Can you keep a secret?

Intro

Do you have secrets? crucial information that only a select few should be aware of? if so then this lab is for you. We saw that hashes can help us turn some information - for example a password - into a different form, but this transformation is a “one way street”: it’s very hard, if not impossible, to figure out the original information from the hash.

Encryption algorithms will also transform data, but with a clear way to get the original data back. In other words: If we have an original message *m,* we can create an encrypted message *c* which can be deciphered back to *m* with another algorithm called a decryption algorithm. There are many techniques for encryption, some of which date thousands of years. This lab will apply the content we recently learned to discuss and analyse encryption strategies, and build up to an understanding of RSA, one of the most commonly used encryption algorithms today.

Instructions

The project has 3 parts:

- Part 1: Defining Encryption
- Part 2: Preparing for RSA
- Part 3: Proving RSA

Each part will involve applying a different set of mathematical concepts you've learned.

Due Dates

This is a multi-week project.

You have until Dec 18 to finish all of the required parts of the project.

You are expected to submit your work by these deadlines:

- Dec 5: Part 1
- Dec 12: Part 2
- Dec 18: Finished Project (all three parts)

Your answers will only be scored after the finished project deadline. You are encouraged to revisit your answers to Part 1 and Part 2 if you think you can improve them.

You have access to the entire project now, but we have not have covered all the topics yet. Plan to revisit the project each week as you learn the skills required to solve each part of the challenge.

Submission

There will be a gradescope link to submit your work. You'll be able to revise and resubmit those parts for the final submission deadline. The final submission is what will be graded.

For each of the questions, create a pdf or scan an image to show your work. It may be easiest to write your solution by hand, then upload a photo of that instead of typesetting your work in a text editor; feel free to do either.

Notes

- For questions that require you to perform calculations, make sure to show your work.

- For questions that ask you for a hypothesis or opinion, we do not expect a correct answer. Feel free to share your genuine thoughts, and we will revisit them later through the exercise.

Part 1: Intuition for encryption

Encryption fundamentally differs from hashing in that we want to ultimately be able to decrypt our initial message. Let's formalize this. We define a function *Encrypt* with domain *D* and range *R*. for some message *m* in *D*, *Encrypt(m)* will be referred to as the cyphertext We also must define a function *Decrypt*, the inverse of *Encrypt*, so that *Decrypt(Encrypt(m)) = m*

Let's look at a practical, historical example:

Letter	a	b	c	d	e	f	g	h	i	j	k
Encrypt(Letter)	s	n	v	f	r	g	h	j	o	k	l

The above table is an example of a 'substituion cypher'. Using the table above, we encrypt each character in our provided message.

1. What is the value of *Encrypt(GOODLUCK)*? what is the value of *Decrypt(ASDYASN)*?

2. More formally, how would you define the domain and range of the Encrypt and Decrypt functions?

This strategy has been used for centuries - perhaps you've used it before yourself? This is known as a symmetric key strategy: both parties communicating need to know this same table in order to encrypt and decrypt messages correctly.

This strategy served as the basis for developing more complex encryption algorithm, which we will explore in the next sections:

Moving from characters to numbers:

We know at this point that arbitrary data on our computers is represented in terms of bits. Everything is a number! We want to be able to encrypt the binary representation of data, as that would equally apply to text as it would audio or images. We can use a look up table as our

'key', mapping numbers to others, but there are many alternative ways to approach this situation:

3. Let's define B_4 as the set of all 4 bit binary numbers. For each of the following relations, justify if they are a valid encryption function, and if so define their corresponding decryption function.
    1. R_1(x, key), where *x* is in B_4 and *key* is a positive integer. We rotate *x* to the left n times to produce our cypher text. For example R_1(1011, 1) = 0111 and R_1(1011, 2) = 1110

    2. R_2(x, key) = x + key, where key is also an element of B_4

    3. R_3(x, key) = x XOR key, where key is also an element of B_4

Dealing with arbitrarily large data:

I trust you've justified this on your own, but R_3 looks promising! say our plan is to XOR our binary input with the secret key k=1010. How can we handle a 12 bit message m instead of a 4 bit one? simply computing m XOR k would only encrypt the four right most bits. Let's think about this situation:

4. Given a 12 bit number, what mathematical operation would allow you to identify the 4 leftmost bits? Based on this operation, what would be your strategy to encrypt a 12 bit number using a 4 bit key?

5. Formalize and proove that your strategy above is correct.

6. How would you change your approach if the input was an 11 bit number instead of a 12 bit one?

We do not trust the messenger:

With this kind of strategy, where a single piece of information - our key - is crucial, we quickly run into a major problem. Imagine that Alice wants to send a secret message to Bob, and that Cynthia is eavesdropping on their communication.

- Alicee computes *Encrypt(message, key)* and sends it to Bob.

- Cynthia intercepts the message but can not decrypt it, as she does not have access to the key!

