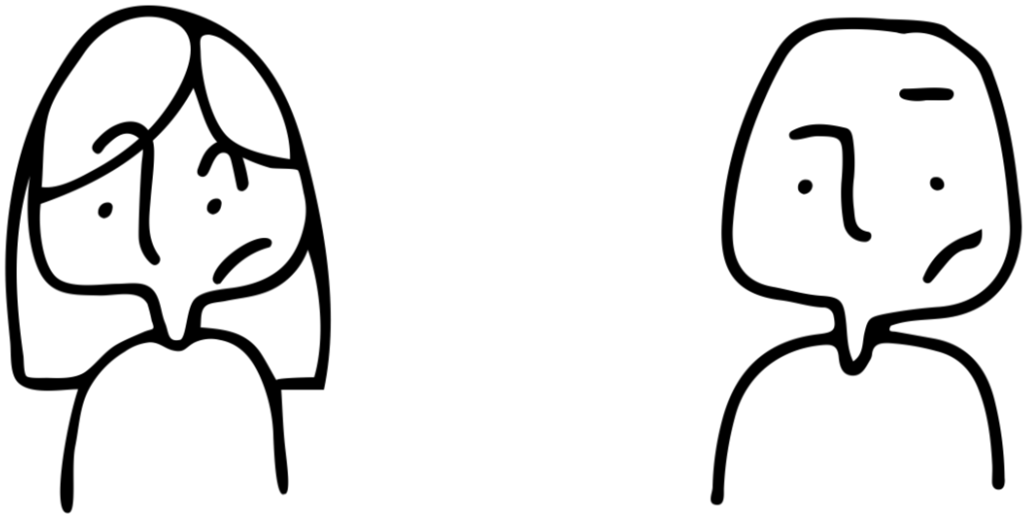
Bitcoin Cryptography: Simply Explained



Mateusz Faltyn

Blockware Solutions

Acknowledgements

I want to thank Mason Jappa, Sam Chwarzynski, Tanner Davis, and the Blockware Solutions Team for their continued support and valuable feedback on drafts of this guide.

I want to thank *ester barbato* from Noun Project for creating the graphics used in this text.

Disclaimer

Contact

Mateusz (Matthew) Faltyn
Blockchain Developer
Blockware Solutions
Twitter: @FaltynMateusz

Table of Contents

Forward

Math is like music.

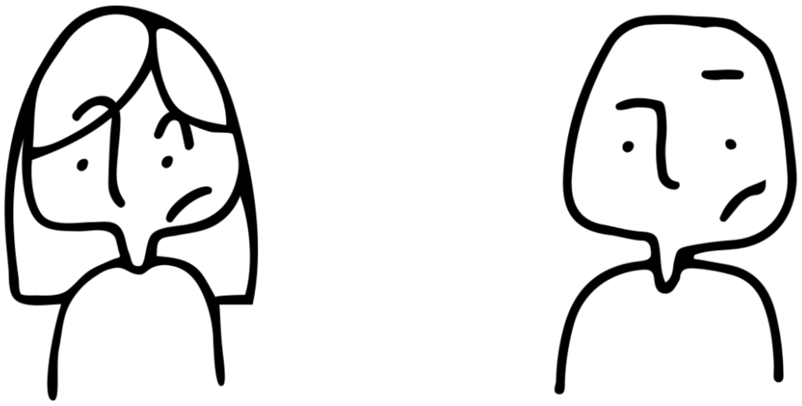
Very few of us dummies have the experience and skill necessary to interpret the sound, emotion, or impact of a song from sheet music.

For most of us, music must be played! To hear and feel the song is to understand the piece and its significance.

Math can be very difficult for many of us dummies to interpret if it is presented in standard symbolic notation. Thus, math must be “played” via illustration for a vast majority of individuals to “hear” the mathematics.

In *Bitcoin Cryptography: Simply Explained*, we will attempt to “play” the mathematics whenever possible in order to help us dummies better understand the formal mathematical and technical details.

Chapter 1: Introduction to Cryptography



Secrets and Privacy

Secrets

Have you ever kept a secret hidden from your family or friends?

Of course you have! You are a dummy, so you probably forgot that you did.

Indeed, the framing of such a question may sound anywhere between silly to conspiratorial or even downright sinister; however, secrets are not inherently negative.

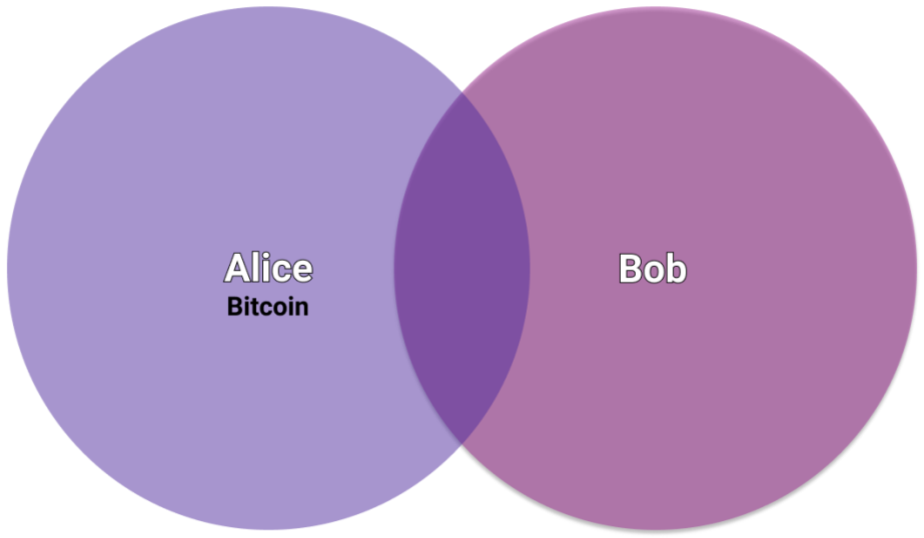
Let us provide an operational definition of *secret* that will help inform our later conversation involving cryptography:

Secret: A piece of information that is known to one party and is unknown to another.

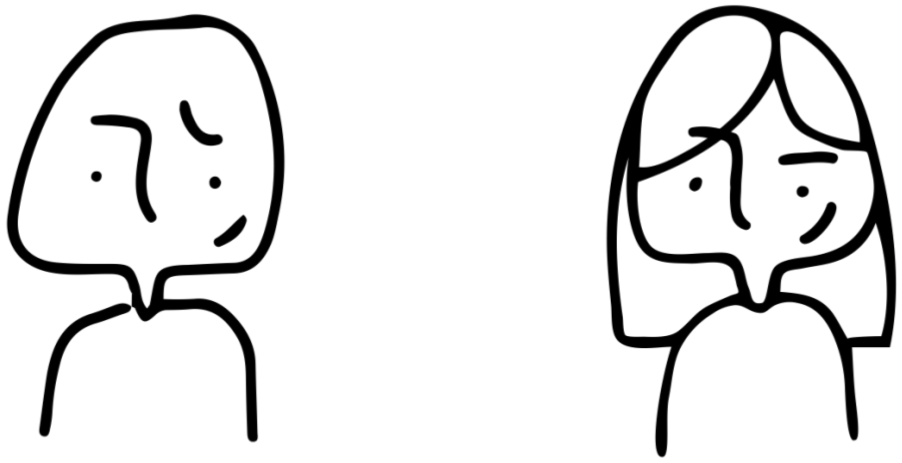
Using this definition, we see that all information that is asymmetrically shared is essentially a composition of secrets.

Suppose Alice and Bob are best friends.

Alice knows that Bitcoin exists while Bob does not. In relation to Bob, Alice knows a secret that is shared amongst all other individuals who are aware of Bitcoin's existence. The knowledge of Bitcoin's existence is asymmetric in Alice and Bob's relationship.



Secrets shift our worldview. Take Alice and Bob once more.



How different are Alice’s and Bob’s worldviews? However far you have fallen down the Bitcoin rabbit hole, you are able to empathize with Bob: at one point in time, Bitcoin was a *secret* between a group of individuals – and you were not in that group. Alice’s worldview is fundamentally different than Bob’s because of all the secrets she holds that he does not and vice versa.

What are the values of secrets?

This question motivates the remainder of this section where we explore privacy as well as the consequences of a lack of privacy.

Privacy

We can adjust the International Association of Privacy Professionals' definition of *privacy* to better fit our definition of *secret*:

Privacy: The right to hold a secret with freedom from intrusion or interference.

In essence, privacy is a promise that you are able to keep secrets without punishment. Privacy entails that you do not have to notify everyone around you of what you are thinking and what you know at all times. It provides us the ability to select which thoughts and feelings we want to express and to whom we want to express them.

Now that we understand the general concept of privacy, let us delve deeper into more formal definitions of privacy rights. William Prosser defined four privacy rights in his 1955 book entitled *Handbook of the Law of Torts*:

1. Intrusion upon a person's seclusion or solitude, or into his private affairs.
2. Public disclosure of embarrassing private facts about an individual.
3. Publicity placing one in a false light in the public eye.
4. Appropriation of one's likeness for the advantage of another

To expand on Prosser's privacy rights, the first right refers to the right of a person to be alone or conduct actions without the knowledge of others. In essence, a person has the right to keep secrets using our definition of the word. Prosser's second and third rights protect individuals from analog or digital doxing. The fourth right protects against identity theft in the broadest sense.

Consequences of a Lack of Privacy

Like running water and stable electricity, the value of privacy is constantly taken for granted when it is present. In order to better understand the value of privacy, we will analyze a recent event where privacy has very clearly been taken away from individuals.

As Hong Kong protests continued throughout June 2019, many protestors were forced to switch to analog measures in order to avoid being tracked by the Chinese government. With the implementation of several severe cybersecurity laws and facial-recognition software by the Chinese government in 2018, Hong Kong protestors were forced to adapt: wearing masks and umbrellas to block cameras, disabling location tracking on smart phones, and completing purchases with cash.

During the protests, digital privacy was largely disrupted by the surveillance of the Chinese state. Just the knowledge of the potential of being digitally tracked forced the protestors into behaviours that would otherwise appear odd in a different context. For example, why would you pay with cash if you could just use the more efficient Octopus smart card?

Excluding all of the human rights and legal issues that arise when privacy is explicitly taken away in such a form, perhaps the greatest and most negative consequence of a lack of privacy is the way in which a lack of privacy contains the way people think and, in turn, behave.

An open question to drive further exploration: If you knew that all of your Google searches was being tracked by a state and the wrong search would have severe negative consequences on the life of you and your family, how would that limit your ability to think and express yourself?

One-Way Functions

How do we ensure privacy and secure communication in a digital space?

This is a big question for us dummies.

We will need to look towards mathematics, specifically cryptography, for a solution.

What is a Function?

In mathematics, *mappings* (more commonly known as *functions*) can be defined in the following manner:

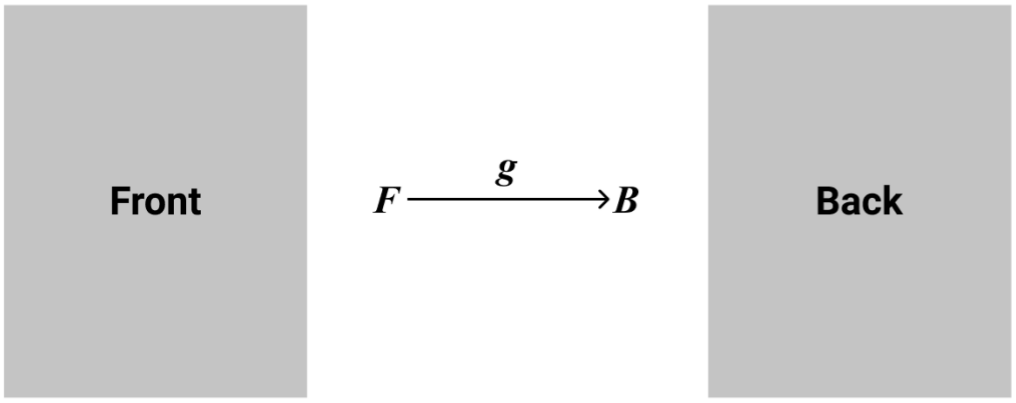
Mapping (Function): A relation between an input to an output. A mapping can be more easily understood as an action that transforms an input into an output.

This definition may be confusing to some of you so we will clarify it with an unexpected example:

Get a single blank paper. Lay out the blank paper in portrait and write “Front” on one side of the paper. Now flip the paper along its length and write “Back”. Looking at our definition, the input is the front of the paper. The output is the back of the paper. The flip alongside the length is the relation between the front and the back of the paper. We have created a mapping:

$$g: F \to B$$

where F is the front of the paper, B is the back of the paper, and g is the flip alongside the paper’s length that maps the front of the paper to the back of the paper. The above notation can be read as g is defined (“defined” is represented by the colon) as a mapping from F to B.



Now, let us define the inverse of a mapping:

Inverse (of a Mapping): Suppose g is a mapping from F to B i.e

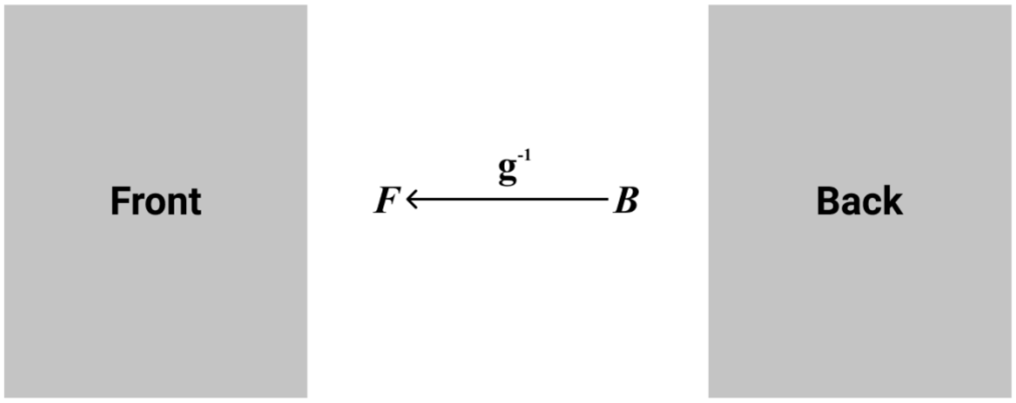
$$g: F \to B$$

The inverse of g denoted g^{-1} is the following mapping:

$$g^{-1}: B \to F$$

In essence, the inverse of a mapping is 1) itself a mapping and 2) is the relation between the output and the input of the mapping.

