for the *extended Euclidean Algorithm* is now

Algorithm 2.2 (Extended Euclidean Algorithm) —

```
def ext_gcd(a, b):
    x, y, z, w = 1, 0, 0, 1
    while b != 0:
        x, y, z, w = z, w, w - (a // b) * z, y - (a // b) * w
        a, b = b, a % b
    return (x, y)
```

§2.2 Modular Arithmetic

Recall: We used a substitution/shift cipher to encrypt text:

$$\begin{array}{ccc} \texttt{Y} & \texttt{E} & \texttt{S} \\ \downarrow & \downarrow & \downarrow \\ \texttt{D} & \texttt{J} & \texttt{X} \end{array}$$

by incrementing 5 letters for each lecture.

$a = 0,\ b = 1,\ \ldots,\ z = 25$.

We had this notion of

$$\text{ciphertext} = \text{plaintext} + 5$$
$$\texttt{d} = \texttt{y} + 5$$
$$3 = 24 + 5 = 29$$

Definition 2.3

We say $a \equiv b \mod m$ if $m \mid a - b$.

We say "a is congruent[a] to b modulo m".

[a] Congruence is a "behave like" equality.

Example 2.4

$$\begin{aligned} 24 + 5 &\equiv 3 \mod 26 \\ 22 + 2 &\equiv 1 \mod 12 \end{aligned}$$

The first example is from our shift sipher, the second example is equivalent to "two hours after 11:00, it is 1:00".

Proposition 2.5

If we have

$$\begin{aligned} a_1 &\equiv a_2 \mod m \\ b_1 &\equiv b_2 \mod m \end{aligned}$$

Then we have the following:

$$a_1 + b_1 \equiv a_2 + b_2 \mod m \tag{1}$$
$$a_1 - b_1 \equiv a_2 - b_2 \mod m \tag{2}$$
$$a_1 \cdot b_1 \equiv a_2 \cdot b_2 \mod m \tag{3}$$

Proof. For eq. (1), realize that we have

$$(a_1 + b_1) - (a_2 + b_2) = (a_1 - a_2) + (b_1 - b_2)$$

and the two terms on the right are each divisible by m by our premise. We can also write out

$$\begin{aligned} a_1 + b_1 &= (a_2 + \alpha m) + (b_2 + \beta m) \\ &= (a_2 + b_2) + (\alpha + \beta) \cdot m. \end{aligned}$$

Similarly, for eq. (2), we have

$$\begin{aligned} a_1 - b_1 &= a_2 + \alpha m - (b_2 + \beta m) \\ &= a_2 - b_2 + (\alpha - \beta) \cdot m. \end{aligned}$$

and for eq. (3), we have

$$\begin{aligned} a_1 \cdot b_1 &= (a_2 + \alpha m) \cdot (b_2 + \beta m) \\ &= a_2 \cdot b_2 + \alpha m b_2 + \beta m a_2 + \alpha\beta m^2 \\ &= a_2 \cdot b_2 + (\alpha b_2 + \beta a_2 + \alpha\beta m) \cdot m. \end{aligned}$$

which concludes the proofs of the premod rules. $\square$

Proposition 2.6

There exists b with

$$a \cdot b \equiv 1 \mod m$$

if and only if $\gcd(a, m) = 1$.

Proof. We can write linear combination equation

$$a \cdot b + m \cdot k = 1$$

and we have that the following are equivalent (we cascade down the list and can easily prove the iff relations):

i. such a b exists,

ii. there is a solution b, k to this equation,

iii. 1 is a linear combination of a and m,

iv. 1 is the *least* linear combination of a and m,

v. $1 = \gcd(a, m)$.

so we have that $1 = \gcd(a, m)$ if and only if a's inverse b exists. □

§3 February 2, 2022

§3.1 Inverses mod m

Recall: Last time, we showed in proposition 2.6 that there exists an integer b with with $a \cdot b \equiv 1 \mod m$ iff $\gcd(a, m) = 1$.

Claim 3.1 — We further claim that if such a b exists, then it is unique mod m.