- Bob receives the message as well...but also can not decrypt it, as he does not have the key.

Unfortunately, we do not have a guaranteed way to share the key over an untrusted network. The moment anyone knows the key, then our encryption serves no purposes. In the next part of this lab we will explore strategies to address this challenge, but before moving to that section:

7. Can you think of a strategy that would allow communication over an untrusted network? this is an open ended question where we expect you to think about the problem and share ideas or questions of your own, no pressure to create a brand new algorithm!

Part 2: Building up to RSA

We concluded the previous checkpoint with the observation that symmetric key cryptography has a major challenge when it comes to sharing the key itself. In Part 3, we will dive deep into

the RSA algorithm, a popular strategy to avoid this problem, but we need to get on the same page on a few definitions and theorems before tackling RSA head on.

Asymmetric key encryption:

- One key idea behind RSA, and other similar algorithms, is that the key used to encrypt the message is *different* from the key used to decrypt it.

- RSA relies on the idea that each party in communication has two different keys: A public key *e* and a private key *d*

- Let's say that pub(m) represents applying the public key on some message m, and priv(m) represents applying the private key. We want to define *e* and *d* such that:
  - priv(pub(m)) = m

  - pub(priv(m)) = m

- If Alice wants to receive messages from Bob, she would make her public key publicly known

- Bob can then send pub_alice(m) to Alice.

- Alice can then decrypt the cypher text by applying her private key, so priv_alice(pub_alice(m))=m.

- If Cynthia is still eavesdropping, then she knows the public key of Alice, she can also intercept pub_alice(m), but because she does not know the private key, she should not be able to decypher the message.

1. How do you feel about this set up? Do you have any questions in mind about e and d, the public and private keys?

RSA relies on some interesting properties of numbers to create these keys and apply them to data. We will now go over some key concepts before diving into the details of RSA.

Relative primes:

We have covered in class what a prime number is: a number that is divisible only by itself and 1.

We say that two numbers are considered relatively prime if their greates common divider is 1. By definition, it then follows that a prime number p is relatively prime with any other number that is not a power of the prime.

2. Show that two non-prime numbers a and b can also be relatively prime.

3. Find a relatively prime number for the following numbers:
    1. 1715

    2. 100

3. 482671

4. Using the definition, test if the following pairs of numbers are relatively prime:
    1. 215 and 216

    2. 17 and 68

    3. 16 and 81

Extended Euclidian algorithm.

You may have used the euclidian algorithm to solve some of the problems above. We will slightly tweak it now to get a bit more information out of it. Before we introduce the extended euclidian algorithm, let's first think through what other information we really want.

5. Prove that if gcd(a, b) = d, then $\exists x, y \in \mathbb{Z}$ such that d = ax + by

The extended euclidian algorithm allows us to identify exactly what this x and y are. It returns 3 different values: the gcd, a factor x, and a factor y, such that gcd(a, b) = ax + by

def extended-euclidian(a, b): if a == 0: return b, 0, 1

```
gcd, x1, y1 = extended-euclidian( b mod a, a)

x = y1 - (b/a) * x1 # we use integer division here
y = x1

return gcd, x, y
```

Proving the correctness of this algorithm is left as an optional exercise. However, we should convince ourselves that it does work.

6. For the following three pairs of integers a and b, apply the extended-euclidian algorithm to identify x, y such that gcd(a, b) = ax + by:
    1. 17 and 68

    2. 16 and 81

    3. 215 and 321

With this knowledge in hand, we are now ready to tackle RSA in the next section of the lab!

Part 3: Proving RSA

Let's jump straight ahead into the RSA algorithm:

- We begin by picking two prime numbers, *p* and *q*

- We compute ***n*** = *pq*

- We compute \phi (n) = *(p-1)(q-1)*

- We compute ***e*** to be a small odd integer which is relatively prime to \phi (n)

- Finally, we compute ***d*** to be the *multiplicative inverse of* ***e***, modulo \phi (n)
  - This last step takes some clarification. We can phrase that step as solving the equation de \equiv 1 \pmod \phi (n)

At this stage, we consider the pair (e, n) to be our public key, and (d, n) our private keys.

1. Manually compute the private RSA keys for the following inputs:

2. p = 17, q = 19, e = 11

3. p = 17, q = 19, e = 5

4. Pick two values for p and q of your chosing, so long as they are primes less than 100. Create your own public and private keys.

5. Generally, write e and d in terms of p and q. Here is a hint: what does *ed* equal to?

We still need to understand the encryption and decryption functions necessary. To encrypt a message m, considering m is an integer, we get cypher text *c* such that: c = P(m) = \m^e \pmod n

In order to obtain the original message *m* from the cypher text *c*, we compute: m = S(c) = \c^d \pmod n

First, let's see this in action: 4. Say our secret message *m* is the number 65. Compute the cypher text, then decrypt it, using the following values: n = 299, e = 5, d = 53