For clarification, we will continue with our paper-flipping example. Ensure that you are looking at the front of your paper. Flip the paper along its length so that you are now looking at the back of the paper. You have now performed the mapping g. Flip the page once more. You have now performed the mapping g^{-1}.



In this example, g and g^{-1} are the same mapping: they are both a flip along the length of the paper. This is not always the case for mappings as we will see in the next section: some maps do not have inverses that are easily performed or computable.

One-Way Mappings

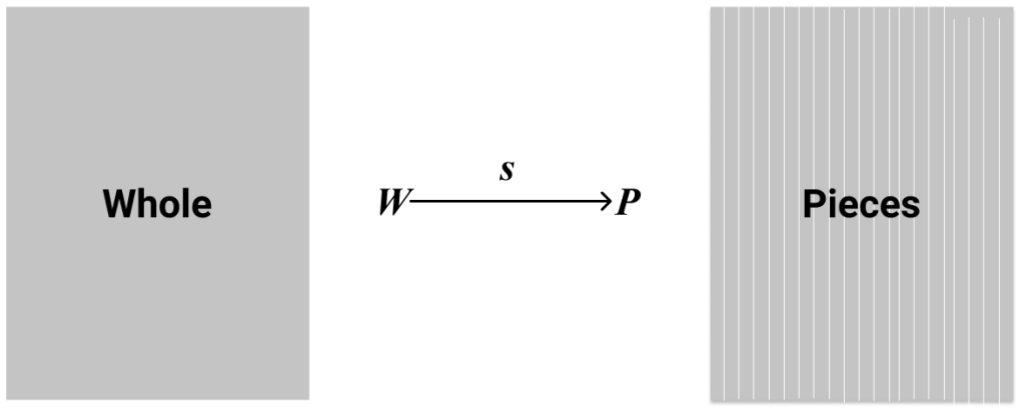
In mathematics, one-way mappings can be defined as follows:

One-Way Mapping: A mapping where it is very easy to compute the output from the input, but it is very difficult to compute the input from the output. In essence, a one-way mapping is defined as a mapping whose inverse is extremely difficult to compute or perform in a short period of time.

Let us go back to our paper-flipping example once more. Imagine that you have a shredder, and you shred your piece of paper. Again, we have created a mapping:

$$s: W \rightarrow P$$

where W denotes the paper when it is whole, and P denotes the paper when it is in tiny pieces. The mapping from W to P is s which denotes shredding the paper.

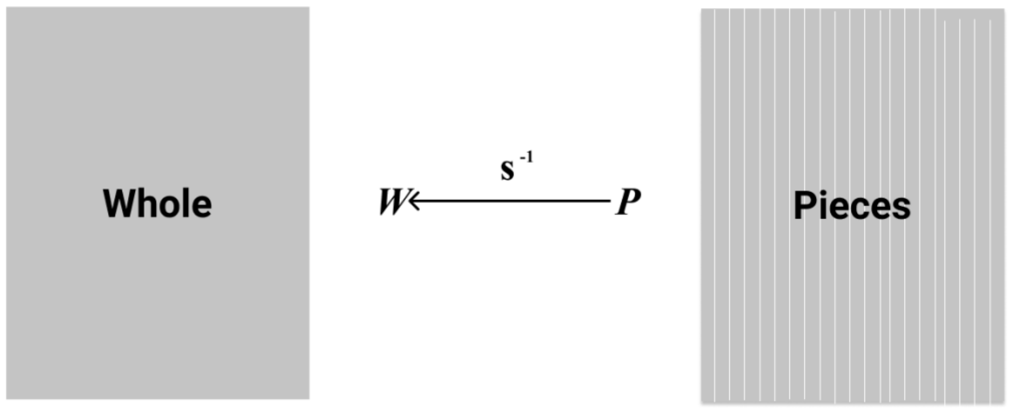


What is the inverse of s (i.e., s^{-1})? Let us look at the notation for a clue:

$$s^{-1}: P \rightarrow W$$

where again P denotes the paper when it is in tiny pieces, and W denotes the paper when it is whole. We see that the mapping from P to W is s^{-1}. Thus, s^{-1} must be a relation (or action) that takes the tiny pieces of shredded paper and turns them back into the whole page.

So, we see that s^{-1} can be something like gluing or taping together the fragments of paper so that they identically match the whole page we had before we shredded it. If the shredding was completed successfully, putting back together the whole page from the shredded pieces would take an enormous amount of time. Thus, shredding paper can be seen as an example of a one-way mapping.



Trapdoor Mappings

In mathematics, *trapdoor mappings* can be defined as follows:

Trapdoor Mapping: A one-way mapping where it is very easy to compute the output from the input, but it is very difficult to compute the input from the output unless you know the “trapdoor” (a special piece of information). In essence, a trapdoor mapping is defined as a mapping whose inverse is extremely difficult to compute or perform in a short period of time unless one has special information that is typically kept secret.

We will thoroughly explore examples of *Trapdoor Mappings* in the following section on cryptography.

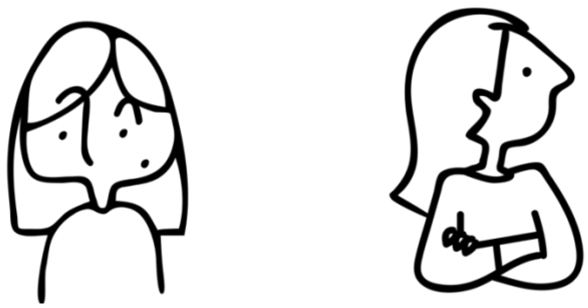
Cryptography

Principal Goal of Cryptography

The principal goal of cryptography is to allow two or more parties to exchange confidential information with one another in a manner where even if the message is intercepted by an adversary, it cannot be understood.

Cryptography Understood via a Contrived Example

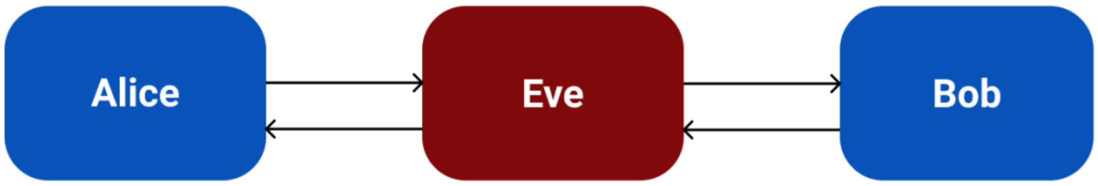
Alice and Bob are students working together on a competitive group project that awards a large cash prize to the winner. Eve is a member of a competing group.



Due to the competition rules, you are unable to leave your desk, but you can pass written notes to your group members. Unfortunately, Alice and Bob are unable to sit beside one another as Eve has taken the desk between them.

Alice
Eve
Bob

Thus, every message passed from Alice to Bob or from Bob to Alice must be transferred via Eve who has an incentive to look at the message.



In order to keep their groupwork confidential, Alice and Bob must use an agreed-upon mechanism to disguise their messages so that Eve cannot understand what they wrote even if she reads it.

Now, let us turn to some basic definitions that we will heavily rely on as we continue our discussion of cryptography:

Basic Definitions

Cipher: a mapping from plaintext (input) to ciphertext (output) or vice versa. A cipher can be understood to be the algorithm or method used for enciphering/deciphering messages.

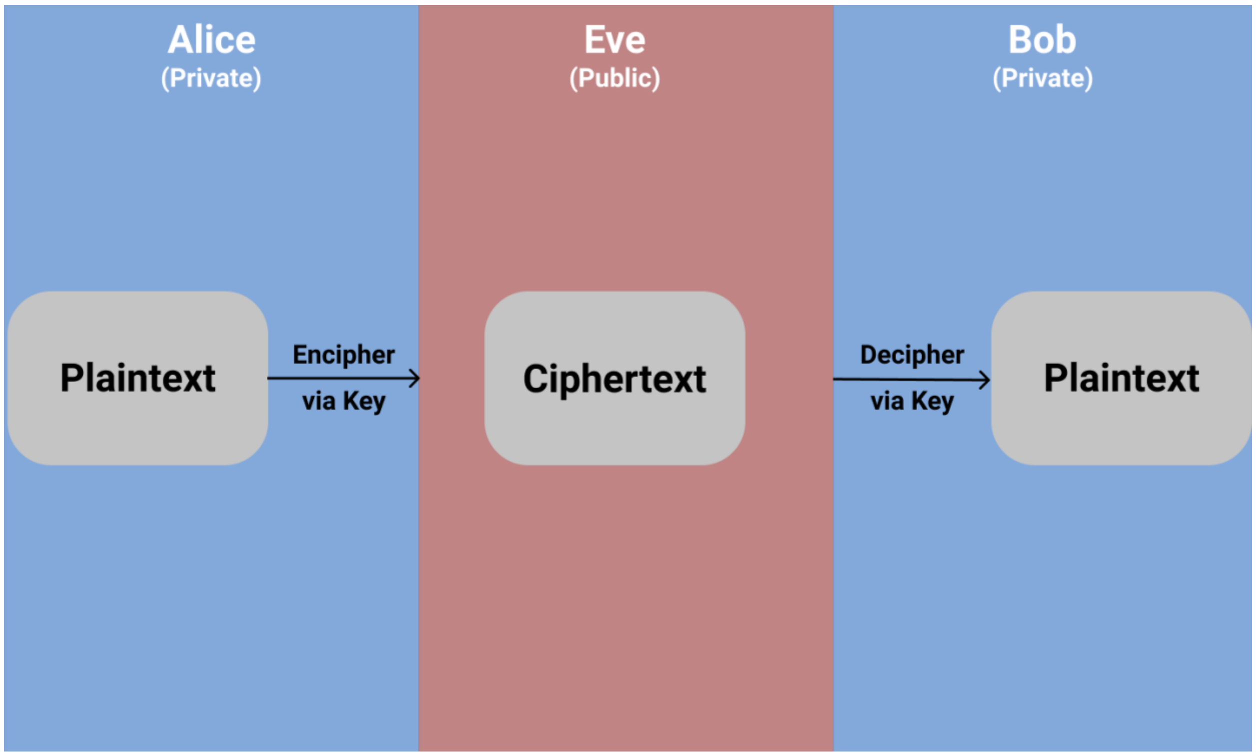
Plaintext: the message to be transmitted or stored.

Ciphertext: the disguised message.

Key: a secret piece of information used with the cipher to convert plaintext to ciphertext or vice versa.

Encipher: to convert plaintext into ciphertext via the key.

Decipher: to convert ciphertext back to plaintext via the key.



Let us return back to our contrived example with Alice, Bob, and Eve. Suppose Alice and Bob want to exchange the following message during the group project:

Plaintext: “LETUSBUILDAROCKET.” (Let us build a rocket)

Of course, Alice and Bob do not want to share this idea with Eve who they are competing against. Luckily, Alice and Bob agreed upon a shared key prior to the event that they can use to encipher the message so that even if Eve looks at the message, she cannot understand it.

The cipher and key that they agreed upon is to **shift each letter** (cipher) in the message by **one letter** (key) down the alphabet. So, to encipher the message:

- A becomes B
- B becomes C
- C becomes D

And so on until,

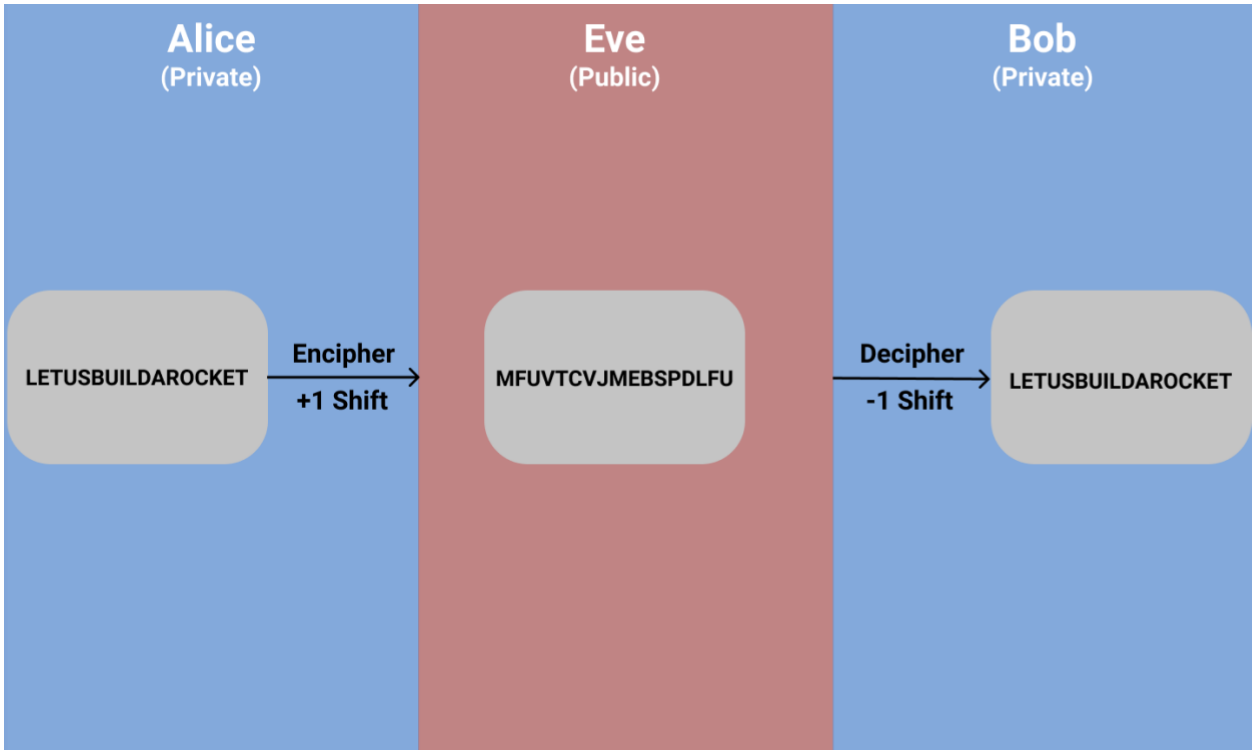
- Z becomes A

Thus, the disguised message using this cipher and key is:

Ciphertext: “MFUVTCVJMEBSPDLFU.”