That is, if we have

$$\begin{aligned} a \cdot b_1 &\equiv 1 \pmod m \\ a \cdot b_2 &\equiv 1 \pmod m \end{aligned}$$

then we have that $b_1 \equiv b_2 \pmod m$.

Proof. We consider $b_1 a b_2$. We have

$$b_2 \equiv (b_1 a) b_2 = b_2 (a b_2) \equiv b_2$$

all taking mod m. $\square$

How, then, could we compute this inverse b efficiently?

Recall that last class, we used the extended Euclidean algorithm to compute the linear combination of a and m efficiently,

$$\begin{aligned} 1 &= a \cdot u + m \cdot v \\ &\equiv a \cdot \boxed{u} \mod m \end{aligned}$$

where u is b.

§3.2 Modular Arithmetic *continued*

Definition 3.2 (Ring of Integers mod m)
$\mathbb{Z}/m\mathbb{Z} = \{0, 1, 2, \ldots, m-1\}$ with operations $+, -, \times \pmod{m}$.

Example 3.3
$\mathbb{Z}/4\mathbb{Z} = \{0, 1, 2, 3\}$. We have the following operation tables for $\mathbb{Z}/4\mathbb{Z}$:

$+$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$\times$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Definition 3.4 (Group of Units mod m)
We have the set of units in $\mathbb{Z}/m\mathbb{Z}$ as

$$\begin{aligned} (\mathbb{Z}/m\mathbb{Z})^\times &= \{a \in \mathbb{Z}/m\mathbb{Z} \mid \exists b \text{s.t. } a \cdot b \equiv 1\} \\ &= \{a \in \mathbb{Z}/m\mathbb{Z} \mid \gcd(a, m) = 1\} \end{aligned}$$

Example 3.5
$(\mathbb{Z}/4\mathbb{Z})^\times = \{1, 3\}$.

Definition 3.6 (Euler Totient Function)
We have

$$\varphi(m) = \#(\mathbb{Z}/m\mathbb{Z})^\times$$

which counts the number of units modulo m.

Example 3.7

$\varphi(4) = 2$.

Let's investigate the properties of units. Let's say a_1, a_2 are units. Which of the following have to be units?

	Does this have to be a unit?
$a_1 \cdot a_2$	Yes! Since $\gcd(a_1, m) = 1$ and $\gcd(a_2, m) = 2$ so we have $\gcd(a_1 a_2, m) = 1$. We also have $a_1 b_1 \equiv 1 \mod m$ and $a_2 b_2 \equiv 1 \mod m$, we have $(a_1 a_2)(b_2 b_1) \equiv 1 \mod m$.
$a_1 + a_2$	No. We have counterexample $m = 4$: $1 + 1$ is not a unit.
$a_1 - a_2$	Also no. For any a, $a - a = 0$ which is never a unit.

Definition 3.8 (Prime Number)

An integer $n \geq 2$ is prime if its only (positive) divisors are 1 and n.

Example 3.9

Numbers like $2, 3, 5, 7, 11, 12, \ldots$.

What if m is a prime number? Then we have

$$(\mathbb{Z}/m\mathbb{Z})^\times = \{1, 2, \ldots, m-1\}$$

so we can divide by elements of $\mathbb{Z}/m\mathbb{Z}$, just like in $\mathbb{Q}, \mathbb{R}, \mathbb{C}$. We can divide by any nonzero element of $\mathbb{Z}/m\mathbb{Z}$. We call these fields!

§3.3 Fast*ish* Powering

Problem. How might we compute $g^a \mod m$?

A naïve solution might be

```
def pow_mod(g, a, m):
    return g ** a % m
```

What if we tried to compute pow_mod(239418762304, 12349876234, 12394876123482783641) or something of the like? Something like this...

We could do something a bit more clever, like taking a mod every time we multiply:

```
def pow_mod(g, a, m):
    p = 1
    for i in range(a):
        p = (p * q) % m
    return p
```

Yet we *still* couldn't do `pow_mod(239418762304, 12349876234, 12394876123482783641)` since that takes the amount of time proportional to `a`[3].

