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 March 21, 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 and 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

$$24 + 5 \equiv 3 \mod 26$$
$$22 + 2 \equiv 1 \mod 12$$

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

$$a_1 \equiv a_2 \mod m$$
$$b_1 \equiv b_2 \mod m$$

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. □

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

$$a \cdot b_1 \equiv 1 \pmod m$$
$$a \cdot b_2 \equiv 1 \pmod m$$

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. □

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 $\gcd(a_1a_2, m) = 1$. We also have $a_1b_1 \equiv 1 \mod m$ and $a_2b_2 \equiv 1 \mod m$, we have $(a_1a_2)(b_2b_1) \equiv 1 \mod m$.
$a_1 + a_2$	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

```
1 def pow_mod(g, a, m):
2     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	1001
61	43	4	100
21	43	2	10
41	43	1	1
	$\boxed{63}$	0	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

$$1 = g = p \cdot u + a \cdot v$$
$$b = p \cdot ub + ab \cdot v$$

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

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 order

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

such that we have

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

(This makes sense since $\text{ord}_p(a)$ *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 $1 \leq i < p$.

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. □

§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 $0 \leq r < k$ ($\mathbb{Z}$ 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

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

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.

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

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

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

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

We want it to be easy to compute $e_{k_{\text{pub}}}$ and $d_{k_{\text{priv}}}$, but hard to compute $d_{k_{\text{priv}}}$ 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:

```
1 import ext_gcd, pow_mod
2 from random import randrange
3 def e(A, m):
4     k = randrange(p)
5     return (pow_mod(g, k, p), m * pow_mod(A, k, p))
6
7 def d(a, c):
8     pow_mod(c[0])
9     ...
```

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 order of g.

If there is no such n, we say g has infinite 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

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

so at some point, we will have

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

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 $G = \{g_1, \dots, g_n\} = \{gg_1, gg_2, \dots, gg_n\}$. 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

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

§9 February 18, 2022

§10 February 23, 2022

§10.1 Chinese Remainder Theorem

Recall: (HW2, Q2) asked us to find x with

$$\begin{aligned} x &\equiv 3 \mod 7 \\ x &\equiv 4 \mod 9 \end{aligned}$$

We solved this by setting

$$\begin{aligned} x = 7y + 3 &\equiv 4 \mod 9 \\ 7y &\equiv 1 \mod 9 \end{aligned}$$

and taking $7^{-1} \pmod 9$ which is 4. So we have

$$\begin{aligned} y &\equiv 4 \mod 9 \\ x = 7y + 3 &\equiv 31 \text{ works} \end{aligned}$$

Theorem 10.1 (Chinese Remainder Theorem)

Let $\{m_i\}$ be a set of pairwise coprime numbers. That is, $\gcd(m_i, m_j) = 1$ for $i = j$. Then the system

$$\begin{aligned} x &\equiv a_1 \mod m_1 \\ x &\equiv a_2 \mod m_2 \\ &\vdots \\ x &\equiv a_n \mod m_n \end{aligned}$$

has a solution.

Proof. By induction on n.

Base case. $n = 1$, then we take $x = a_1$ which is a solution.

Mini inductive step. We first solve for $n = 2$. We have

$$\begin{aligned} x &\equiv a_1 \mod m_1 \\ x &\equiv a_2 \mod m_2 \end{aligned}$$

Let $u_1 \cdot m_1 + u_2 \cdot m_2 = 1$ (by Bezout's identity). Consider quantity

$$\begin{aligned} u_1 m_1 a_2 + u_2 m_2 a_1 &\equiv 0 + 1 \cdot a_1 \equiv a_1 \mod m_1 \\ &\equiv 1 \cdot a_2 + 0 \equiv a_2 \mod m_2 \end{aligned}$$

which solves for $n = 2$.

Full inductive step. Let $n \geq 3$, we solve equation

$$\begin{aligned} x &\equiv a \mod m_1 m_2 \cdots m_{n-1} \\ x &\equiv a_n \mod m_n \end{aligned}$$

where the solution a from the first equation comes from our inductive hypothesis. We can solve this by our mini inductive step (the $n = 2$ case).

Which concludes this proof. □

Example 10.2

The Chinese Remainder Theorem allows us to solve congruences with composite moduli. For example, solve

$$x^2 \equiv 18 \mod 21$$

This is equivalent to solving

$$\begin{aligned} x^2 \equiv 18 = 0 \mod 3 \\ x^2 \equiv 18 = 4 \mod 7 \end{aligned}$$

since $21 = 3 \cdot 7$. So this is equivalent to solving

$$x \equiv 0 \mod 3$$
$$x \equiv \pm 2 \mod 7$$

Using CRT, we have

$$1 \cdot 7 + (-2) \cdot 3 = 1$$

so

$$x = 0 \cdot 1 \cdot 7 + (-2) \cdot 2 \cdot 2 = -12 \equiv 9 \mod 21$$

we do check that $9^2 = 81 \equiv 18 \mod 21$.

In general, we claim that square roots mod p are easy to compute.

Proposition 10.3

Let $p \equiv 3 \mod 4$ and a is a square in $\mathbb{Z}/p\mathbb{Z}$ (that is, $x^2 \equiv a$ has a solution).

Then

$$x \equiv a^{\frac{p+1}{4}}$$

is a solution.

Proof. Say $a \equiv b^2 \mod p$. Then

$$(a^{\frac{p+1}{4}})^2 = a^{\frac{p+1}{2}} \equiv (b^2)^{\frac{p+1}{2}} \equiv b^{p+1} \equiv b^{p-1} \cdot b^2 \equiv 1 \cdot b^2 = b^2 \equiv a \mod p$$

which concludes the proof. □

So we can compute square roots modulo a composite number N by taking the prime factor decomposition of N, and taking the square root mod each factor, then using CRT.

Conversely, any efficient algorithm to find square roots mod N can be used to factor N.

Why?

1. We generate an element x mod N.

2. Ask for square root of x^2.

3. Good chance that we get $y \not\equiv \pm x$. We now have

$$x^2 \equiv y^2 \mod N$$
$$(x+y)(x-y) \equiv 0 \mod N$$

4. We can now calculate $\gcd(x+y, N)$ and find some factors of N.

§10.2 Euler's Theorem

Recall: Fermat's Little Theorem which says that

$$a^{p-1} \equiv 1 \mod p \quad \text{for } p \nmid a$$

What happens when we replace p with N where N is composite?

$$a^{N-1} \stackrel{?}{\equiv} 1 \mod N \quad \text{for } \gcd(N, a) = 1$$

No! Recall demo showing counterexample.

Proposition 10.4

If $N = pq$ for primes p and q, then

$$a^{(p-1)(q-1)} \equiv 1 \mod N \quad \text{for } \gcd(N, a) = 1$$

Proof. WLOG taking mod p, we have

$$a^{(p-1)(q-1)} \equiv (a^{p-1})^{q-1} \equiv 1^{q-1} \equiv 1 \mod p$$

Similarly mod q by symmetry. Then it is congruent to 1 mod pq. □

We can generalize this...

Proposition 10.5 (Euler's Theorem)

For any composite N, we have

$$a^{\varphi(N)} \equiv 1 \mod N \quad \text{for } \gcd(N, a) = 1$$

§10.3 Exponentiation

Given an exponent x, we can compute e^x fast. Inverting this gives us the *Discrete Log Problem*. Diffie-Hellman Key Exchange and Elgamal rely on this.

What if we think of this as a function of the base? Given a base x, we want to take it to exponent e to get x^e. Inverting this is the *Extracting Roots* problem. This is the basis of the RSA cryptosystem (see next time!).

Claim — Let $\gcd(e, p-1) = 1$. We can construct $de \equiv 1 \mod p-1$. Then $(x^e)^d \equiv x$.

So *extracting roots* is easy mod prime p, but hard mod composites. We'll see this next time.

§11 February 25, 2022

Recall: Midterm discrete log problem where

$$x \to x^e \mod p$$

and a solution we gave was take $(x^e)^p \equiv x$ where $de \equiv 1 \mod (p-1)$.

What if we didn't take this mod p, but instead took it mod pq.

We can take the analog of Fermat's Little Theorem mod pq, where

$$a^{(p-1)(q-1)} \equiv 1 \mod pq$$

What if we did the same thing, instead of taking inverse mod $p-1$, we took it mod $(p-1)(q-1)$ to extract e^{th} roots if we know $(p-1)(q-1)$.

We know p and q, we can easily figure out $(p-1)(q-1)$. Also, if we know $(p-1)(q-1)$, we also know

$$pq - p - q + 1$$

We know that $pq = x$ and $p + q = y$, then pq are roots of $t^2 - yt + x = 0$.

§11.1 RSA Public-Key Cryptography

Alice generates

$$\begin{aligned} p, q &\to \text{Two large prime numbers} \\ e &\to \text{“Public exponent”} \\ N = pq &\to \text{“Public modulo”} \\ \boxed{(e, N)} &\to \text{Public key} \\ (d, N) &\to \text{Private key} \\ \text{where } d \cdot e \equiv 1 &\mod (p-1)(q-1) \end{aligned}$$

Bob has some message m he wishes to send to Alice. Bob sends $m^e \mod N$ and sends it to Alice.

After receiving this message, Alice can recover $(m^e)^d \equiv m \mod N$.

p, q are private, but pq is private. The security of RSA rests on multiplication being easy, but factorization being hard (pq is hard to factorize into p and q).

We might implement such an algorithm like so:

```
from crypto import gcd, ext_gcd, pow_mod
N = p * q
d = ext_gcd(e, (p-1) * (q-1))[0] % ((p-1) * (q-1))
m = 12345678912347869512340102398471237 48
# 0 < m < N is True
c = pow_mod(m, e, N)
pow_mod(c, d, N) # => m
```

If we were Alice and Bob, we *still* have one step to go! We need to find ourselves big prime numbers p, q (finding e is easy, we can just use our gcd algorithm). How do we do that?

§11.2 Primality Testing

Prime hunting!

Prime numbers are reasonably common. So generating a large prime number amounts to generating a number, checking if it's prime, and repeating until we get a prime.

This reduces to the problem of checking if a number is prime. Given an n, is it a prime?

```
from crypto import pow_mod
n = 12387461023948710289374189023 7023
pow_mod(2, n-1, n)
```

gives us a basic primality check. Fermat's Little Theorem says that if n is prime, then $2^{p-1} \equiv 1 \mod n$.

Definition 11.1 (Witnesses)

We say that a is a witness for the compositeness of n if

$$a^{n-1} \not\equiv 1 \mod n$$

and $gcd(a, n) = 1$.

We have a problem! Fermat's Little Theorem is not an if-and-only-if. We can have numbers that pass this test for almost every base. Take $n = 3 \cdot 11 \cdot 17 = 561$.