When Eve receives this message, she will be very confused for one of two reasons: 1) she does not know the cipher i.e., the method of enciphering/deciphering, and has no idea what algorithm to use to decipher the ciphertext or 2) even if she knows that Alice and Bob are shifting letters to disguise their message, she does not know by how many letters (i.e., the key) they have shifted the original message.

This cipher is an example of a *trapdoor mapping*: if one does not know the key, it is very difficult to convert the ciphertext into plaintext.



Admittedly, there are many cryptographic attacks than can be performed on such a simple encryption scheme; this scheme is most definitely not considered to be secure by modern cryptographic standards. We will explore cryptographic attacks as we delve deeper into the specific cryptographic system that underpins Bitcoin.

Introduction to Private-Key (Symmetric) Cryptography

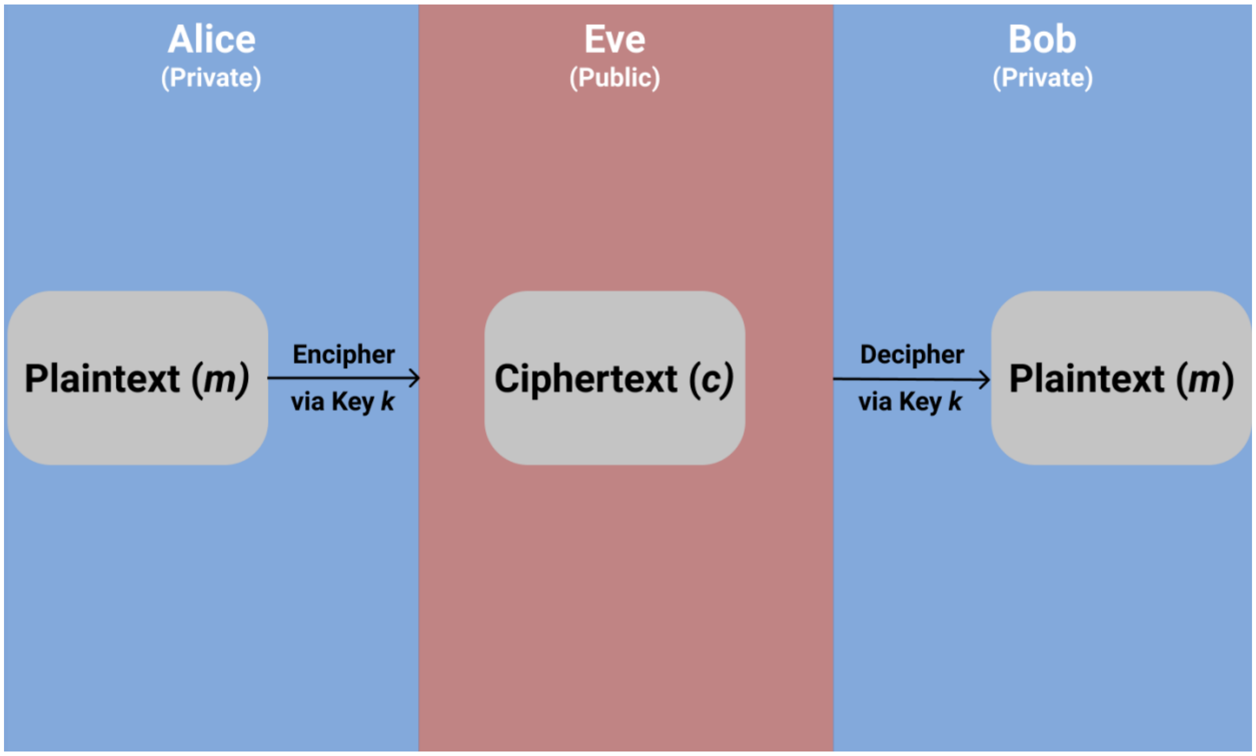
In the extended example with Alice, Bob, and Eve in the previous section, we demonstrated examples of Private-Key (Symmetric) Cryptography.

Overview

Private-Key (Symmetric) Cryptography requires both parties, Alice and Bob, to share a secret key prior to exchanging messages. This method is called *symmetric* because both Alice and Bob have the same level of knowledge of the secret key.

General Setup

1. Bob and Alice share a secret key *k* through a private channel.
2. Bob wants to send a plaintext message *m* to Alice in a secret manner so that an eavesdropper, Eve, cannot understand the message.
3. Bob enciphers the message with the shared secret key. I.e., the plaintext turns into ciphertext *c*.
4. Bob sends the ciphertext to Alice through a public channel.
5. After receiving the ciphertext, Alice deciphers the ciphertext into plaintext via the key.
6. If the procedure is conducted properly, Eve cannot know the key, cannot guess they key, or cannot convert the ciphertext to plaintext without knowing the key.



Some examples of Private Key Cryptography include:
- Caesar Cipher (Practice Problem 1)
- Vigenère cipher
- Advanced Encryption Standard
- Data Encryption Standard

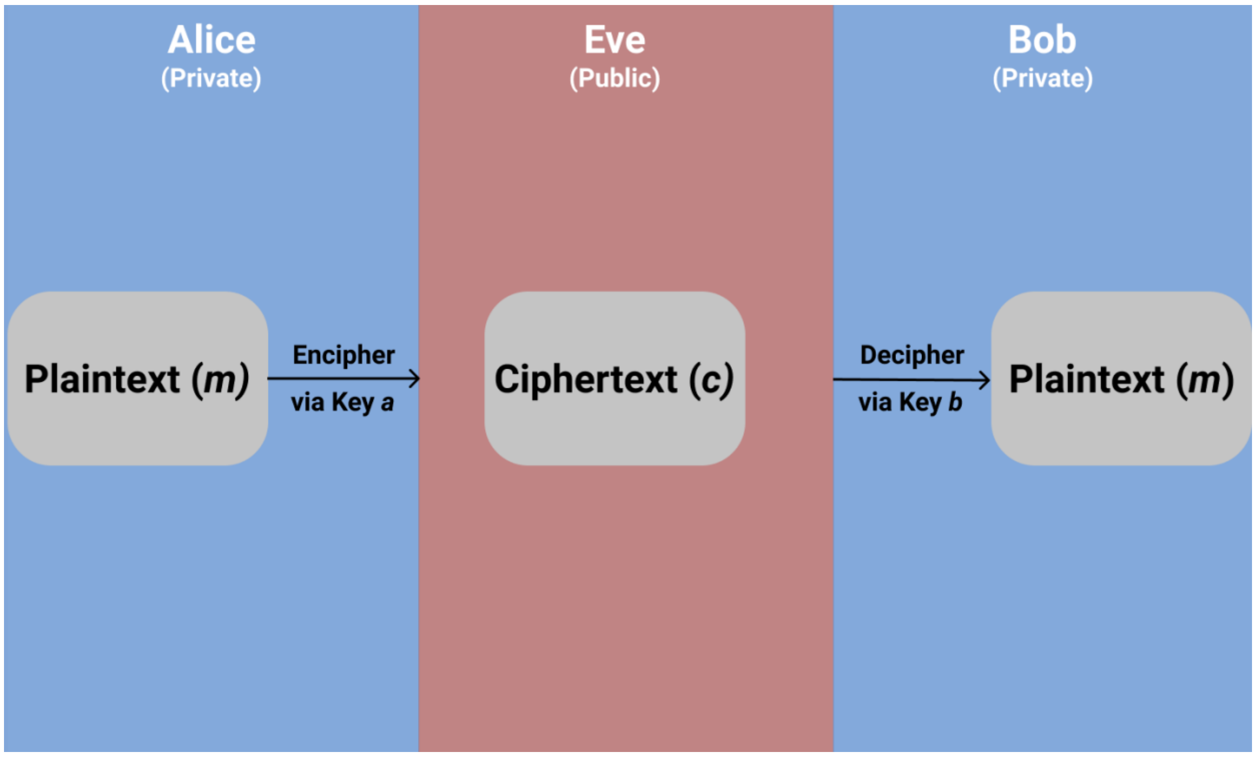
Introduction to Public-Key (Asymmetric) Cryptography

Overview

Public-Key (Asymmetric) Cryptography does not require both parties, Alice and Bob, to share a secret key prior to exchanging messages. In Public-Key Cryptography, Alice has her own secret key and Bob has his own secret key. When combined together by some mathematical operation, these two private keys form the shared secret key. This method is called *asymmetric* as Alice and Bob have different levels of knowledge of the shared secret key.

General Setup

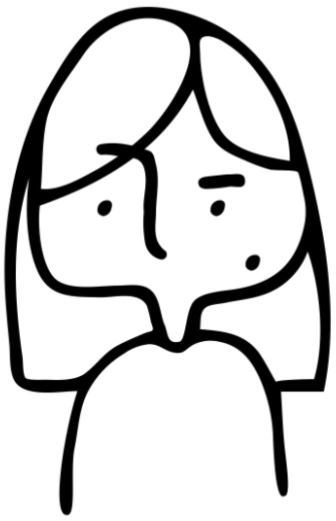
1. Bob and Alice do not share a secret key through a private channel.
2. Alice has her own secret key *a*.
3. Bob has his own secret key *b*.
4. Bob wants to send a plaintext message *m* to Alice in a secret manner so that an eavesdropper, Eve, cannot understand the message.
5. Bob enciphers the message with his secret key. I.e., the plaintext turns into ciphertext *c*.
6. Bob sends the ciphertext to Alice through a public channel.
7. After receiving the ciphertext, Alice applies her secret key to decipher the ciphertext into plaintext.
8. If the procedure is conducted properly, Eve cannot know the either Alice's or Bob's key, cannot guess their keys, or cannot convert the ciphertext to plaintext without knowing either one of their keys.



You may be asking yourself: if the two secret keys are different, how do we ensure that the plaintext that Bob receives is identical to the plaintext the Alice sent? This is a great question and will be explored in the next chapter.

Public-Key Cryptography is more complicated and harder to grasp than Private-Key Cryptography. That is okay; we will spend the next article exploring Public-Key Cryptography as well as the specific public-key cryptosystem, Elliptic Curve Cryptography, that underlies the Bitcoin protocol.

Chapter 2: Public-Key Cryptography



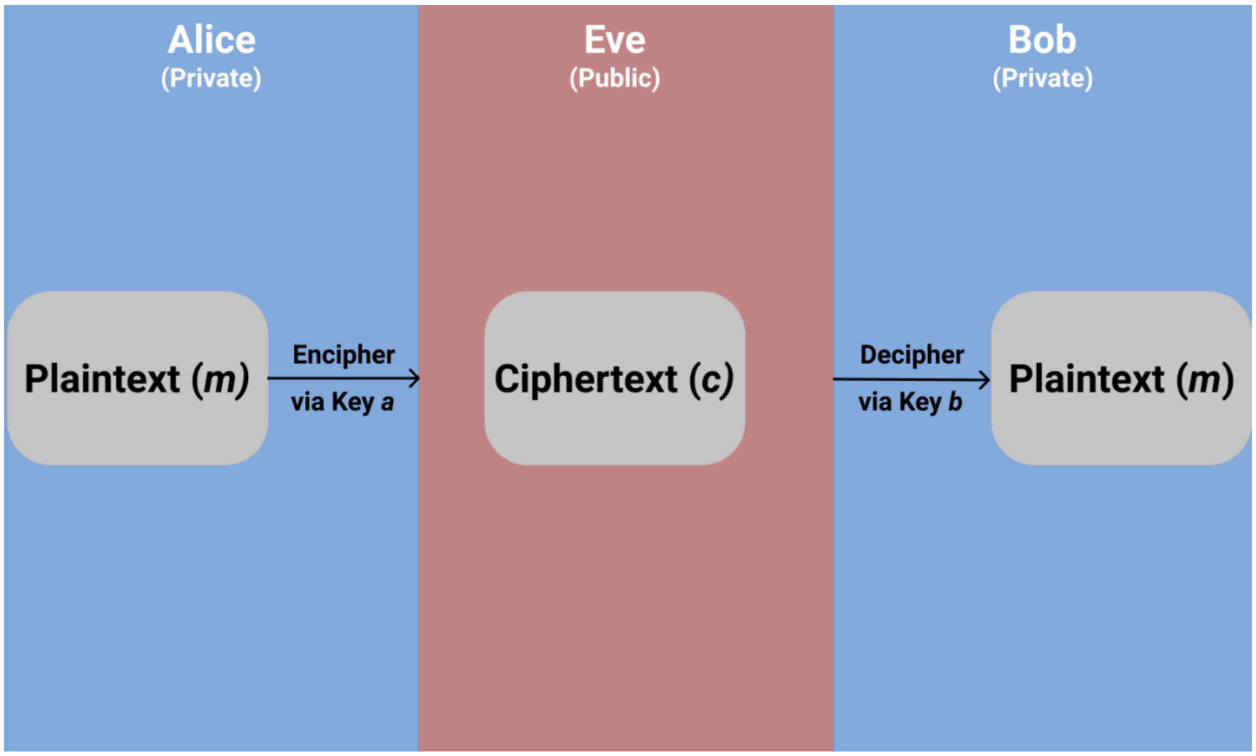
Public-Key (Asymmetric) Cryptography

Review

As discussed in the last chapter, *Public-Key (Asymmetric) Cryptography* does not require both parties, Alice and Bob, to share a secret key prior to exchanging messages. In Public-Key Cryptography, Alice has her own secret key and Bob has his own secret key. When combined together by some mathematical operation, these two private keys form the shared secret key. This method is called *asymmetric* as Alice and Bob have different levels of knowledge of the shared secret key.