Example 3.10

Let's try to compute 3^{37} by hand.

$$\begin{aligned}
3^1 & \equiv 3 \mod 100 \\
3^2 & \equiv 9 \mod 100 \\
3^4 = (3^2)^2 = & \equiv 81 \mod 100 \\
3^8 = (3^4)^2 = 81^2 = 6561 & \equiv 61 \mod 100 \\
3^{16} = (3^8)^2 \equiv 61^2 = 3721 & \equiv 21 \mod 100 \\
3^{32} = (3^{16})^2 \equiv 21^2 = 441 & \equiv 41 \mod 100
\end{aligned}$$

Since $37 = 32 + 4 + 1$, we can simply do

$$3^{37} = 3^{32} \cdot 3^4 \cdot 3^1 = 41 \cdot 81 \cdot 3 = 1863 \equiv 63 \mod 100$$

§4 February 4, 2022

§4.1 Fast Powering *continued*

[3]Which can become big. . .

Example 4.1

Recall: we wanted to compute $3^{37} \mod 100$

$$\begin{aligned} 3^1 &\equiv 3 \pmod{100} \\ 3^2 &\equiv 9 \\ 3^4 &\equiv 81 \\ 3^8 &\equiv 61 \\ 3^{16} &\equiv 21 \\ 3^{32} &\equiv 41 \end{aligned}$$

so we have

$$37 = 1 + 4 + 32 \qquad 3^{37} = 3^1 \cdot 3^4 \cdot 3^{32} \equiv 3 \cdot 81 \cdot 41 \equiv 63$$

How might we do this as an algorithm? We want to keep track of a few things, such as g (the current power), p (the multiple we are building), a (the remaining powers). This is akin to *deconstructing the power in binary and composing our product.*

Algorithm 4.2 (Fast Powering Algorithm) —

```
def pow_mod(g, a, m):
    p = 1
    while a != 0:
        if a % 2 == 1:
            p = (p * g) % m
        a = a // 2
        g = g**2 % m
    return p
```

Example 4.3

$37 = 100101_2$, so we peel off last digits and multiply g into p.

Thinking about iterations, we have

g	p	a	a_2
3	1	37	$10010\underline{1}$
9	3	18	$10010\underline{0}$
81	3	9	$100\underline{1}$
61	43	4	$10\underline{0}$
21	43	2	$1\underline{0}$
41	43	1	$\underline{1}$
	$\boxed{63}$	0	$\underline{0}$

This algorithm takes approximately $\log_2(a)$ time to run, since it does as many steps for each digit in the binary representation of a.

§4.2 Fun Integers

Recall: An integer p is prime if $p \geq 2$ and

$$a \mid p \Rightarrow a = \pm 1, \pm p$$

Proposition 4.4

Let p be prime. Then $p \mid ab \Rightarrow p \mid a$ or $p \mid b$.

Example 4.5

p is not prime, this doesn't work. $p = 6$. $p \mid 4 \cdot 9 = 36$ but $6 \nmid 4$ and $6 \nmid 9$.

Proof. Let $g = \gcd(p, a)$. g is either 1 or p.

If $g = p$, then we have that $p = g \mid a$.

If $p = 1$, we can write this as

$$\begin{aligned} 1 = g &= p \cdot u + a \cdot v \\ b &= p \cdot ub + ab \cdot v \end{aligned}$$

since p is a multiple of p and ab is a multiple of p, we have that $p \mid b$. $\square$

Theorem 4.6 (Fundamental Theorem of Arithmetic)

Any integer $a \geq 1$ can be factored into product of primes

$$a = p_1^{e_1} \cdots p_n^{e_n}$$

and this product of primes is *unique* up to rearrangement.[a]

[a] This is to say, $\mathbb{Z}$ is a UFD!

Example 4.7

Instead of thinking about integers, we think about $\mathbb{Z}[\sqrt{-5}]$, like

$$\mathbb{Z}[\sqrt{-5}] = \{a + b\sqrt{-5} \mid a, b \in \mathbb{Z}\}$$

Consider

$$6 = (1 + \sqrt{-5})(1 - \sqrt{-5}) = 2 \cdot 3$$

and each of $(1+\sqrt{-5})$, $(1-\sqrt{-5})$, 2, 3 have no divisors besides themselves and ± 1 (units).