```
from crypto import pow_mod, gcd
for a in range(n):
    if gcd(a, n) == 1:
        print(pow_mod(a, n-1, n))
```

gives 1 for *everything*[10]

[10] oh no!

Definition 11.2 (Carmichael Number)

A Carmichael number is a composite number with *no* witness of compositeness.

Example 11.3

561 is a Carmichael Number.

We now *almost* have a way of checking for primality, but it doesn't always quite work.

Since taking to the power of $n-1$ is a group homomorphism, then Lagrange's Theorem states that if there is a witness, then there are *a lot* of witnesses.

Proposition 11.4

Let p be an odd prime. Write

$$p-1 = 2^k \cdot q \quad \text{with } q\text{-odd}$$

Then either

$$a^q \equiv 1 \mod p$$

or one of $a^q, a^{2q}, a^{4q}, \ldots, a^{2^{k-1}q}$ is $\equiv -1 \mod p$.

Proof. We look at this sequence

$$a^q, a^{2q}, a^{4q}, \ldots, \underbrace{a^{2^i q}}_{\not\equiv 1 \pmod p}, \underbrace{a^{2^{i+1}q}}_{\equiv 1 \pmod p}, \ldots, a^{2^{k-1}q}, a^{2^k q}$$

We have that $a^{2^k q} \equiv 1 \mod p$ by Fermat's Little Theorem. There's some point where we 'become' congruent to $1 \mod p$. If the first one is one, then we have the first case. Otherwise we have $a^q \not\equiv 1 \mod p$, then we repeatedly square until we get to $1 \mod p$. Then *right before* we turned to $1 \mod p$, we would have had $-1 \mod p$. In our example above, this is $a^{2^i q}$. □

§12 February 28, 2022

§12.1 Miller-Rabin Primality Test

Recall proposition 11.4 from last class.

Proposition

Let p be an odd prime. Write

$$p - 1 = 2^k \cdot q \quad \text{with } q\text{-odd}$$

Then either

$$a^q \equiv 1 \mod p$$

or one of $a^q, a^{2q}, a^{4q}, \ldots, a^{2^{k-1}q}$ is $\equiv -1 \mod p$.

For 561, we write $561 = 2^4 \cdot 35$.

$$\begin{aligned} 2^{35} &\equiv 263 \pmod{561} \\ 2^{70} &\equiv 166 \pmod{561} \\ 2^{140} &\equiv 67 \pmod{561} \\ 2^{280} &\equiv 1 \pmod{561} \end{aligned}$$

using the following code...

```
from crypto import pow_mod

pow_mod(2, 35) # 263
263 ** 2 % 561 # 166
166 ** 2 % 561 # 67
67 ** 2 % 561 # 1
```

Definition 12.1 (Miller-Rabin Witness)

a is a Miller-Rabin witness if a does not satisfy above proposition.

Theorem 12.2

If n is composite, then at at least 75% of $a \in (\mathbb{Z}/n\mathbb{Z})$ are Miller-Rabin witnesses.

Proof. Given on faith. □

Algorithm 12.3 (Miller-Rabin Probabilistic Primality Test) —

```
from random import randrange
from crypto import pow_mod

def miller_rabin(n, a):
    """
    Miller–Rabin primality test on number n and base a
    """
    q, k = n - 1, 0
```

```
    # Write n − 1 = 2^k · q
    while q % 2 == 0:
        k = k + 1
        q = q // 2
    a = pow_mod(a, q, n)
    if a == 1 or a == n − 1:
        return False
    for _ in range(k − 1):
        a = a ** 2 % n
        if a == n − 1:
            return False
    return True

def is_prime(n):
    for _ in range(50):
        if miller_rabin(n, randrange(1, n)):
            return False
    return True
```

Using this, we can find large prime numbers using

```
from crypto import miller_rabin, is_prime
def next_prime(n):
    while not is_prime(n):
        n = n + 1
    return n
```

Theorem 12.4 (Prime Number Theorem)

Probability that n is prime is about

$$\frac{1}{\log(n)}$$

More formally,

$$\lim_{x\to\infty} \frac{\#\text{ of primes} \le x}{x/\log x} = 1$$

So the Miller-Rabin test lets us efficiently find (large) prime numbers.

§13 March 7, 2022

§13.1 Pollard's $p-1$ Method

Recall: the Pollard's $p-1$ method from last time. Let

$$N = p \cdot q$$

$$\begin{matrix} p-1 \mid n! \\ q-1 \nmid n! \end{matrix} \to \gcd(N, a^{n!}-1) = p$$