General Setup

1. Bob and Alice do not share a secret key through a private channel.
2. Alice has her own secret key *a*.
3. Bob has his own secret key *b*.
4. Bob wants to send a plaintext message *m* to Alice in a secret manner so that an eavesdropper, Eve, cannot understand the message.
5. Bob enciphers the message with his secret key. I.e., the plaintext turns into ciphertext *c*.
6. Bob sends the ciphertext to Alice through a public channel.
7. After receiving the ciphertext, Alice applies her secret key to decipher the ciphertext into plaintext.
8. If the procedure is conducted properly, Eve cannot know the either Alice's or Bob's key, cannot guess their keys, or cannot convert the ciphertext to plaintext without knowing either one of their keys.



Throughout this chapter, we will answer the following question: if the two secret keys are different, how do we ensure that the plaintext that Bob receives is identical to the plaintext that Alice sent?

Importance of Public-Key Cryptography

Public-Key Cryptography enables 3 properties to emerge in digital space:

1. *Non-repudiation* – once she sends a message, Alice cannot falsely claim that she did not send it. Similarly, once he receives a message, Bob cannot falsely claim that he did not receive it.
2. *Confidentiality* – Bob alone can read Alice's messages.
3. *Authenticity* – Alice can "sign" her messages to prove to Bob that only she could have sent it in a manner that prevents tampering from Eve.

Public-Key Cryptography is the foundation that most encryption schemes are built upon in modern computing and networking. Whether you are using email, social media, electronic payments, or electronic signatures, you are interacting in one way or another with Public-Key Cryptography. In fact, every time you use a secure connection over the internet via a VPN, you interact with Public-Key Cryptography.

Modular Arithmetic

An Introduction to Modular Arithmetic

Similar to the way that calculus contains building blocks that underpin psychical systems, modular arithmetic contains building blocks that make up Public-Key Cryptosystems.

In this section, we will explore modular arithmetic. Without a rudimentary knowledge of this subject, moving forward while maintaining a high level of understanding is frankly infeasible. Therefore, we must cover this (admittedly dry) material for readers unfamiliar with the topic.

For readers who are familiar with modular arithmetic, feel free to skip this section. For readers who wish to delve deeper into the mathematics of groups and fields – algebraic structures closely related to modular arithmetic – see the following:

- Sections 1.3 – 1.5 in *An Introduction to Mathematical Cryptography* by Hoffstein 2014[1]

Integers and Primes

Before delving into modular arithmetic, let us first remind ourselves of two very basic definitions:

Integers: numbers (whole numbers) that can be written without the use of fractions or decimals. We will focus on the positive integers (0, 1, 2, 3, …).

Primes: integers that can only be divided by 1 or themselves. A few examples of small primes are 2, 3, 5, 7, 11, and 13.

Telling Time

What happens to the time on your smartphone as midnight approaches?

If you set your smartphone's time to 24-hour (military time) instead of AM/PM, you will see that the time approaches 23:59 and then resets to 00:00 at midnight. If you struggle to interrupt military time, please read this short guide for a quick explanation (Military Time Chart).

Let us ignore minutes and just focus on the hours displayed in military time. If you were to stay up throughout an entire 24-hour period starting at midnight, you would see your smartphone display 0, 1, 2, 3, …, 22, and 23 until the next day began where again your smartphone would display 0. Focusing only on hours, there are 24 different types of units (in this case, integers or whole numbers) that can be displayed by your smartphone throughout an entire 24-hour period:

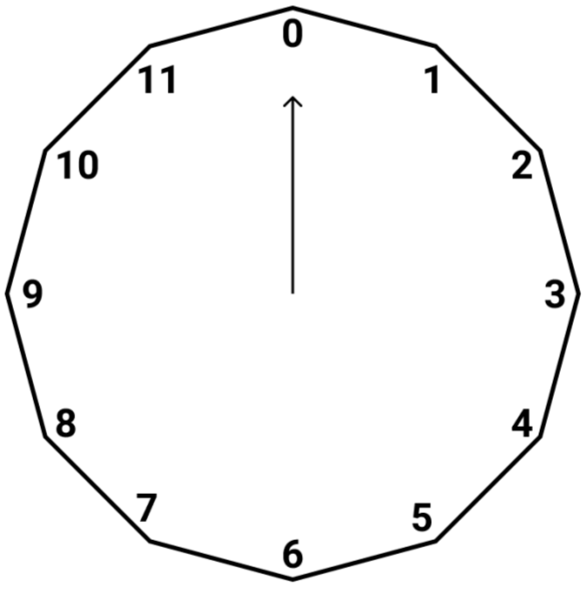
$$\mathbb{Z}_{24} = 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23$$

The symbol above denotes *the set of Integers modulo 24*. Do not let the scary name or notation fool you; when you are telling time (ignoring minutes) using a 24-hour clock, you are interacting with *the set of Integers modulo 24*. All this notation means is that there are 24 units (in this case, integers or whole numbers) that you can work with. If you were to add 23+1, it would be just like waiting an hour starting at 23: your smartphone would indicate that the time is 0.

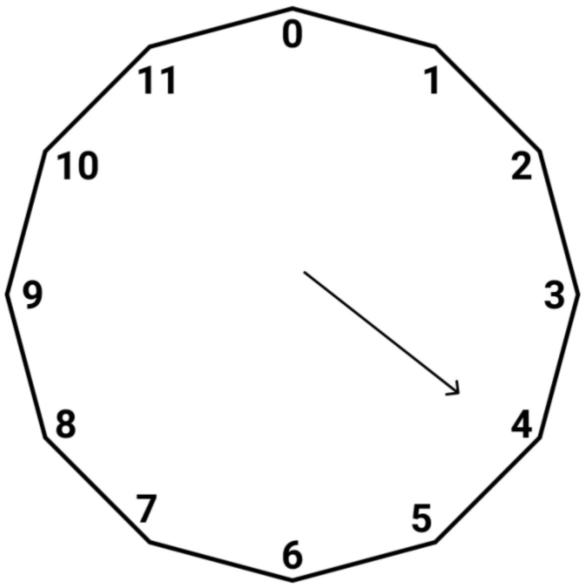
Modular Addition

Telling time (ignoring minutes) using a 24-hour clock is an example of modular arithmetic that you are intimately familiar with; however, very few individuals are familiar with the symbolic notation. There is a power to identifying the name or definition of a thing or action which we will try to capture in the following example:

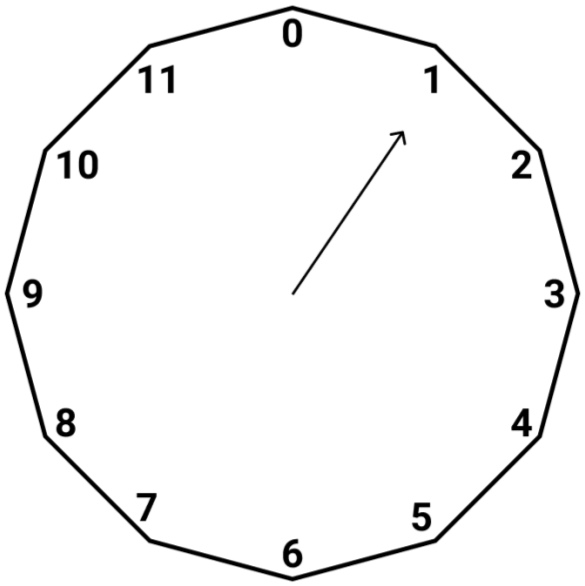
Let us now look at a wall clock that only displays hours with the 12 replaced by a 0. There are 12 units (integers) on this clock that denote the time in hours that range from 0 to 11:



If it is currently 1, what will be the time in 3 hours? This is trivial – it will be 4.



If it is currently 10, what will be the time in 3 hours? This is not difficult looking at the clock. You move 3 units clockwise from 10 until you land on the desired unit: 1.



What is going on here from a technical perspective? *Integer additional modulo 12*. Below is the *set of integers modulo 12* that corresponds to our clock:

$$\mathbb{Z}_{12} = 0,1,2,3,4,5,6,7,8,9,10,11$$

If it is currently 1 and we want to know the time in 3 hours, we perform modular addition:

$$1 + 3 \;= 4 = 4 \, mod \; 12$$

If its currently 10 and we want to know the time in 3 hours, we again perform modular addition:

$$10 + 3 \;= 13 = 1 \, mod \; 12$$

The *mod 12* part of this equation will be new to many of you. Do not fear, all *mod* indicates is the type of "clock" that you are working with. In this case, it is a clock with units 0 through 11 as above.

If we perform the following modular addition with a 24-hour "clock":

$$22 + 3 \;= 25 \;= 1 \, mod \; 24$$

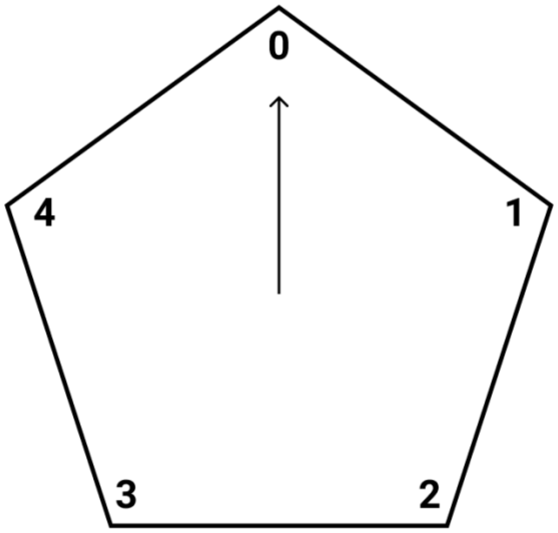
we see that the principles are the same.

Modular Addition (mod N):

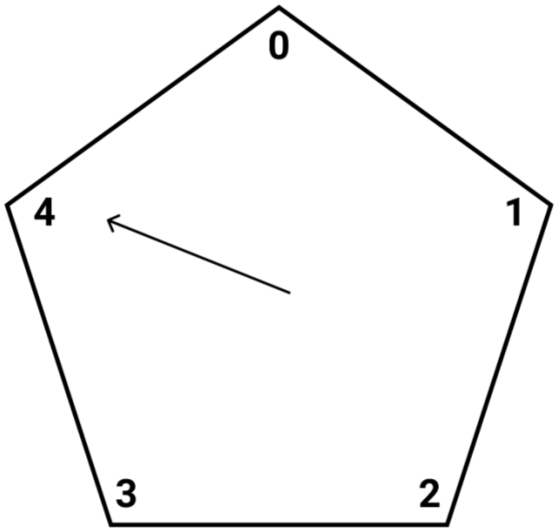
1. *Suppose you have two integers x and y the sum z (i.e., x + y = z). If you want to solve x + y = z mod N, construct a "clock" as above for size N.*
2. *Start at 0 and move around the clock in a clockwise direction x times. Where you land on the clock is x mod N.*
3. *Next, from x mod N, move along the clock in a clockwise direction y times. Where you land on the clock is x + y mod N = z mod N.*

Let x = 9 and y = 3. We want to solve x + y = z mod N where N = 5.

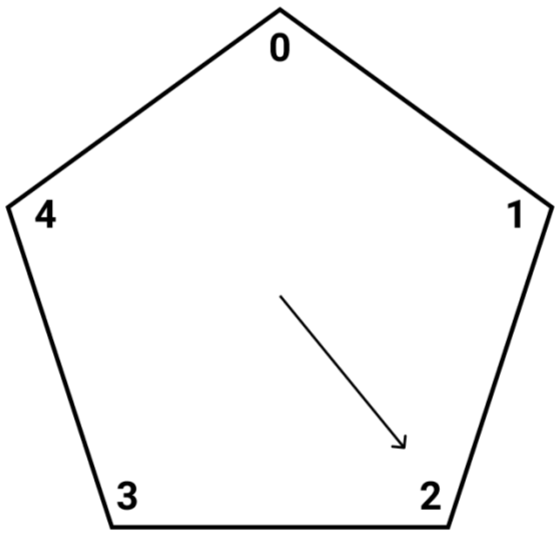
First, construct a clock of size N = 5:



Second, starting from 0, move clockwise 9 times:



Third, starting from 4, move clockwise 3 times:



You have arrived at the solution.

We have just performed modular addition:

$$\begin{aligned} x &= 9 &&= 4 \ mod\ 5 \\ y &= 3 &&= 3 \ mod\ 5 \\ x + y &= 9 + 3 &&= 12 = 2 \ mod\ 5 \end{aligned}$$

Note: when performing modular arithmetic, it is sometimes easier to perform the calculations using standard arithmetic and then reduce the final answer by modulo N via a modulo calculator. For those who are following along with the calculations in this section, you can access a modulo calculator here.

Modular Subtraction

Let us now turn to modular subtraction.

Modular subtraction is very similar to modular addition with one caveat: we add in the counter-clockwise direction on our clock.