Proof. We begin by working out an example:

Example 4.8
Let's factor 60, we can write this as

$$60 = 6 \cdot 10 = (2 \cdot 3) \cdot (2 \cdot 5) = 2^2 \cdot 3 \cdot 5.$$

What if we had different answers

$$p_1 p_2 \cdots p_t = a = q_1 q_2 \cdots q_s$$

We have that

$$\begin{aligned} p_1 \mid p_1 \cdots p_t &= q_1 \cdots q_s \\ &= q_1(q_2 \cdots q_s) \end{aligned}$$

So we have that $p_1 \mid q_1$ or $p_1 \mid q_2 \cdots q_s$, and we go on. So p_1 has to divide *one* of q_i. But both are primes, so they are equal $p_1 = q_i$. We rearrange so q_i is q_1. We strip off p_1 and q_1 and we have

$$p_2 \cdots p_t = q_2 \cdots q_s$$

we continue until we have no factors left[4] □

Definition 4.9 (Order)
We define the <u>order</u>

$$\mathrm{ord}_p(a) = \text{the power of } p \text{ in the factorization of } a$$

such that we have

$$a = \prod_p p^{\mathrm{ord}_p(a)}$$

(This makes sense since $\mathrm{ord}_p(a)$ *is finite for finitely many* p*.)*

Theorem 4.10 (Fermat's Little Theorem)
Let p be prime, $a \in \mathbb{Z}/p\mathbb{Z}$,

$$a^{p-1} \equiv \begin{cases} 0 & \text{if } a \equiv 0 \\ 1 & \text{otherwise} \end{cases}$$

[4] We could also have taken a well-ordering approach to this statement, taking a to be the least such non-uniquely factorizable number and showing that by peeling off p_1 and q_1, we get a smaller such a, which is a contradiction.

In abstract algebra, this directly follows from Lagrange's Theorem for $\mathbb{Z}/p\mathbb{Z}$, we give another argument.

Proof. If $a \equiv 0$, this is sufficiently clear.

Let $a \not\equiv 0$. We look at the numbers

$$a, 2a, 3a, \ldots, (p-1)a$$

We consider 2 questions:

i. Are any of these divisible by p?

   No! $p \nmid a$ and $p \nmid i$ so $p \nmid ia$ for $1 \le i < p$.

ii. Are any of these equal? i.e. $ia \equiv ja \mod p$.

   No again! a has an inverse mod p.

So we have that this list is a permutation of $\{1, 2, \ldots, p-1\}$, that is,

$$\{1, 2, \ldots, p-1\} = \{a, 2a, \ldots, (p-1)a\} \mod p$$

we multiply these sets together[5],

$$\begin{aligned} 1 \cdot 2 \cdot 3 \cdots (p-1) &\equiv a \cdot 2a \cdots (p-1)a \mod p \\ &\equiv (1 \cdot 2 \cdots p-1)a^{p-1} 1 \cdot 2 \cdot 3 \cdot (p-1)(a^{p-1}-1) \equiv 0 \mod p \\ \implies a^{p-1} &\equiv 1 \mod p. \end{aligned}$$

Which is as desired. $\square$

§5 February 7, 2022

§5.1 Orders mod p

Recall: If $a \not\equiv 0 \pmod p$, then we have $a^{p-1} \equiv 1 \pmod p$, which was theorem 4.10, Fermat's Little Theorem.

Definition 5.1 (Order of a mod p)
The order of $a \pmod p$ is the smallest positive k such that

$$a^k \equiv 1 \pmod p$$

This is not to be confused with definition 4.9 which is the power of p in the prime factorization of a. This is the order of a in the multiplicative group $\mathbb{Z}/p\mathbb{Z}$.

[5] This is truly a pro-gamer move

Proposition 5.2
let $a \in (\mathbb{Z}/p\mathbb{Z})^\times$ be of order k. If $a^n \equiv 1 \pmod p$, then $k \mid n$.

In particular, $k \mid p-1$ by theorem 4.10, Fermat's Little Theorem.