```
from pollard import *

def factor(N):
    # Naive, brute force factoring algorithm.
    i = 2
    while N % i != 0:
        i += 1
    return i

def factor(N, a=2):
    # Pollard's method
    i = 1
    while gcd(a - 1, N) == 1:
        i += 1
        a = pow_mod(a, i, N)
    return gcd(a - 1, N)
```

This algorithm is *significantly* faster than random guesses. By how much? We can formalize this intuition of size of prime factors:

Definition 13.1

We say n is B-smooth if all prime factors are $\leq B$. We define

$$\Psi(X, B) = \#\text{ of } B\text{-smooth } n \leq X.$$

Theorem 13.2

Let X, B increase together. Suppose

$$(\log X)^{\varepsilon} \leq \log B \leq (\log X)^{1-\varepsilon}$$

Then

$$\frac{\Psi(X,B)}{X} = u^{-u\cdot(1+o(1))} \quad \text{where } u = \frac{\log X}{\log B}.$$

What is $o(1)$?

Definition 13.3

Say $f(x) = o(g(x))$ if

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$

So "$o(1)$" is to mean something whose limit is 0. This is in contrast to $O(1)$ which means something whose limit is finite.

For our purposes, we say that the probability X is B-smooth is $\approx u^{-u}$ where $u = \frac{\log X}{\log B}$.

What if we do Pollard $p-1$ for $e^{\sqrt{\log p}}$ steps? The probability of success is then

$$\sqrt{\log p}^{-\sqrt{\log p}} \approx e^{-\frac{1}{2}\log\log p \cdot \sqrt{\log p}} \gg p^{-\varepsilon}$$

By brute force, doing p^{ε} steps gives a probability of success $\approx p^{\varepsilon-1}$.

§13.2 Quadratic Sieve

Goal: find a, b with $a^2 \equiv b^2 \pmod{N = pq}$, hence $(a-b)(a+b) \equiv 0 \mod N$. $\gcd(a+b, N)$ will allow us to recover p or q with decent probability.

Example 13.4

For example, if

$$\begin{aligned}
(\lfloor\sqrt{N}\rfloor + 1)^2 &\equiv (\lfloor\sqrt{N}\rfloor + 1)^2 - N = 2^1 \cdot 3^1 \\
(\lfloor\sqrt{N}\rfloor + 2)^2 &\equiv (\lfloor\sqrt{N}\rfloor + 2)^2 - N = (\text{not smooth}) \\
(\lfloor\sqrt{N}\rfloor + 3)^2 &\equiv (\lfloor\sqrt{N}\rfloor + 3)^2 - N = 2^2 \cdot 3^1 \\
(\lfloor\sqrt{N}\rfloor + 4)^2 &\equiv (\lfloor\sqrt{N}\rfloor + 4)^2 - N = 2^1 \cdot 3^2 \\
&\vdots
\end{aligned}$$

then we have

$$\left[(\lfloor\sqrt{N}\rfloor + 1)^3\right]^2 \equiv \left[(\lfloor\sqrt{N}\rfloor + 3)(\lfloor\sqrt{N}\rfloor + 4)\right]^2$$

Which comes from the fact that the exponent vectors

$$\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}2\\1\end{bmatrix}, \begin{bmatrix}1\\2\end{bmatrix}$$

are linearly dependent. We can take these mod 2 (the parity) since we square. We can also rewrite this as[a]

$$\left[(\lfloor\sqrt{N}\rfloor + 1)(\lfloor\sqrt{N}\rfloor + 3)(\lfloor\sqrt{N}\rfloor + 4)\right]^2 \equiv \left(2^2 \cdot 3^2\right)^2$$

[a] Using the definition of linear dependence

So we can do the following steps:

1. Pick smoothness bound B.

2. Find integers
$$(\lfloor\sqrt{N}\rfloor + i)^2 - N$$
that are B smooth.

3. Find linear relationship between exponent vectors. Then we get a congruence $a^2 \equiv b^2 \pmod{N}$ after which we can try to find factors of N.

§14 March 9, 2022

***(missed?)

§15 March 11, 2022

§15.1 Quadratic Sieve *continued*

We resume our discussion of the example of example 13.4. We want to find an optimal B for the algorithm, and we do this by analyzing runtimes.

The runtime of X is first approximately $B \cdot u^u$ where
$$u = \frac{\log\sqrt{N}}{\log B}.$$

And in addition, solving a $B \cdot B$ system of linear equations. It takes B^2 operations to zero out one column (B rows and subtracting a column takes B operations), then we do this for B columns. So the runtime is B^3.

We now have u^u is decreasing in B and B^3 is increasing in B. Minimizing the runtime, we set $B^3 \sim B \cdot u^u$, i.e. $B^2 \sim u^u$. Using very sketchy mathematics,
$$\begin{aligned} u^u &\sim B \\ u\log u &\sim B \\ \frac{logN}{\log B} \sim u &\sim B \\ \log B &\sim \sqrt{\log N} \\ B &\sim e^{\sqrt{\log N}} \\ u \sim \frac{\log\sqrt{N}}{\log B} &\sim \frac{\log N}{\log B} \sim \sqrt{\log N} \end{aligned}$$
is a *loose* guess. But we can use this to make a more approximate guess.

Being more rigorous, $u^u = B^2$ gives $u \log u = 2 \log B$. Then using our estimate for u from above, we have

$$\begin{aligned} u \cdot \log\left[\sqrt{\log N}\right] &= 2 \log B \\ \frac{1}{2} \frac{\frac{1}{2} \log N}{\log B} \log\log N = \frac{1}{2} u \log\log N &= 2 \log B \\ (\log B)^2 &= \frac{1}{8} \log N \log\log N \\ \Rightarrow B &\sim e^{\sqrt{\frac{1}{8} \log N \log\log N}} \end{aligned}$$

where we note the difference of a factor of $\frac{1}{8} \log\log N$ isn't that far off from $e^{\sqrt{\log N}}$. So total runtime is around B^3 which is $e^{\sqrt{\frac{9}{8} \log N \log\log N}}$. *It's not super fast but not totally stupid.*

Realistically, the B^3 can be reduced to B^2 in solving our $B \times B$ system, but we can get rid of our factors of 2's from before. This lets us get rid of the factor of $\frac{9}{8}$ so our total runtime is like

$$e^{\sqrt{\log N \log\log N}}$$

There are even faster algorithms, namely the number field sieve. It replaces the square root from above into a cube root and has runtime

$$e^{\sqrt[3]{c \cdot \log N (\log\log N)^2}}$$

§15.2 Index Calculus & Discrete Logs

The discrete log problem is solving x given $g^x = a$ and we know g and a. We can do a similar strategy to the quadratic sieve. We calculate

$$\begin{aligned} g^1 &\pmod p = 2 \\ g^2 &\pmod p = (\text{not smooth}) \\ g^3 &\pmod p = 2 \cdot 3 \end{aligned}$$

where we pick some smoothness bound B. We then find $g^i \equiv (\text{B-smooth}) \mod p$. We find enough B smooth things such that by linear algebra, we can solve $g^x = 2, 3, 5, 7, \ldots$.

We take

$$\begin{aligned} a &\mod p = (\text{not smooth}) \\ a \cdot g^{-1} &\mod p = (\text{not smooth}) \\ a \cdot g^{-2} &\mod p = 3 \end{aligned}$$

which is smooth. So we find $a \cdot g^{-i} \equiv (\text{B-smooth}) \mod p$. Then by linear algebra we can solve for a given lots of smooth g^i.

Question. What is the runtime of this?