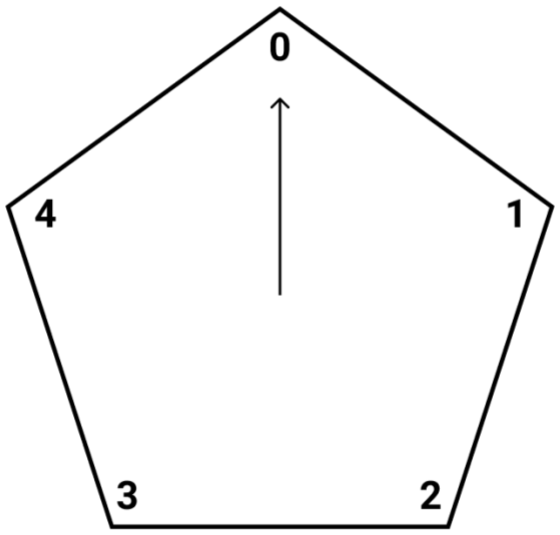
Modular Subtraction (mod N):

1. *Suppose you have two integers x and y the difference z (i.e., x - y = z). If you want to solve x - y = z mod N, construct a "clock" for size N.*

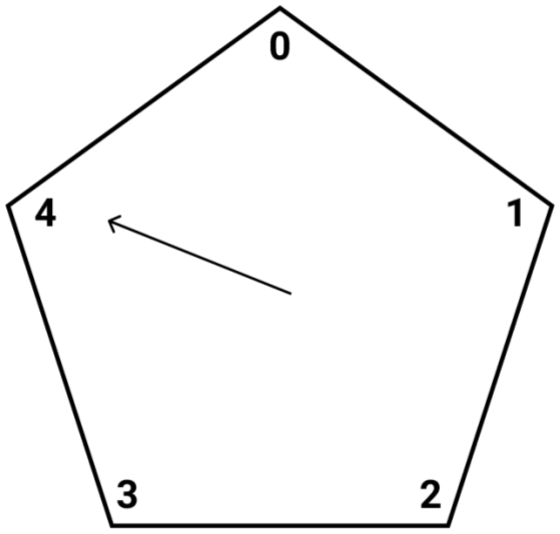
2. *Start at 0 and move around the clock in a clockwise direction x times. Where you land on the clock is x mod N.*
3. *Next, from x mod N, move along the clock in a counter-clockwise direction y times. Where you land on the clock is x - y = z mod N.*

Let x = 9 and y = 3. We want to solve x - y = z mod N where N = 5.

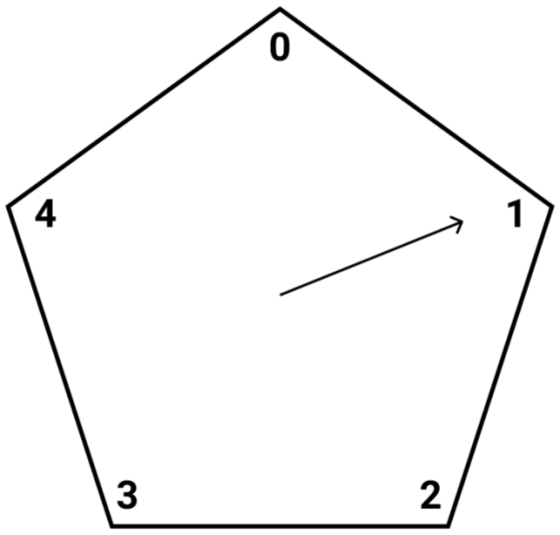
First, construct a clock of size N = 5:



Second, starting from 0, move clockwise 9 times:



Third, starting from 4, move counter-clockwise 3 times:



You have arrived at the solution.

We have just performed modular subtraction:

$$\begin{aligned} x &= 9 &&= 4 \; mod \; 5 \\ y &= 3 &&= 3 \; mod \; 5 \\ x - y &= 9 - 3 &&= 6 = 1 \; mod \; 5 \end{aligned}$$

Modular Multiplication

Let us now turn to modular multiplication:

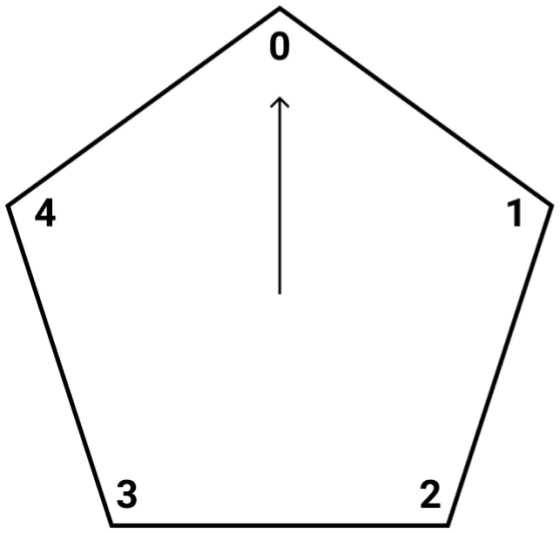
Modular Multiplication: Let x, y, and z be integers modulo N. We define modular multiplication to be the arithmetic:

xy = z mod N

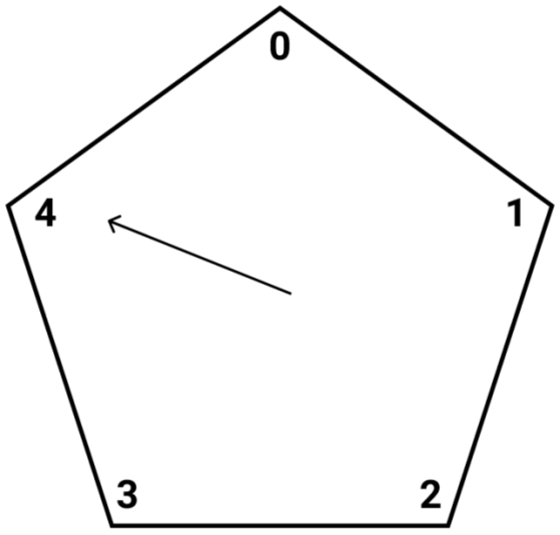
Modular multiplication, like the regular multiplication that you are used to, is basically repeated addition.

Let x = 9 and y = 3. We want to solve xy = z mod N where N = 5.

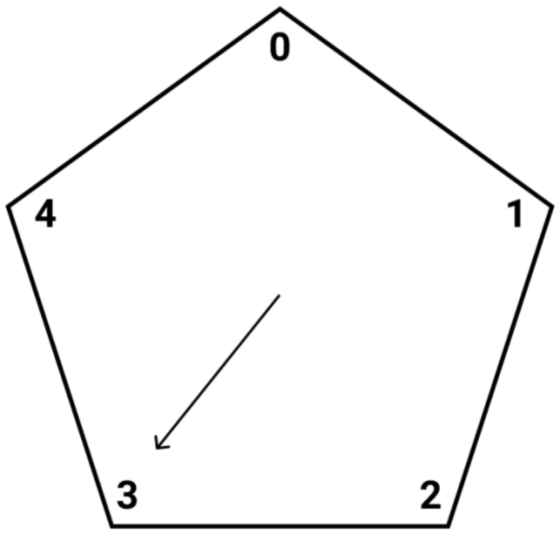
First, construct a clock of size N = 5:



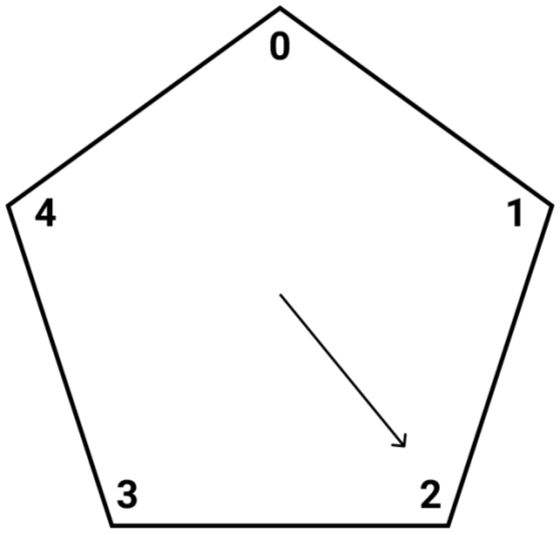
Second, starting from 0, move clockwise 9 times:



Third, starting from 4, move clockwise 9 times:



Fourth, starting from 3, move clockwise 9 times:



You have arrived at the solution.

We have just performed modular multiplication:

$$\begin{aligned} x &= 9 = 4 \; mod\; 5 \\ y &= 3 = 3 \; mod\; 5 \\ xy &= 9(3) = 27 = 2 \; mod\; 5 \end{aligned}$$

Modular Multiplicative Inverses

Let us now turn to find a modular multiplicative inverse – the equivalent of division in modular arithmetic.

First, we will introduce a concept known as the Greatest Common Divisor:

Greatest Common Divisor (gcd): Let x and y be integers. The Greatest Common Divisor of x and y (denoted gcd(x,y)) is the largest number that divides both x and y.

Example: What is *gcd(6,3)*?

A naïve method to solve this example would be to find all the numbers that divide 6 and all the numbers that divide 3 and compare:

Factors (Divisors) of *6: 1,2,3,6*
Factors of *3: 1,3*

We have a match: *gcd(6,3) = 3.*

There are faster algorithms to find the gcd of two or more integers; one algorithm that is commonly used is the Euclidean Algorithm.

Returning to modular multiplicative inverses:

Modular Multiplicative Inverse: Let x and y be integers modulo N. If xy = 1 mod N, then y is the modular multiplicative inverse of x and vice-versa. Furthermore, x has a modular multiplicative inverse if and only if gcd(x,N) = 1. In most applications of cryptography, N = p prime so gcd(x,N) = 1 holds true.

Example: What is the modular multiplicative inverse of 2 mod 5?

First, let's check if 2 mod 5 has a modular multiplicative inverse:

Factors of *2: 1,2*
Factors of *5: 1,5*

$$gcd(2,5) = 1$$

Great! *2* has a modular multiplicative inverse! Now let's find it by naïve trial and error:

$$2(1) = 2 = 2 \bmod 5$$
$$2(2) = 4 = 4 \bmod 5$$
$$2(3) = 6 = 1 \bmod 5$$

Awesome! The modular multiplicative inverse of 2 mod 5 is 3 mod 5 as 2(3) mod 5 = 1 mod 5!

A common notation that we will use for multiplicative modular inverses is as follows:

$$x = y^{-1} \bmod N$$

which is equivalent to:

$$xy = 1 \bmod N$$

For those who are closely following along with the arithmetic, you can find a modular multiplicative inverse calculator here.

Modular Exponentiation

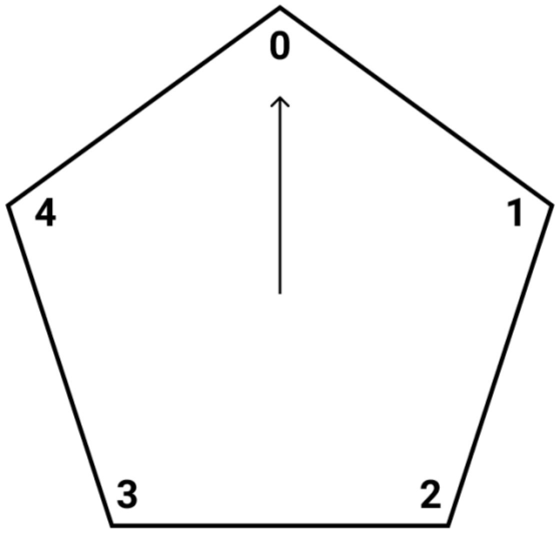
Let us now turn to modular exponentiation. Similar to typical exponentiation, modular exponentiation can be understood as repeated modular multiplication.

Modular Exponentiation: Let x, y, and z be integers modulo N. We define modular exponentiation to be the arithmetic:

$x^y = z \bmod N$

Let x = 9 and y = 3. We want to solve $x^y = z \bmod N$ where N = 5.

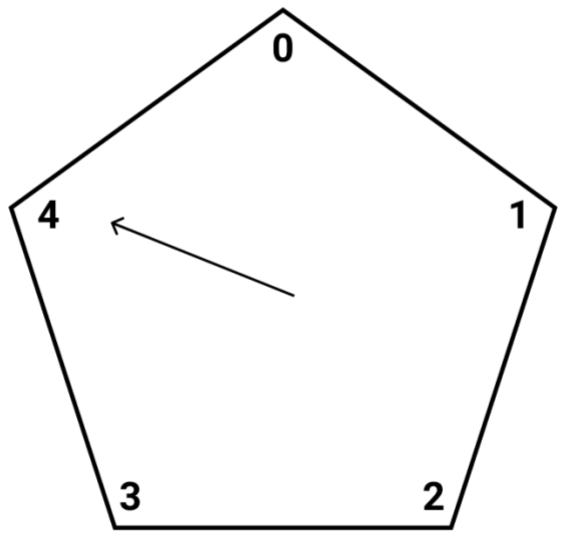
First, construct a clock of size N = 5:



Second, rewrite the exponentiation as repeated multiplication for simplicity:

$9^3 = 9(9)(9) = 729$

Third, perform modular multiplication to arrive at the solution:



We have just performed modular multiplication:

$$\begin{aligned} x &= 9 = 4 \; mod\; 5 \\ y &= 3 = 3 \; mod \; 5 \\ x^y &= 9^3 = 9(9)(9) = 729 = 4 \bmod 5 \end{aligned}$$

Let us now define multiplicative order – an extension of modular exponentiation that is used heavily in the following section:

Multiplicative Order: Let x be an integer modulo N. We define the multiplicative order of x to be the integer n such that:

$$x^n = 1 \bmod N$$

If *x = 3* and *N = 5*, what is the multiplicative order of *x*? Let us find *n* by naïve trial and error:

$$\begin{aligned} x^1 &= 3 = 3 \; mod \; 5 \\ x^2 &= 9 = 4 \bmod 5 \\ x^3 &= 27 = 2 \bmod 5 \\ x^4 &= 81 = 1 \bmod 5 \end{aligned}$$

So, *x* has a multiplicative order of *4*!

Multiplicative order leads us to the final tool that we will need to construct the Discrete Log Problem:

Primitive Root of p: Let p be a prime. We define a primitive root of p to be an integer x modulo p such that the multiplicative order of x is (p – 1):

$$x^{(p-1)} = 1 \bmod N$$

In our previous example, $x = 3$ and $p = 5$.