Proof. We write $n = k \cdot q + r$ such that $0 \le r < k$ ($\mathbb{Z}$ is a Euclidean domain)

$$1 \equiv a^n \equiv a^{kq+r} \equiv (a^k)^q \cdot a^r \equiv a^r$$

Since k is the minimal positive number such that $a^k \equiv 1$, then this forces $r = 0$. Then $k \mid n$. □

Theorem 5.3 (Primitive Root Theorem)
Let p be prime. Then there is a g such that

$$(\mathbb{Z}/p\mathbb{Z})^\times = \{1, g, g^2, \ldots, g^{p-2}\}.$$

We call g a primitive root or generator.

Example 5.4
$p = 5$, $(\mathbb{Z}/5\mathbb{Z})^\times = \{1, 2, 3, 4\}$.

1? No: $\{1, 1^2, 1^3\} = \{1\}$

2? Yes: $\{1, 2, 2^2, 2^3\} = \{1, 2, 4, 3\}$

3? Yes: $\{1, 3, 3^2, 3^3\} = \{1, 3, 4, 2\}$

4? No: $\{1, 4, 4^2, 4^3\} = \{1, 4\}$

Remark 5.5. In general, the number of primitive roots is $\varphi(p-1)$. (Take the group of exponents and solve for power).

§5.2 Discrete Logarithm Problem

We go on to discuss a fundamental property about exponentiation mod p. Let's fix some p and primitive root g.

Given some a, we can compute g^a efficiently

$$a \longrightarrow g^a \qquad \text{This is easy}$$
$$a \xleftarrow{?} g^a \qquad \text{This is hard}$$

Note that
$$g^a \equiv g^b \Leftrightarrow g^{a-b} \equiv 1 \Leftrightarrow p-1 \mid a-b$$
so a is determined mod $p-1$.

Definition 5.6 (Discrete Logarithm)
The discrete logarithm of g^a is a.

This is known as the "Discrete Logarithm Problem" (DLP), which is concerned with how we can compute discrete logarithms.

This idea is fundamental to computer security! The real-world analogue is if you go to the bank after hours and deposit a check or cash into the deposit slot. It is relatively easy for one to deposit an item but hard for someone who doesn't work at the bank[6] to access that item.

§5.3 Cryptographic Systems

§5.3.1 Symmetric Cryptography

We have 3 people, *Alice*, *Bob*, and *Eve*.

Bob has a message m which he wants to send to Alice. However, everything he sends to Alice can (and is) intercepted by Eve. He wants to encrypt this message m he sends to Alice.

We say that a message $m \in \mathcal{M}$ in the space of possible messages. We have secret key $k \in \mathcal{K}$ that can encrypt m into ciphertext $c \in \mathcal{C}$ in the space of ciphertexts.

$$\left\{ \begin{array}{c} \text{Message } m \in \mathcal{M} \\ \text{Secret key } k \in \mathcal{K} \end{array} \right\} \rightsquigarrow \text{Ciphertext } c \in \mathcal{C} \longrightarrow \text{Alice} \rightsquigarrow m$$

If we fix k, we have
$$e_k(m) = e(k,m)$$
$$d_k(c) = d(k,c)$$

be our encryption and decryption functions. We usually take m to be a number, and we can encode letters to numbers (0-255) using ASCII.

In Python, this is implemented using functions like `ord` (character to encoding) and `chr` (encoding to character).

We'll just talk about transmitting numbers since we can convert freely between them and text.

[6] Say, possessing a *key* or *password*.

Q: What do we want out of our cryptosystem?

0. The system is secure even if Eve knows the design. (Assume Eve knows the encryption and decryption functions, but so long as she doesn't know the key).

1. e, the encryption function, is easy to compute.

2. d, the decryption function, is similarly easy to compute.

3. Given $c_1, c_2, \ldots$ a collection of ciphertexts, encrypted with the *same* key k, it's hard to compute any message m_i.

4. Given $(m_1, c_1), \ldots (m_n, c_n)$ some collection of messages and their encryptions, it remains difficult to compute $d_k(c)$ for $c \notin \{c_1, \ldots, c_n\}$. This is called a *"chosen plaintext attack"*.