We need to go up to g^x where x is about $B \cdot u^u$ (u^u is the probability of being B-smooth). Linear algebra will take runtime B^2. Similar to just now, this is exactly the same problem as above except $\sqrt{N}$ is replaced with P. Before (in the quadratic sieve), we had

$$B \sim \exp\left(\frac{1}{4}\log N \log\log N\right)$$

and

$$\text{Runtime} \sim \exp\left(\log N \log\log N\right)$$

Replacing $\log N$ with $2\log P$, our bound becomes

$$B \sim \exp\left(\frac{1}{2}\log p \log\log p\right)$$

$$\text{Runtime} \sim \exp\left(2\log p \log\log p\right)$$

What if $g \in$ subgroup of order q? Babystep-Giantstep changes from $\sqrt{p}$ to $\sqrt{q}$. With index calculus, there is no advantage of knowing that g is im a smaller subgroup. We don't have a decrease in runtime.

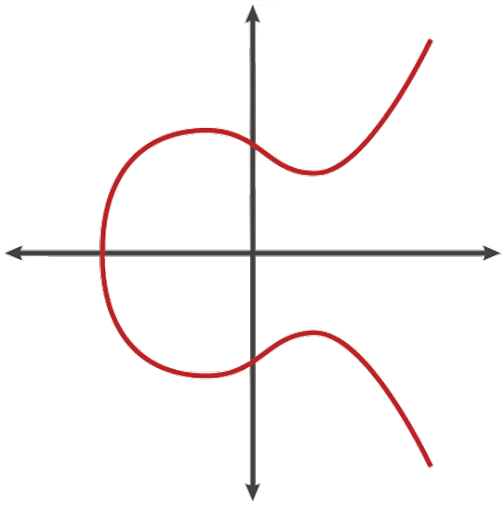
§16 March 14, 2022

§16.1 Elliptic Curves

An elliptic curve is an equation of the form

$$y^2 = x^3 + ax + b$$

It looks like this:



What's special about cubic equations? We have a special property that every line L meets E in 3 points.

What happens when we have a tangential line? We count multiplicity.

What happens when it only meets at 1 or 2 points? We count complex roots.

What happens with vertical lines that only meet at 2 points? We include $\mathcal{O} =$ "point at ∞".

Where does $\mathcal{O}$ come from? It comes from the $\mathbb{RP}^2$ (the real projective plane) or $\mathbb{CP}^2$ (the complex projective plane).

Given two points A and B, we can get a third point C which is the third point on the line passing through A and B. Taking this as a binary operation...does this give us a group?

Consider:

$$\begin{aligned} A + B &= C \\ A + C &= B \\ A + A &= \mathcal{O}? \end{aligned}$$

Maybe we can declare

$$A + B + C = \mathcal{O}$$

If we call the reflection of C across the x-axis is D, we have

$$\begin{aligned} A + B + C &= \mathcal{O} \\ C + D + \mathcal{O} &= \mathcal{O} \\ A + B &= D \end{aligned}$$

So we have that the group law is $A + B$ is the reflection of the third point C across the x-axis.

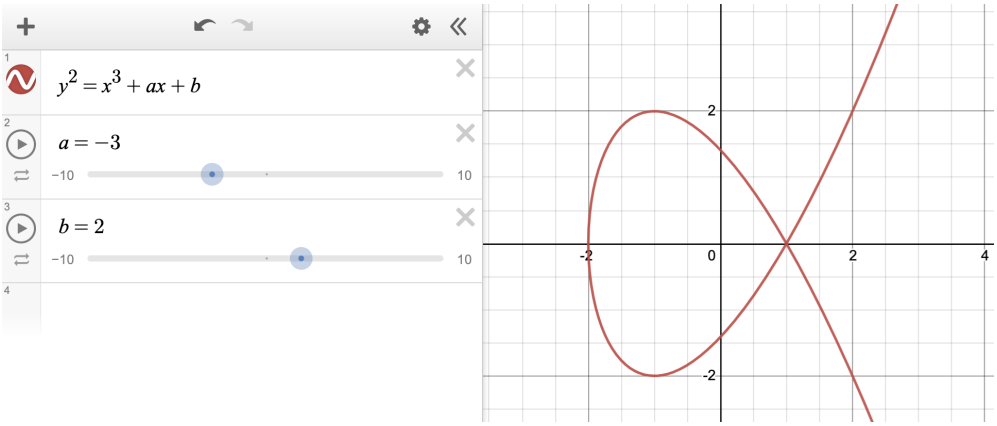
Definition 16.1 (Elliptic Curve)

An elliptic curve is the set of solutions to

$$y^2 = x^3 + ax + b$$

plus a point $\mathcal{O}$ at infinity...where a, b satisfy $4a^3 + 27b^2 \neq 0$.

Recall from high school that $ax^2 + bx + c$ gives discriminant $\Delta = b^2 - 4ac$. Taking a cubic equation $x^3 + ax + b$, the discriminant is $\Delta = -16(4a^3 + 27b^2)$. This is also saying $x^3 + ax + b$ has no repeated roots. For it to be tangent to the x-axis, it has to self-intersect. But every line passing through the intersection is a tangent line. Messy messy things happen:



We take the fact that an elliptic curve is a group on faith, with the group operation defined above.

§16.2 Addition on Elliptic Curves

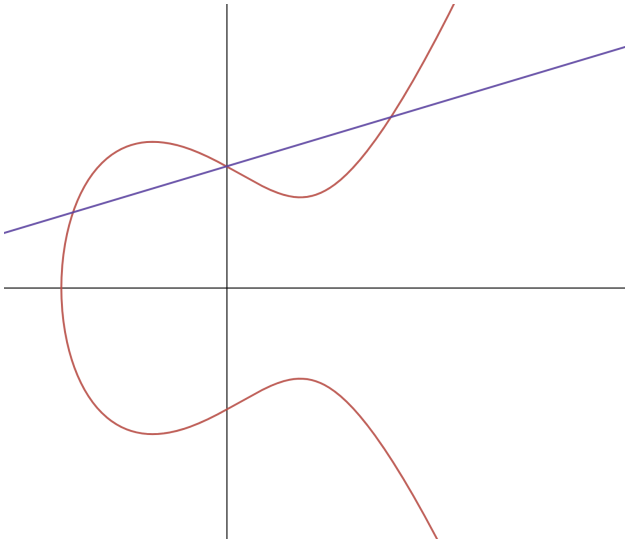
- $P + \mathcal{O} = \mathcal{O} + P = P$. That is, $\mathcal{O}$ is the identity.
- If for points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ and $x_1 = x_2$, $y_1 = -y_2$. Then $P_1 + P_2 = \mathcal{O}$.
- If $P_1 \neq P_2$, $\lambda =$ Slope of $L = \frac{y_2 - y_1}{x_2 - x_1}$. So the equation of L is $y - y_1 = \lambda(x - x_1)$. (We find the third point and reflect it). Will pick up here on Wednesday.

§17 March 16, 2022

§17.1 Addition on Elliptic Curves *continued*

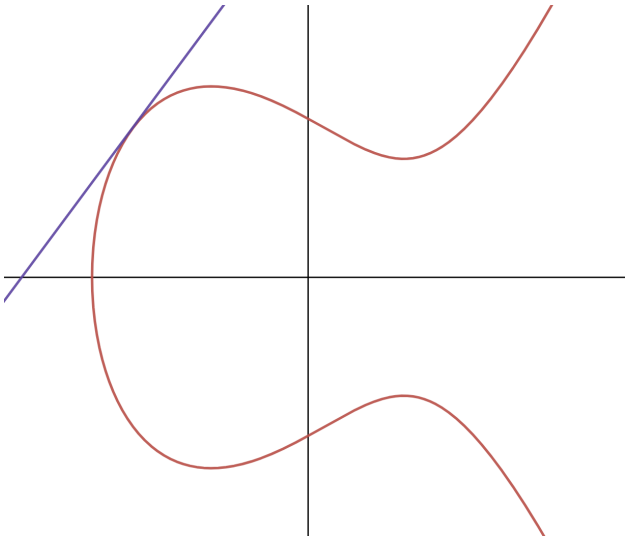
We said last time that we had some rules:

- $P + \mathcal{O} = \mathcal{O} + P = P$.
- If $x_1 = x_2$ and $y_1 = -y_2$, we have
$$P_1 + P_2 = \mathcal{O}$$
- In other cases, we need to calculate the slope of the line.



If $P_1 \neq P_2$, the slope of L is

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$



If $P_1 = P_2$, then the slope would be the tangent:

$$\begin{aligned} y^2 &= x^3 + ax + b \\ 2y \cdot dy &= (3x^2 + a) \cdot dx \\ \frac{dy}{dx} &= \frac{3x^2 + a}{2y} \end{aligned}$$

so we have

$$\lambda = \frac{3x^2 + a}{2y}$$

We want to solve systems

$$\begin{cases} y^2 = x^3 + ax + b \\ y = y_1 + \lambda(x - x_1) \end{cases}$$

which gives us

$$\begin{aligned} [y_1 + \lambda(x - x_1)]^2 &= x^3 + ax + b \\ 0 &= x^3 - \boxed{\lambda^2} x^2 + (a + 2\lambda_1 x_1)x + b - y_1 - \lambda x_1^2 \\ &= (x - x_1)(x - x_2)(x - x_3) \\ &= x^3 - \boxed{(x_1 + x_2 + x_3)} x^2 + (x_1x_3 + x_2x_3 + x_1x_2)x - x_1x_2x_3 \end{aligned}$$

With some working out (taking the coefficient of x^2),

$$\boxed{x_3 = \lambda^2 - x_1 - x_2}$$

and $-y_3 = y_1 + \lambda(x_3 - x_1)$ so we have

$$\boxed{y_3 = -y_1 - \lambda(x_3 - x_1)}$$

are our points by addition where

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

or when $P_1 = P_2$,

$$\lambda = \frac{3x_1^2 + a}{2y_1}.$$

§17.2 Elliptic Curves over Finite Fields

Our definition stays the same, except x, y are elements of $\mathbb{Z}/p\mathbb{Z}$. How do we add points? We could do it geometrically, but setting this up is outside the scope of this class...

For addition, we use the same formulas that we've derived for x_3 and y_3, and they still make perfect sense mod p. *For whatever reasonable notion of geometry we have over $\mathbb{Z}/p\mathbb{Z}$, they work with these formulas.*

Example 17.1

We take elliptic curve

$$y^2 = x^3 + x + 2 \quad \text{over } \mathbb{Z}/5\mathbb{Z}.$$

How do we find elements in this elliptic curve? We can try them all.

- If $x = 0$, $y^2 = 2$, of which there are no solutions.
- If $x = 1$, $y^2 = 4$, of which $y = 2, 3$ are solutions.
- If $x = 2$, $y^2 = 2$ again, of which there are no solutions.

- If $x = 3$, $y^2 = 2$, of which there are no solutions.
- If $x = 4$, $y^2 = 0$, so $y = 0$ is one solution.

We have $(1,2), (1,3), (4,0), \mathcal{O}$ are the elements of this elliptic curve. We have these

$+$	$\mathcal{O}$	$(1,2)$	$(4,0)$	$(1,3)$
$\mathcal{O}$	$\mathcal{O}$	$(1,2)$	$(4,0)$	$(1,3)$
$(1,2)$	$(1,2)$	$(4,0)$	$(1,3)$	$\mathcal{O}$
$(4,0)$	$(4,0)$	$(1,3)$	$\mathcal{O}$	$(1,2)$
$(1,3)$	$(1,3)$	$\mathcal{O}$	$(1,2)$	$(4,0)$

Let's implement this:

```
O = "the point O"
def add(P1, P2, a, p):
    if P1 == O:
        return P2
    if P2 == O:
        return P1
    x1, y1 = P1
    x2, y2 = P2
    if x1 == x2 and (y1 + y2) % p == 0:
        return O
    if P1 == P2:
        lam = (3 + x1**2 + a) * ext_gcd(2 * y1, p)[0] % p
    else:
        lam = (y2 - y1) * ext_gcd(x2 - x1, p)[0] % p
    x3 = (lam**2 - x1 - x2) % p
    y3 = -y1 - lam * (x2 - x1)
    return x3, y3
```

§18 March 18, 2022

§18.1 Elliptic Curves over Finite Fields *continued*

Recall: from last class, we had our toy example

Example

We take elliptic curve

$$y^2 = x^3 + x + 2 \quad \text{over } \mathbb{Z}/5\mathbb{Z}.$$

with group law

$+$	$\mathcal{O}$	$(1,2)$	$(4,0)$	$(1,3)$
$\mathcal{O}$	$\mathcal{O}$	$(1,2)$	$(4,0)$	$(1,3)$
$(1,2)$	$(1,2)$	$(4,0)$	$(1,3)$	$\mathcal{O}$
$(4,0)$	$(4,0)$	$(1,3)$	$\mathcal{O}$	$(1,2)$
$(1,3)$	$(1,3)$	$\mathcal{O}$	$(1,2)$	$(4,0)$

We define some more useful functions:

```
def minus(P, p):
    if P == O:
        return O
    else:
        x, y = P
        return (x, (−y) % p)
```

What about multiplication? We can repeatedly add:

```
def multiply(P, n, a, p):
    S = O
    for _ in range(n):
        S = add(P, S, a, p)
    return S
```

but this might be very slow (it does n iterations). We can do something similar to fast powering for exponentiation, except we repeatedly double our point:

```
def multiply(P, n, a, p):
    S = 0
    while n != 0:
        if n % 2 == 1:
            S = add(S, P, a, p)
        n = n // 2
        P = add(P, P, a, p)
    return S
```

Question. What is the order of $E(\mathbb{F}_p)$? That is, how many points are there on the elliptic curve?

Let's say we take $x, y, \ldots$. We want to solve

$$y^2 \stackrel{?}{=} x^3 + ax + b$$

There are p^2 different (x, y). The probability that this equality holds is *like* $\frac{1}{p}$. So there are about $p^2 \cdot \frac{1}{p} + 1 \approx p + 1$ (added one for the point $\mathcal{O}$) elements in $E(\mathbb{F}_p)$.

Theorem 18.1

$$|(p+1) - \#E(\mathbb{F}_p)| \leq 2\sqrt{p}.$$

That is, the difference between our estimate $p+1$ and the actual number of points in $E(\mathbb{F}_p)$ is bounded by $2\sqrt{p}$.

Proof. (Beyond the scope of this class.) □

Remark 18.2.

1. We note that this number can be efficiently computed (in polynomial time in the digits of p). *(Again, beyond the scope of this class.)*
2. We call this number $|(p+1) - \#E(\mathbb{F}_p)|$ the trace of Frobenius. *(You guessed it: again, beyond the scope of this class.)*

Just for funsies, we can compute the *trace of Frobenius*[11]:

```
p = next_prime(92834712736591432)
E = EllipticCurve([34123498, GF(p)(2349182347)])
E.trace_of_frobenius()
```

§18.2 Elliptic Diffe-Hellman Key Exchange

§18.2.1 Elliptic Discrete Log Problem

This is based on the *Elliptic Discrete Log Problem*: Given E an elliptic curve over $\mathbb{F}_p$ with P a point on E. We take P, n and calculate

$$n \cdot P$$

The elliptic curve discrete log problem is given point $n \cdot P$, computing n.

The best known algorithm for *ECDLP* is Babystep-Giantstep, which runs in $\mathcal{O}(\sqrt{p})$ and $\mathcal{O}(\sqrt{p})$ memory. There is *no* analog of index calculus. This could be good or bad... This could mean a lack of knowledge about elliptic curves. We could also create greater security at smaller key sizes.

§18.2.2 Sharing Secrets

Public information: we have some p prime and E an elliptic curve over $\mathbb{F}_p$. We have P a point in the elliptic curve E.

[11] Using *Sage*.

Alice and Bob do the following:

1. Alice and Bob each generate a and b. Alice computes $a \cdot P$ and Bob computes $b \cdot P$. This is shared to each other (and public).

2. Alice now computes $a \cdot (b \cdot P)$ and Bob computes $b \cdot (a \cdot P)$. These are all equal to $(a \cdot b) \cdot P$ which is a shared secret.

§19 March 21, 2022

§19.1 Elliptic Curve Elgamal

As usual, we have some public knowledge, private key, and public key. The idea is to replace multiplication in $\mathbb{F}_p^\times$ with addition on E.

Public Knowledge:
p — prime.
E — elliptic curve over $\mathbb{F}_p$.
$P \in E(\mathbb{F}_p)$ — point.

Private Key:
n — private key.

Public Key:
$Q = n \cdot P$ — public key.

Encrpytion:
Bob has a message $M \in E(\mathbb{F}_p)$.

1. Choose random k.

2. Compute

$$C_1 = k \cdot P$$
$$C_2 = M + k \cdot Q$$

Send (C_1, C_2) to Alice.

Decryption:
Alice will compute

$$C_2 - n \cdot C_1 = M + k \cdot Q - nk \cdot P = M$$

We now implement this:

```sage
from ec import ext_gcd, add, minus, multiply

# Private key for Alice
n = randrange(q)

# Public key for Alice
Q = multiply(P, n, a, p)

def e(Q, M):
    """Encryption Function"""
    k = randrange(q)
    c1 = multiply(P, k, a, p)
    c2 = add(M, multiply(Q, k, a, p), a, p)
    return (c1, c2)

def d(n, C):
    """Decryption Function"""
    c1, c2 = C
    return add(c2, minus(multiply(c1, n, a, p), p), a, p)
```

The expansion factor is 2. Even if we think of putting our message into only the x coordinate, the y coordinate is determined by the x coordinate so the factor is still 2.

§19.2 Elliptic Curve DSA

Since we have Elgamal, we can also have DSA with Elliptic Curves.

Public Knowledge & Public Key:
Same as Elgamal
p — prime.
E — elliptic curve over $\mathbb{F}_p$.
$Q = n \cdot P$ — public key.

Private Key:
n — private key.

Signing:
Alice has document $d \in \mathbb{Z}/q\mathbb{Z}$.

1. Choose random k.

2. Compute $(x, y) = k \cdot P$.

$$s_1 = x$$
$$s_2 = (d + ns_1) \cdot k^{-1} \pmod{q}$$

Verification:
We can verify the signature as follows:

$$v_1 = d \cdot s_2^{-1} \pmod{q}$$
$$v_2 = s_1 s_2^{-1} \pmod{q}$$

$$\begin{aligned} v_1 \cdot P + v_2 \cdot Q &= ds_2^{-1} \cdot P + s_1 s_2^{-1} k \cdot P \\ &= (d + s_1 n) s_2^{-1} P \\ &= kP \end{aligned}$$

(x-coord of $v_1 P + v_2 Q) = s_1$ check to verify signature.

Again, we can implement this:

```
from ec import *

def sign(n, d):
    """Signing document d with private key n"""
    k = randrange(q)
    x, _ = multiply(P, k, a, p)
    s1 = x
    s2 = ((d + n * s1) * ext_gcd(k, q)[0]) % q
    return (s1, s2)

def verify(Q, d, s):
    """Verifies document d's signature s with public key Q"""
    s1, s2 = s
    v1 = (d * ext_gcd(s2, q)[0]) % q
    v2 = (s1 * ext_gcd(s2, q)[0]) % q
    return add(multiply(P, v1, a, p), multiply(Q, v2, a, p), a, p)[0] == s1
```

Remark 19.1. The specific numbers used in lecture is the elliptic curve used in the Bitcoin blockchain. Being able to forge signatures in this elliptic curve is to topple the Bitcoin market.