So, *3* is a primitive root of *5:*

$$x^4 = x^{p-1} = 1 \text{ mod } 5$$

Discrete Log Problem

For the mathematically inclined, the next sections correspond with the following:

- Sections 2.2 – 2.4 in *An Introduction to Mathematical Cryptography* by Hoffstein 2014[1]

Return of Trapdoor Functions

In Chapter 2, we introduced the notion of trapdoor functions. It is in this section that the concept of trapdoor functions makes a return through the Discrete Log Problem.

Reviewing the previous section of this chapter, we know modular exponentiation has the form:

$$x^{y} = z \bmod N$$

Suppose we know $x = 3$, $z = 5$, and $N = 17$:

$$3^{y} = 5 \bmod 17$$

How hard is it to find *y*?

It turns out that for this example, $y = 5$ and is not that difficult to find by plugging in numbers from 0 to 16.

However, in general, finding *y* is extremely difficult for very large numbers. It is this computational difficulty that underpins the Discrete Log Problem.

Of course, if you know *y* it is very easy to check as modular exponentiation is very fast using a modern computer. The "trapdoor" in this setup is knowledge about *y* and its properties. The more you know about *y*, the easier it is to solve the aforementioned equation.

Discrete Log Problem

To construct the Discrete Log Problem, we must use the following tools that were defined in the previous section:

- Integers
- Primes
- Modular Exponentiation
- Multiplicative Order
- Primitive Roots

Discrete Log Problem: Let x be a primitive root of a prime p. Let z be a positive integer modulo p. The Discrete Logarithm Problem (DLP) is the mathematical problem of finding an integer y such that:

$$x^{y} = z \bmod p$$

Attacks

If constructed properly (i.e., large, carefully chosen integers), the DLP is extremely difficult to solve for modern computers. We will not spend a lot of time focusing on potential attacks in this chapter; instead, we will link a few contemporary generic attacks on the DLP for those who would like to learn more:

1. Baby-Step Giant-Step algorithm
2. The Pollard's ρ algorithm
3. Index calculus algorithm
4. Pohlig-Hellman algorithm

Diffie-Helman Key Exchange

Diffie-Helman Key Exchange

In 1976, Diffie and Hellman created the first Public-Key protocol: the Diffie-Helman Key Exchange. This cryptographic protocol uses the same tools that we set up in the previous section on the DLP to create an ingenious method to exchange secret keys across public channels.

Throughout this chapter, we are focusing on exploring the question: if the two secret keys are different in Public-Key Cryptography, how do we ensure that the plaintext that Bob receives is identical to the plaintext that Alice sent?

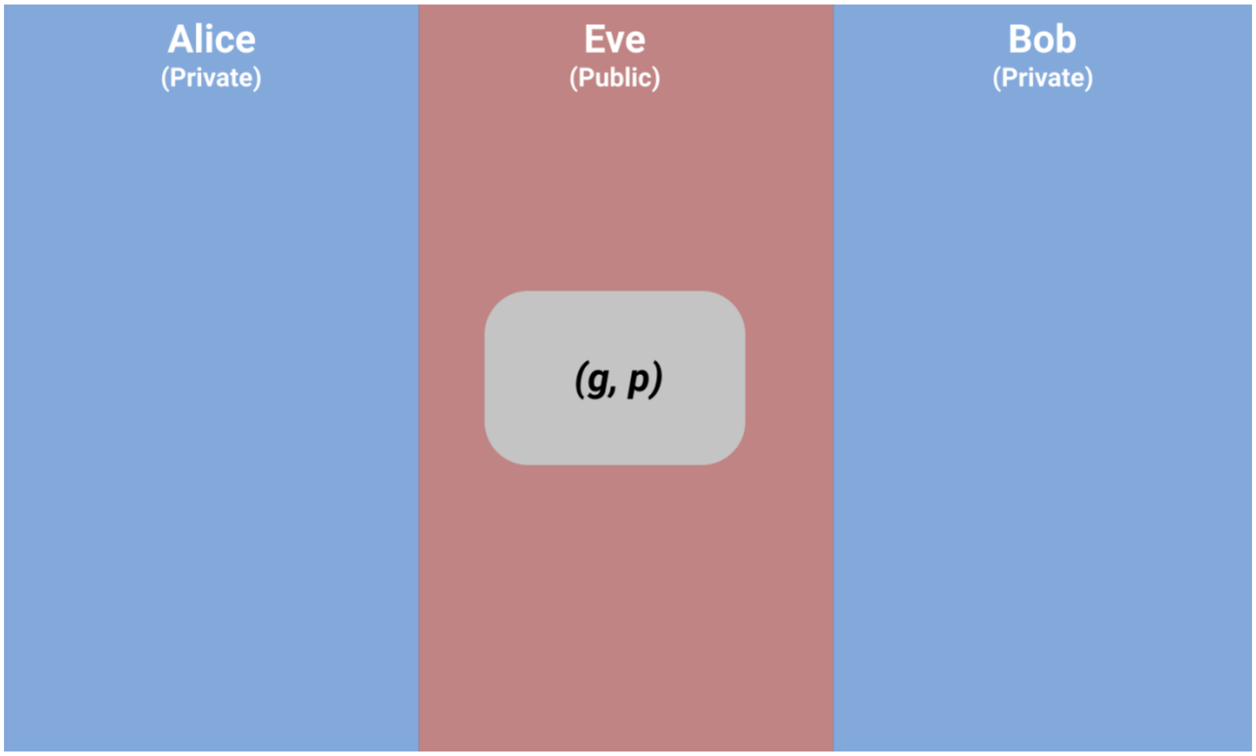
The Diffie-Helman Key Exchange provides an answer to this question: Alice and Bob can create a shared key without exposing their either of their private keys to Eve or to the other individual.

You may be asking yourself: What the hell does that mean?

Well, let us examine the setup of the Diffie-Helman Key Exchange:

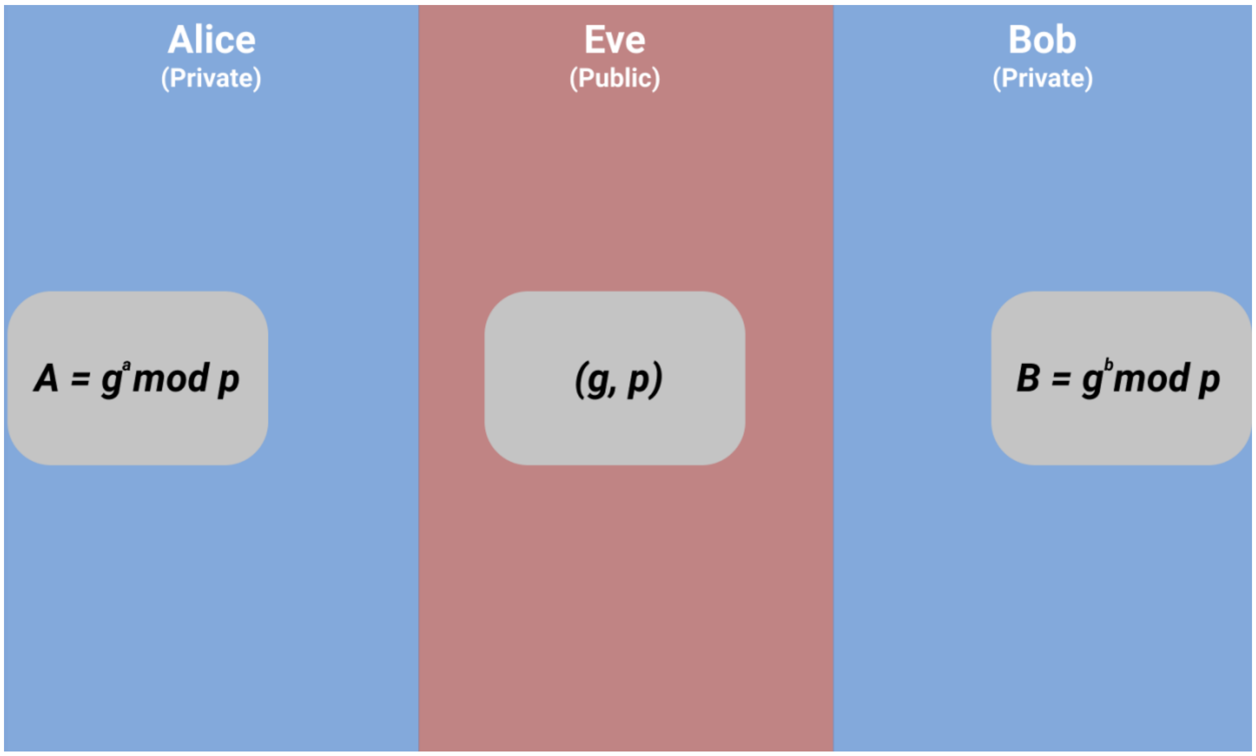
Creation of Public Parameters

1. Bob and Alice do not share a secret key through a private channel.
2. Alice or Bob chooses and publishes a (large) prime p and a primitive root g *mod* p. These two values are public and shared by Alice and Bob.



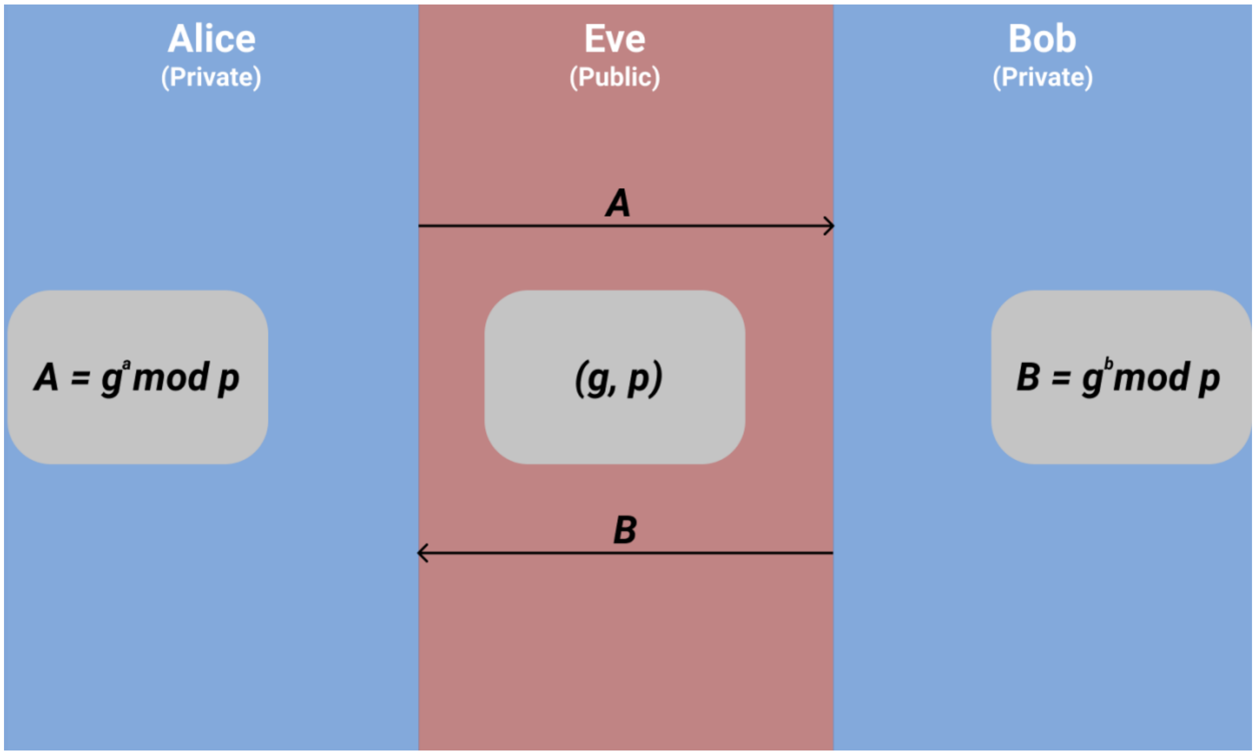
First Private Computations

3. Alice chooses her own secret key a.
4. Bob chooses his own secret key b.
5. Alice computes $A = g^a \bmod p$.
6. Bob computes $B = g^b \bmod p$.



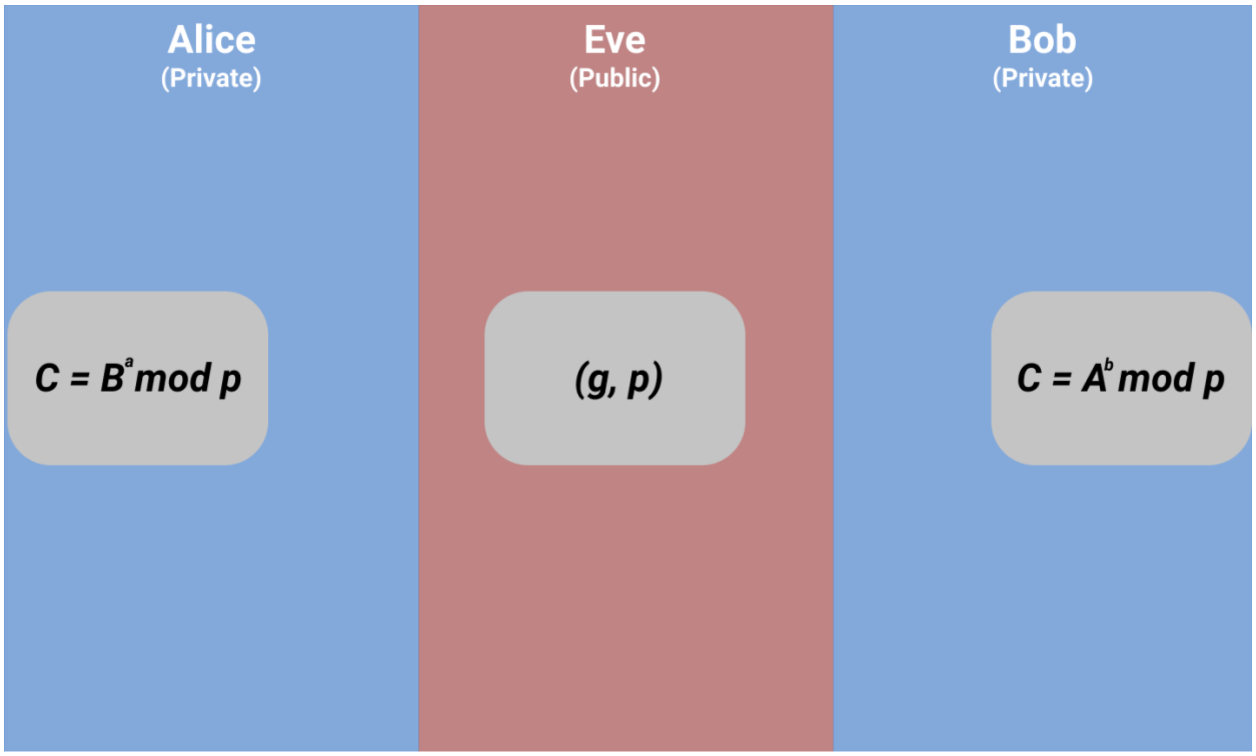
Public Exchange

7. Alice sends *A* to Bob via a public channel.
8. Bob sends *B* to Alice via a public channel.



Second Private Computations

9. Alice computes $B^a \bmod p$.
10. Bob computes $A^b \bmod p$.
11. By the properties of modular exponentiation, $C = A^b \bmod p = B^a \bmod p$. So, C is their shared secret key that they can use in further computations without exposing a or b.