§6 February 9, 2022

§6.1 Asymmetric/Public Key Cryptography

The premise is that we have *Alice* and *Bob* who are communicating, and *Eve* intercepts all communications between them. There is **no** communication between Alice and Bob ahead of time. A priori, it's not entirely obvious that this is possible...

We'll see that this is indeed possible!

Example 6.1

Analogy: Alice and Bob are communicating by writing messages on pieces of paper.

Symmetric cryptography is having a shared safe, Alice and Bob both have the key/know the combination to, and both can leave messages and retrieve messages.

1. Alice sets up a box with a thin slot with a lock on it. Alice has the key to this lock.

2. Bob is able to deposit messages into the slot in the box, and Alice can retrieve it using her key.

Our key is now $k = (k_{\mathsf{priv}}, k_{\mathsf{pub}}) \in \mathcal{K} = \mathcal{K}_{\mathsf{priv}} \times \mathcal{K}_{\mathsf{pub}}$ which consists of a private key and public key.

Our encryption and decryption functions are now

$$e : \mathcal{K}_{\mathsf{pub}} \times \mathcal{M} \to \mathcal{C}$$
$$d : \mathcal{K}_{\mathsf{priv}} \times \mathcal{C} \to \mathcal{M}$$

$$d(k_{\mathsf{priv}}, e(k_{\mathsf{pub}}, m)) = m$$

We want it to be easy to compute $e_{k_{\mathsf{pub}}}$ and $d_{k_{\mathsf{priv}}}$, but hard to compute $d_{k_{\mathsf{priv}}}$ only knowing k_{pub}.

Something easier to construct, before a full-fledged public key system, is a key exchange:

§6.2 Diffie-Hellman Key Exchange

Q: How can Alice and Bob agree on a secret key over an insecure channel?

Example 6.2

Analogy: A lockbox that can only be used by one person...and both people have to participate to set it up.

Both parties have to agree on a key and have a line of communication before agreeing on a key. This can only be used if both parties are online at the same time.

We start with a prime p and $g \in (\mathbb{Z}/p\mathbb{Z})^\times$ suitably. Alice and bob does the following, all mod p:

Alice	Bob
Generates a	Generates b
$\downarrow$	$\downarrow$
Computes g^a	Computes g^b
Send g^a to Bob	Send g^b to Alice
Computes $(g^b)^a$	Computes $(g^a)^b$

Alice and Bob now know g^{ab}, which is the secret key. Eve, however, only knows g^a and g^b. Alice and Bob can now use this shared secret g^{ab} as a key for symmetric cryptography.

Definition 6.3 (The Diffie-Hellman Problem (DHP))

Given g^a, g^b, calculate g^{ab}.

Remark 6.4. If we can solve the discrete log problem, we can solve the Diffie-Hellman problem.

Vice versa? Can one solve DLP given solution to DHP? *This is unknown*[7].

[7] There is no known method.

§6.3 Elgamal Public Key Cryptography

We again start with p prime and $g \in (\mathbb{Z}/p\mathbb{Z})^\times$ suitably. This could be public knowledge, or Alice selects these.

Alice: We have a be Alice's private key, and $A = g^a$ be Alice's public key.

Bob: Has message m he wishes to send. Bob does the following:

1. Generate random k (used only once, to send this message).
2. Compute the following:
   a) $c_1 = g^k \mod p$
   b) $c_2 = m \cdot A^k \mod p$
3. Send c_1 and c_2 to Alice.

Alice:

$$(c_1^a) = A^k \text{ so } c_2 \cdot (c_1^a)^{-1} \equiv m\left((g^a)^k\right) \cdot \left((g^a)^k\right)^{-1} \equiv m$$

Basically, they are using Diffie-Hellman key exchange, except g^a is a public key and Bob assumes a secret key, and uses that to encrypt the message and sends it in one go.

§6.3.1 Implementation

We have the following algorithm for encryption and decryption in Elgamal:

```
import ext_gcd, pow_mod
from random import randrange
def e(A, m):
    k = randrange(p)
    return (pow_mod(g, k, p), m * pow_mod(A, k, p))

def d(a, c):
    pow_mod(c[0])
    ...
```

to be continued...

§7 February 11, 2022

§7.1 Elgamal *continued*

Recall: we perform Elgamal by starting with a prime p and $g \in (\mathbb{Z}/p\mathbb{Z})^\times$ which is *public knowledge.*

Alice computes a which is her *private key*, and $A = g^a$ which is her *public key.*

Encryption: Bob generates a random k and sends Alice

$$c_0 \equiv g^k \mod p \qquad c_1 \equiv mA^k \mod p$$

Decryption: Alice computes

$$c_1 \cdot (c_0^a)^{-1} \equiv m(g^a)^k \left((g^k)^a\right)^{-1}$$

We continue as we did from last time:

```
import ext_gcd, pow_mod
from random import randrange
def e(A, m):
    k = randrange(p)
    return (pow_mod(g, k, p), m * pow_mod(A, k, p))

def d(a, c):
    return c[1] * ext_gcd(pow_mod(c[0], a, p), p)[0]
```

Which works as intended (try it out!).

We note a property of Elgamal that there is an expansion factor of 2. It takes *twice* as much space to store c as m. We note that the expansion factor is always at least 1 (otherwise, we wouldn't be able to invert it).

§7.2 Midterm Details

Feb 16 @ 2pm in class. If remote, send email.

Topics will include: *everything up to now* (literally right now).

Focus: More theoretical, less computational. (Both are fair game!)

Resources: Pen/pencil, paper. No notes and no book. Nothing else.

Weighting: 20% Midterm 1 and 30% on Final. 30% Midterm 2, 20% Homework. Half on written and half on in-class exams.

Problem set #3 which is shorter than #2. (Good practice!)

Midterm results/curve will be announced hopefully by Friday after the midterm.

§7.3 Introduction to Group Theory

Groups are an algebraic structure... they're sets endowed with an operation.

Example 7.1

We have that $(\mathbb{Z}/p\mathbb{Z}, +)$ and $((\mathbb{Z}/p\mathbb{Z})^\times, \cdot)$ are both groups.

	$(\mathbb{Z}/p\mathbb{Z}, +)$	$((\mathbb{Z}/p\mathbb{Z})^\times, \cdot)$
Identity:	$0 + a = a$	$1 \cdot a = a$
Inverse:	$a + (-a) = (-a) + a = 0$	$a \cdot a^{-1} = a^{-1} \cdot a = 1$
Associative:	$a + (b + c) = (a + b) + c$	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
Commutative:	$a + b = b + a$	$a \cdot b = b \cdot a$

Definition 7.2 (Group)

A group G is a set plus an operation

$$\circ : G \times G \to G$$

satisfying

1. *Identity:* There is $e \in G$ with $e \circ a = a \circ e = a$.

2. *Inverse:* For any $a \in G$, there is $a^{-1} \in G$ with

$$a \circ a^{-1} = a^{-1} \circ a = e$$

3. *Associativity:* $a \circ (b \circ c) = (a \circ b) \circ c$

We additionally say G is Abelian if we have

$$a \circ b = b \circ a$$

Definition 7.3 (Group Order)

The order of G written $\#G$ is the number of elements in group G. If the order is finite, we say G is finite.

Example 7.4
$(\mathbb{Z}/p\mathbb{Z}, +)$ and $((\mathbb{Z}/p\mathbb{Z})^\times, \cdot)$ are both Abelian and finite.

§8 February 14, 2022

§8.1 Groups *continued*

Example 8.1
Some itemize of groups and nongroups:

- $(\mathbb{Z}/N\mathbb{Z}, +)$: Yes (Abelian).
- $(\mathbb{Z}/N\mathbb{Z}, \times)$: No. 0^{-1} does not exist (inverse).
- $((\mathbb{Z}/n\mathbb{Z})^\times, \times)$: Yes (Abelian).
- $(\mathbb{Z} \setminus \{0\}, \times)$: No. 2^{-1} does not exist (inverse).
- $(\mathbb{Z} \setminus \{0\}, +)$: No. No identity e.
- $(\{n \times n \text{ matrices} : \det M \neq 0\}, \times)$: Yes (not Abelian for $n \geq 2$).