To see that $A^b \bmod p = B^a \bmod p$, notice that $A^b = (g^a)^b = (g^{ab}) = (g^b)^a = B^a \bmod p$ by the properties of modular exponentiation.

The security of this protocol relies on the difficulty on solving the DLP. An attacker knows $p, g, A,$ and B. To find Alice's or Bob's secret keys, they would have to solve the DLP:

- Solve for a for $A = g^a \bmod p$ (Alice's Secret Key)

- Solve for b for $B = g^b \bmod p$ (Bob's Secret Key)

If the appropriate integers are chosen, this setup is extremely difficult for even modern supercomputers to break.

We will construct an (very insecure) example with small numbers to demonstrate the utility of the Diffie-Helman Key Exchange:

Creation of Public Parameters

- Alice and Bob agree to use the fowling public parameters $(g, p) = (2, 13)$.

First Private Computations

- Alice chooses her secret key $a = 3$ and computes $A = g^a \bmod p = 8 \bmod 13$.

- Bob chooses his secret key $b = 7$ and computes $B = g^b \bmod p = 11 \bmod 13$.

Public Exchange

- Alice sends *A* to Bob, while Bob sends *B* to Alice.

Second Private Computations

- Alice computes $C = B^a \bmod p = 5 \bmod 13$.
- Bob computes $C = A^b \bmod p = 5 \bmod 13$.

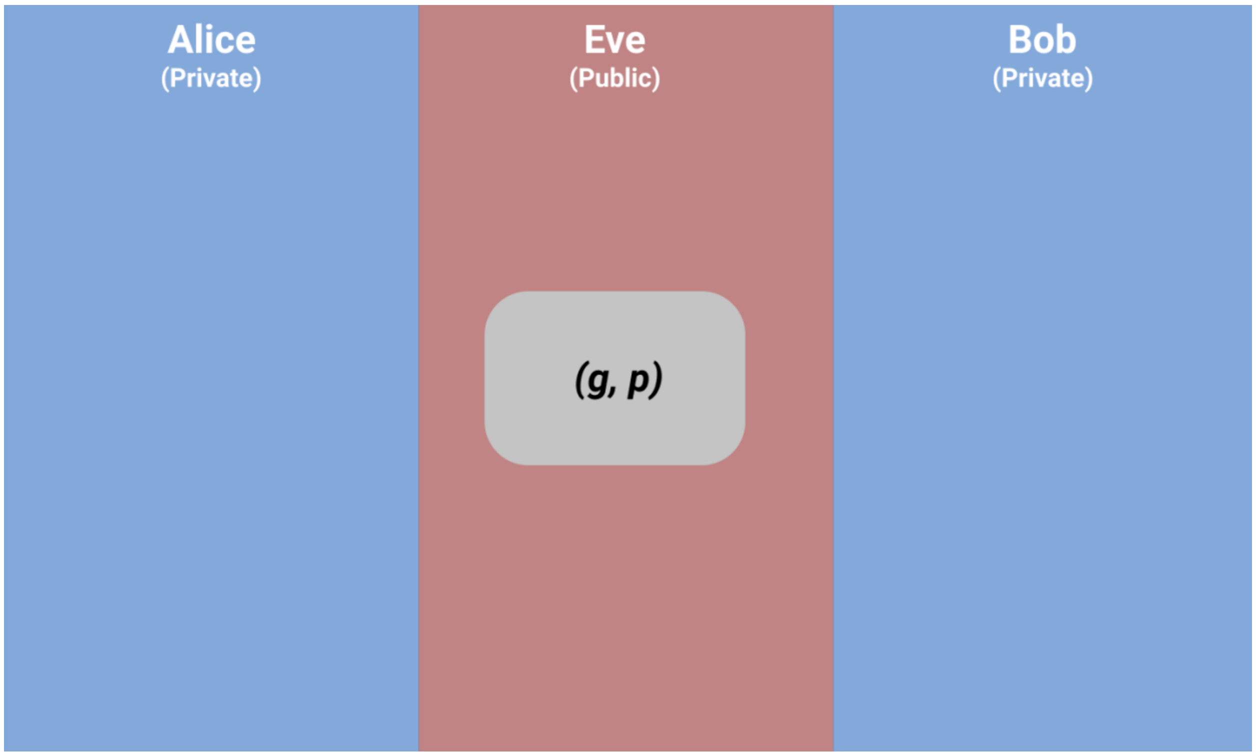
Elgamal Public-Key Cryptosystem

Elgamal: A Functional Public Key Cryptosystem
While the Diffie-Helman Key Exchange protocol allows Alice and Bob to share secret keys without exposing their own private keys, it does not allow for the exchange of full messages (which is ultimately what we want from a cryptosystem).

However, in 1985, Elgamal described a new cryptosystem based on the DLP and the Diffie-Helman Key Exchange that finally addressed this issue.

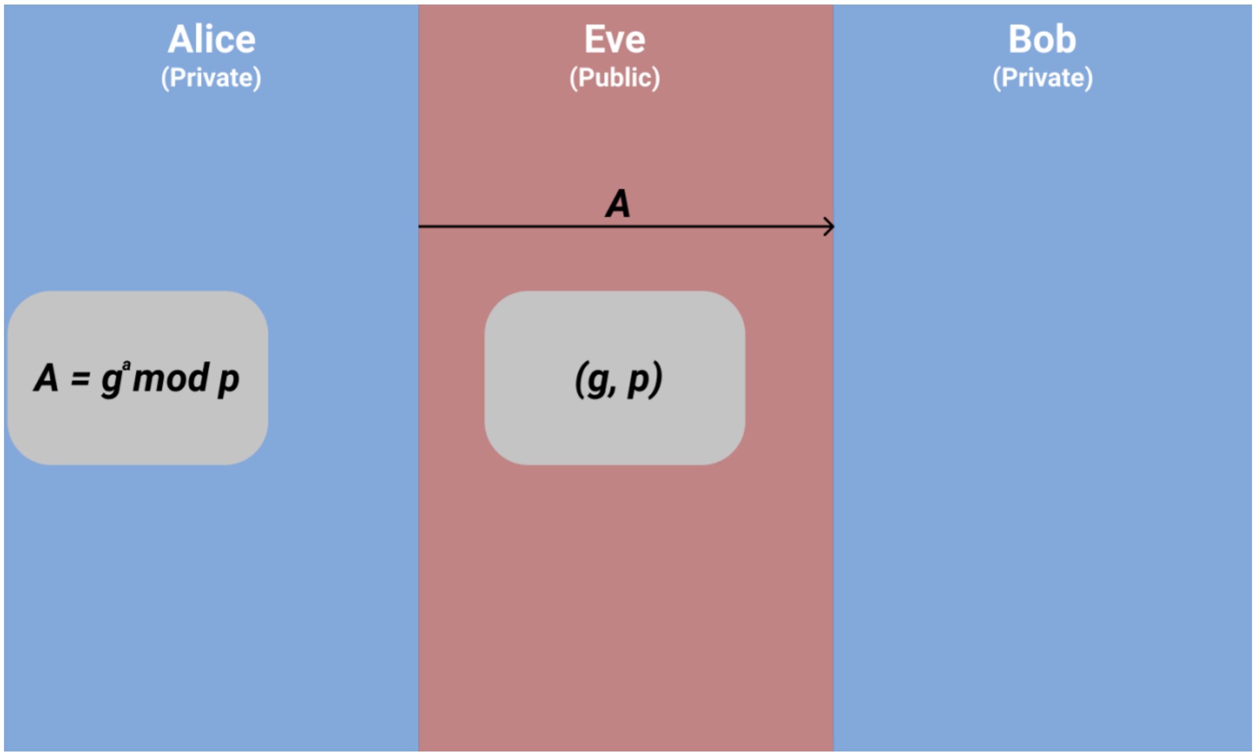
Let us now examine the setup of the Elgamal Public-Key Cryptosystem:

Creation of Public Parameters
1. Bob and Alice do not share a secret key through a private channel.
2. Alice or Bob chooses and publishes a (large) prime *p* and a primitive root *g mod p*. These two values are public and shared by Alice and Bob.



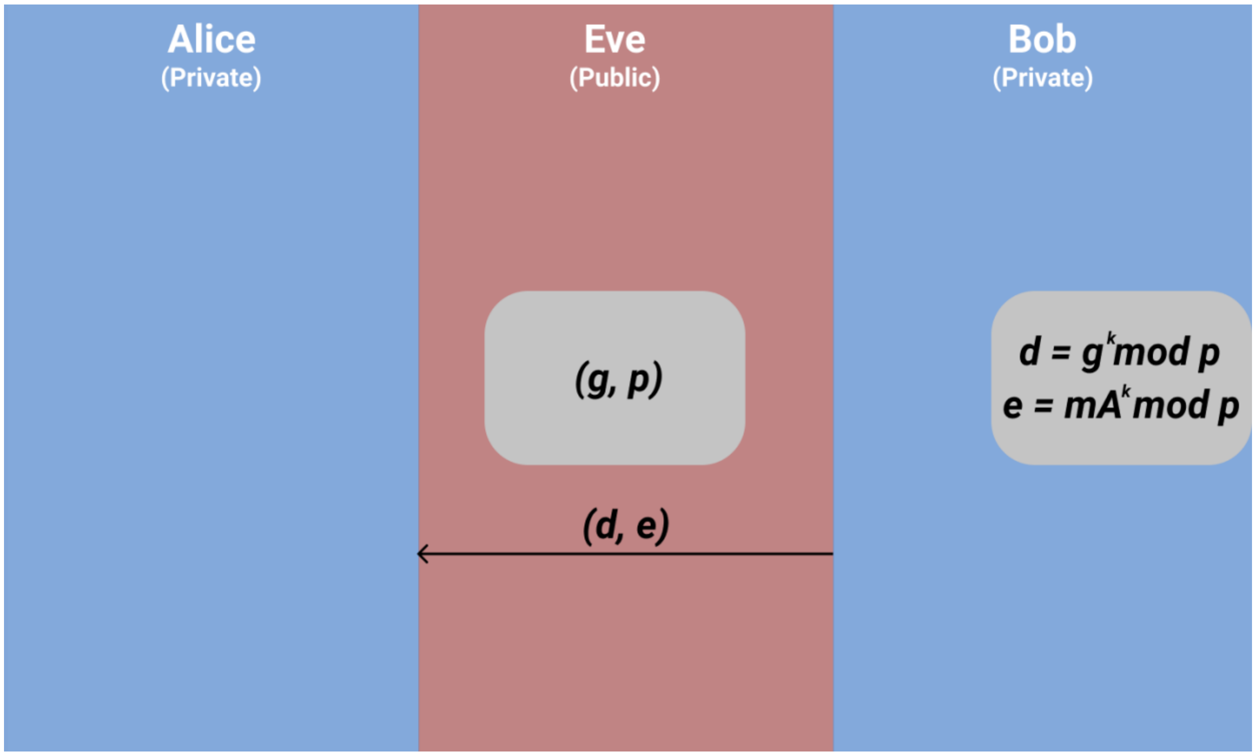
Alice: Key Creation

3. Alice chooses her own secret key a where a is greater than (or equal to) 0 but less than (or equal to) $p - 1$.
4. Alice computes $A = g^a \bmod p$.
5. Alice sends A to Bob.