Definition 8.2
For $g \in G$, $x = 1, 2, 3, \ldots,$

$$g^x = \underbrace{g \circ g \circ \cdots \circ g}_{x \text{ times}}$$

We extend this to define $g^0 = e$ and $g^{-n} = (g^n)^{-1}$ (so that our usual exponent rules also apply).

Example 8.3
From just now, in $(\mathbb{Z}/N\mathbb{Z}, +)$, $1^3 = 3$.

Definition 8.4 (Element Order)
The smallest (positive) n with $g^n = e$ is called the <u>order</u> of g.

If there is no such n, we say g has <u>infinite</u> order.

Proposition 8.5

If G is a finite group, then every element $g \in G$ has finite order.

Proof. Consider all powers of g

$$g, g^2, g^3, g^4, \ldots$$

so at some point, we will have

$$g, g^2, g^3, g^4, \ldots, g^i, \ldots, g^j, \ldots$$

where g^i and g^j are equal. Then $g^{j-i} = e$. Hence G has a finite order. □

Proposition 8.6

Let $g \in G$ have order k, with $g^n = e$. Then $k \mid n$.

Proof. We use the division algorithm. We write

$$n = q \cdot k + r \text{ with } 0 \leq r < k$$

then we have

$$e = g^n = (g^k)^q g^r = e^q \cdot g^r$$

so $g^r = e$, which forces $r = 0$ since $0 \leq r < k$. So $n = qk \Rightarrow k \mid n$. □

Theorem 8.7

$g^{\#G} = e$. In particular, $\operatorname{ord} g \mid \#G$.

Proof for Abelian groups. Let $G = \{g_1, \ldots, g_n\} = \{gg_1, gg_2, \ldots, gg_n\}$. No two are equal, since we can take inverse of g. We multiply them all together:

$$\begin{aligned} g_1 g_2 \cdots g_n &= (gg_1) \cdots (gg_n) \\ \cancel{g_1 g_2 \cdots g_n} &= \cancel{g_1 \cdots g_n} g^n \\ e &= g^n \end{aligned}$$

so we have as desired. □

This is true even if G is not Abelian - it's Lagrange's Theorem, which we won't cover here[8].

Note that our previous cryptosystems: Diffie-Hellman key exchange and Elgamal, works in *any* group.

[8] Covered in Math 1530, Abstract Algebra.

Q: Why would we want to be able to pick our group?

Might we want to do this in a group that allows for fast operations? That makes encryption and decryption easy, but it also makes computing the discrete log difficult. We want groups that are *easy enough* and *hard enough*. We might appreciate this by the end of the course...

§8.2 Computation Complexity

How might we quantify "easy" or "hard" in cryptography.

Example 8.8

Let $g \in G$ a group. Let's consider exponentiation

$$x \longmapsto g^x$$

if x has k bits (i.e. $x \approx 2^k$). How many steps does it take us to compute g^x? At most $2k$ multiply and add steps.

What about solving the discrete log problem:

$$g^x \longmapsto x$$

where x has k bits. How many steps does this take (naïvely, trying every power)? About 2^k steps.

Definition 8.9 (Big-O)

We say $f(x) = \mathcal{O}(g(x))$ if there are *constants* c and c' with

$$f(x) \leq c \cdot g(x) \quad \text{for all } x \geq c'$$

Example 8.10

Say $f(x) = \mathcal{O}(1) \Leftrightarrow f$ is bounded.

If $f(x) = \mathcal{O}(x^c)$, then we say this is a "easy" problem.[9]

If $f(x) = \mathcal{O}(c^x)$, we think of this as a "hard" problem.

[9] We take x to be the number of bits of the input

Proposition 8.11

If
$$\lim_{x\to\infty} \frac{f(x)}{g(x)} < \infty$$
then $f(x) = \mathcal{O}(g(x))$.

Proof. Using definition of limits, for any $\varepsilon > 0$:
$$\left| \frac{f(x)}{g(x)} - L \right| < \varepsilon \text{ for } x \geq c$$
then $f(x) < (L + \varepsilon) \cdot g(x)$. □

Example 8.12

$2x^2 + 5x + 7 = \mathcal{O}(x^2)$.