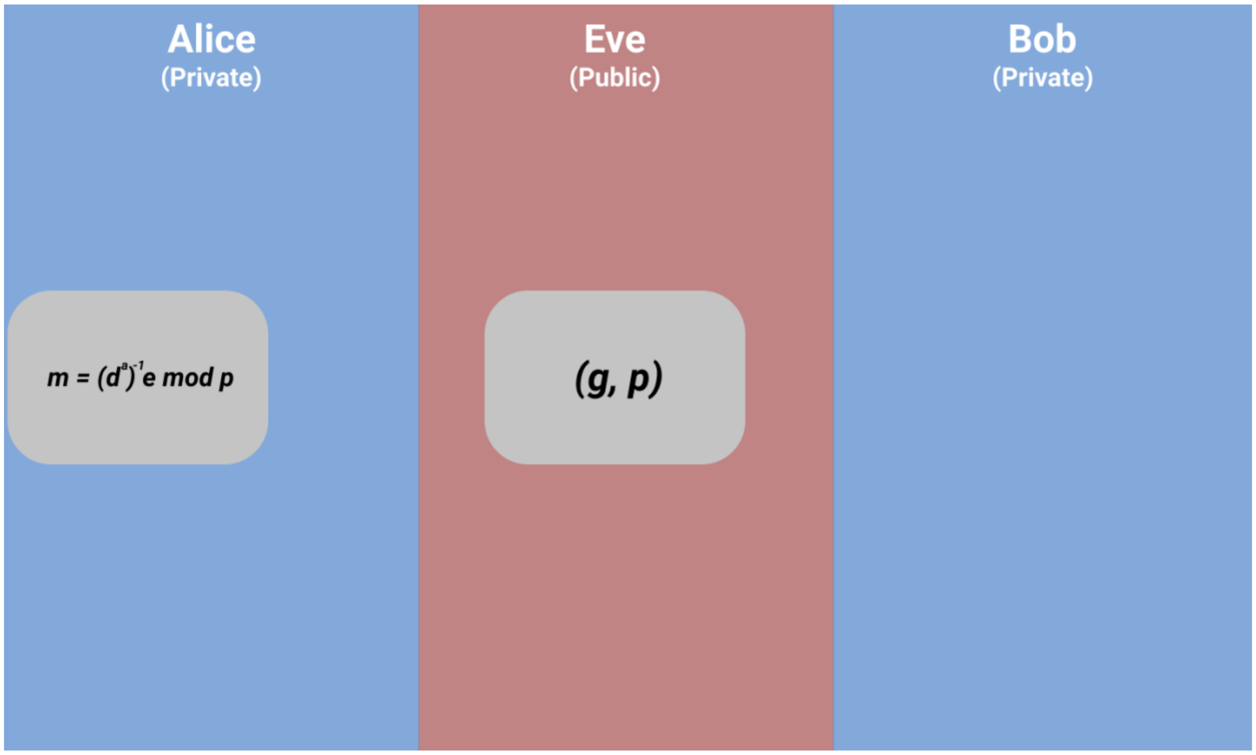
Bob: Encryption

6. Bob chooses a plaintext message *m* along with random integer *k*.
7. Bob computes $d = g^k \mod p$ and $e = mA^k \mod p$.
8. Bob sends *d* and *e* to Alice via a public channel.



Alice: Decryption

9. Alice computes $(d^a)^{-1}(e) \bmod p = m$.



Again, the security of this cryptosystem relies on the difficulty on solving the DLP. An attacker knows *p, g, A, d,* and *e*. While the security of the Elgamal Cryptosystem is more complex than Diffie-Helman, it is still secure given the appropriate selection of integers.

We will construct an (very insecure) example with small numbers to demonstrate the utility of the Elgamal Public-Key Cryptosystem:

Creation of Public Parameters

- Alice and Bob agree to use the fowling public parameters $(g, p) = (2, 13)$.

Alice: Key Creation

- Alice chooses her secret key $a = 3$ and computes $A = g^a \bmod p = 8 \bmod 13$.
- Alice sends A to Bob.

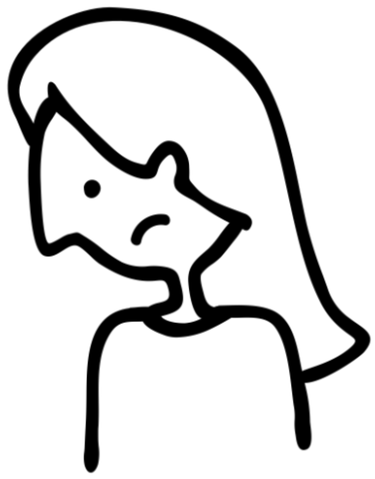
Bob: Encryption

- Bob selects plaintext $m = 10 \bmod 13$.
- Bob selects random integer $k = 6$.
- Bob computes $d = g^k \bmod p = 12 \bmod 13$.
- Bob computes $e = mA^k \bmod p = 3 \bmod 13$.
- Bob sends d and e to Alice.

Alice: Decryption

- Alice computes $(d^a)^{-1}(e) \bmod p = (12)(3) = 36 = 10 \bmod 13 = m$.

Chapter 3: Elliptic Curve Cryptography



I Have a Bad Feeling About This

Have you ever looked at a test and realized that you have no idea how to answer the first question?

Of course you have! You are a dummy, just like the rest of us!

Maybe you told yourself that the second question will be easier, only to find that you do not know how to answer that question either?

Or perhaps your manager assigns you an objective that you have no idea how to accomplish. When you start breaking down the objective into smaller tasks, you realize that you have no idea how to tackle those smaller tasks either.

With enough fortitude, support, and luck, we manage to successfully push through these tests and objectives (or we don't, and they haunt us for the rest of our lives). Not knowing how to even begin an assignment is an extremely challenging state of being for most of us; we do not know what background or context we need to understand in order to start tackling the assignment.

Understanding Elliptic Curve Cryptography is such an assignment. The mathematical subject of elliptic curves is both deep and dense; there are entire textbooks too heavy to carry around in a backpack that are devoted to this subject. Most undergraduate math majors lack the technical tools to thoroughly understand Elliptic Curve Cryptography. Naturally, this may leave you saying, "I have a bad feeling about this. How can I understand anything in this chapter if most math majors have a difficult time? I don't even enjoy math!".

Elliptic Curves

When teaching elliptic curves, there are two general approaches: 1) a geometric approach or 2) an algebraic approach. As the latter approach requires a much deeper mathematical toolset to handle, we will use the former approach. With this in mind, we will first review the concepts of intersections and tangents from pre-calculus and calculus, respectively. These two tools will allow us to create a geometric understanding of what constitutes an elliptic curve.

Intersections

Let us define the intersection of two curves:

Intersection (of two curves): Suppose you have two curves f and g. The intersection of f and g is the point (or points) where f = g.

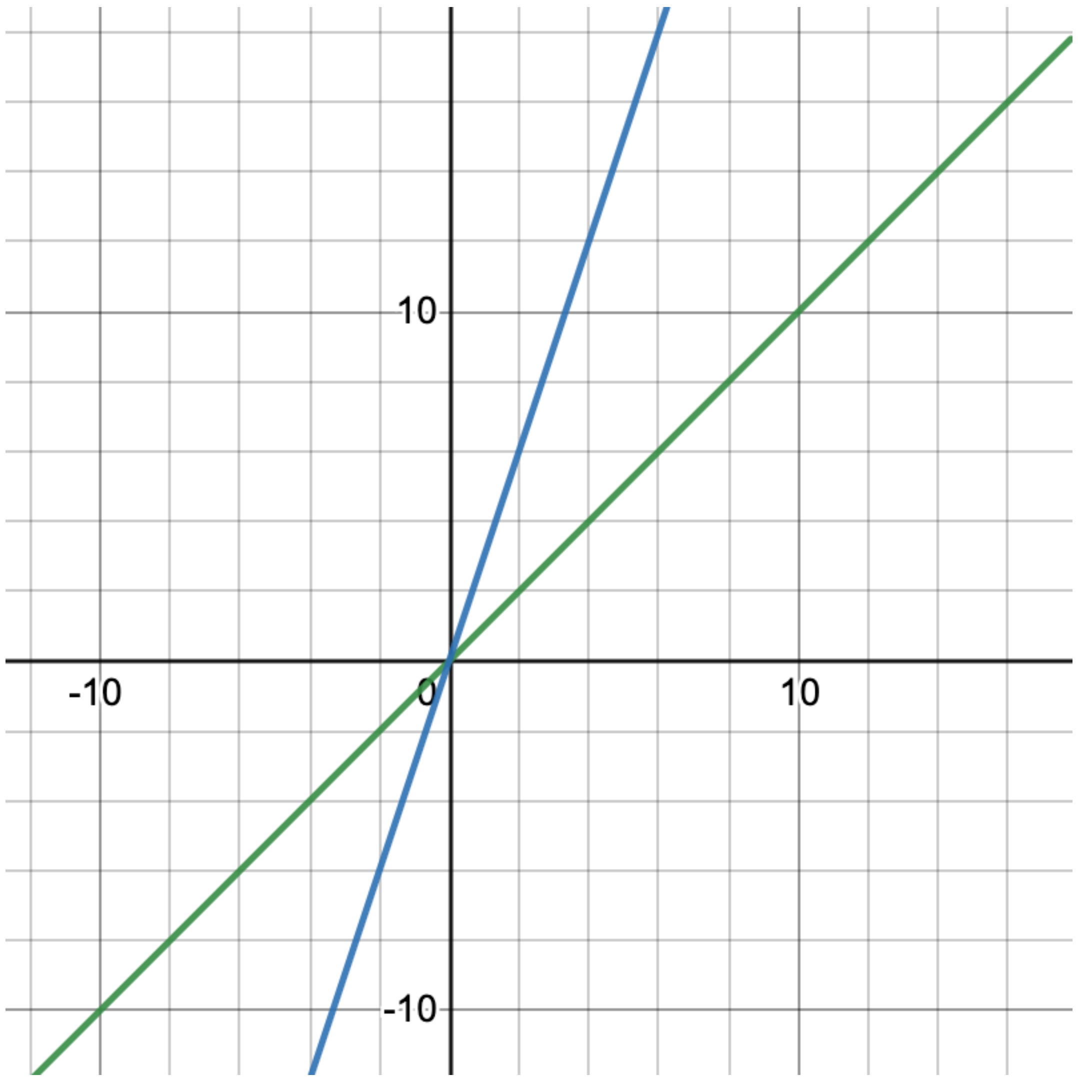
Suppose that you have two curves:

$$F: y = x$$
$$G: y = 3x$$

How do you find the intersection of *F* and *G*?

This is quite simple with such elementary curves: set F = G:

$$F = G$$
$$x = 3x$$



F (green), G (blue)

Clearly, *F = G* at *x = 0*.

Geometrically, the intersection of two curves is the point or points at which they cross over one another.

Suppose that you have two different curves:

$$H: y^2 = x^3-3x+3$$
$$I: y = 2x+3$$

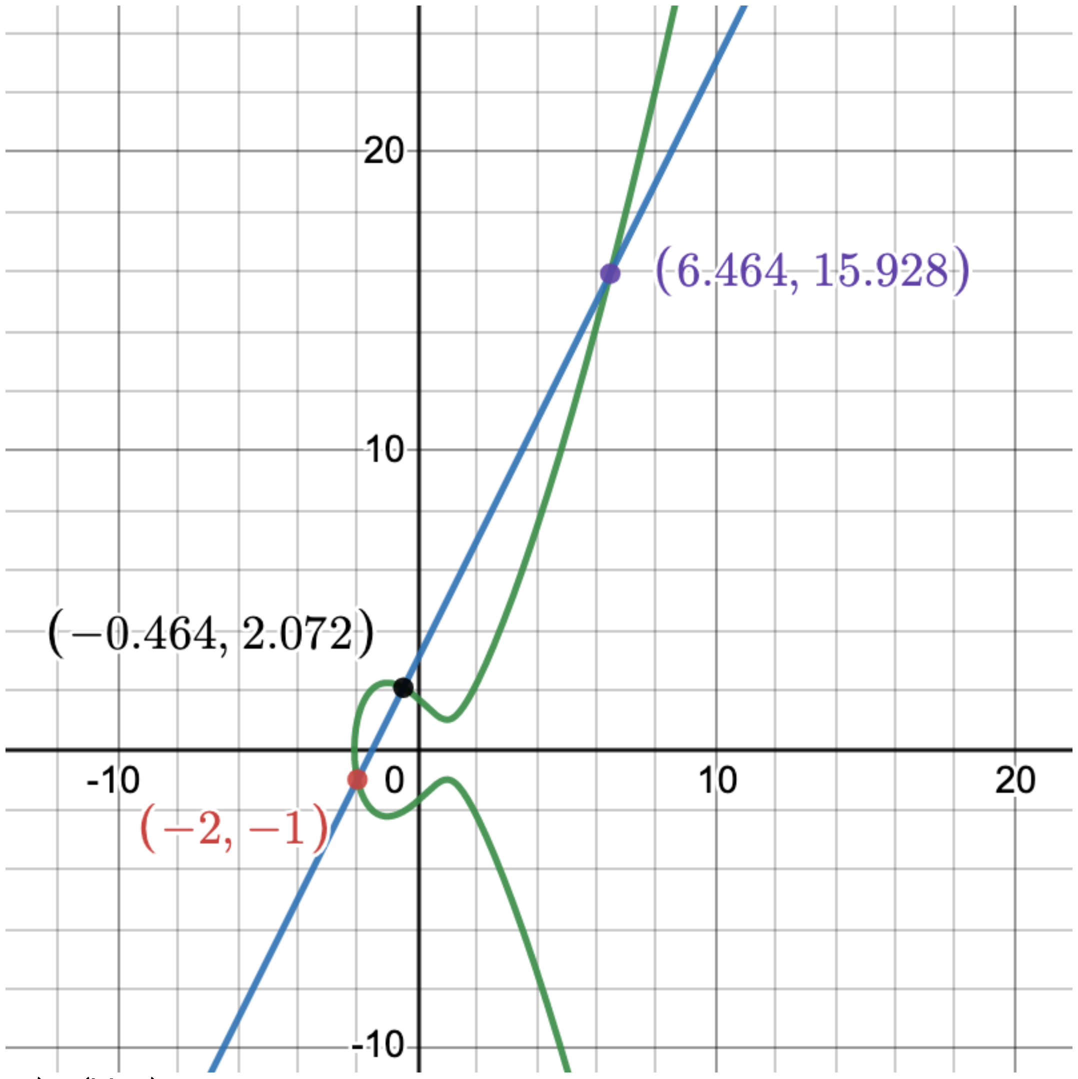
How do you find the intersection of *F* and *G*?

This is a little more complicated such curves: substitute I into H and solve for x:

$$(2x+3)^2=x^3-3x+3$$
$$4x^2+12x+9=x^3-3x+3$$
$$0 = -x^3+4x^2+15x+6=0$$

This equation has 3 solutions:

$x_1 = -2$
$x_2 = 3-2\sqrt{3}$
$x_3 = 3+2\sqrt{3}$



H (green), I (blue)

Abstracting this concept, finding the intersection of two curves is as simple as substituting one equation into the other. No matter how complicated the curves may look, this simple concept holds; please do not let the algebra scare you away.

Tangents

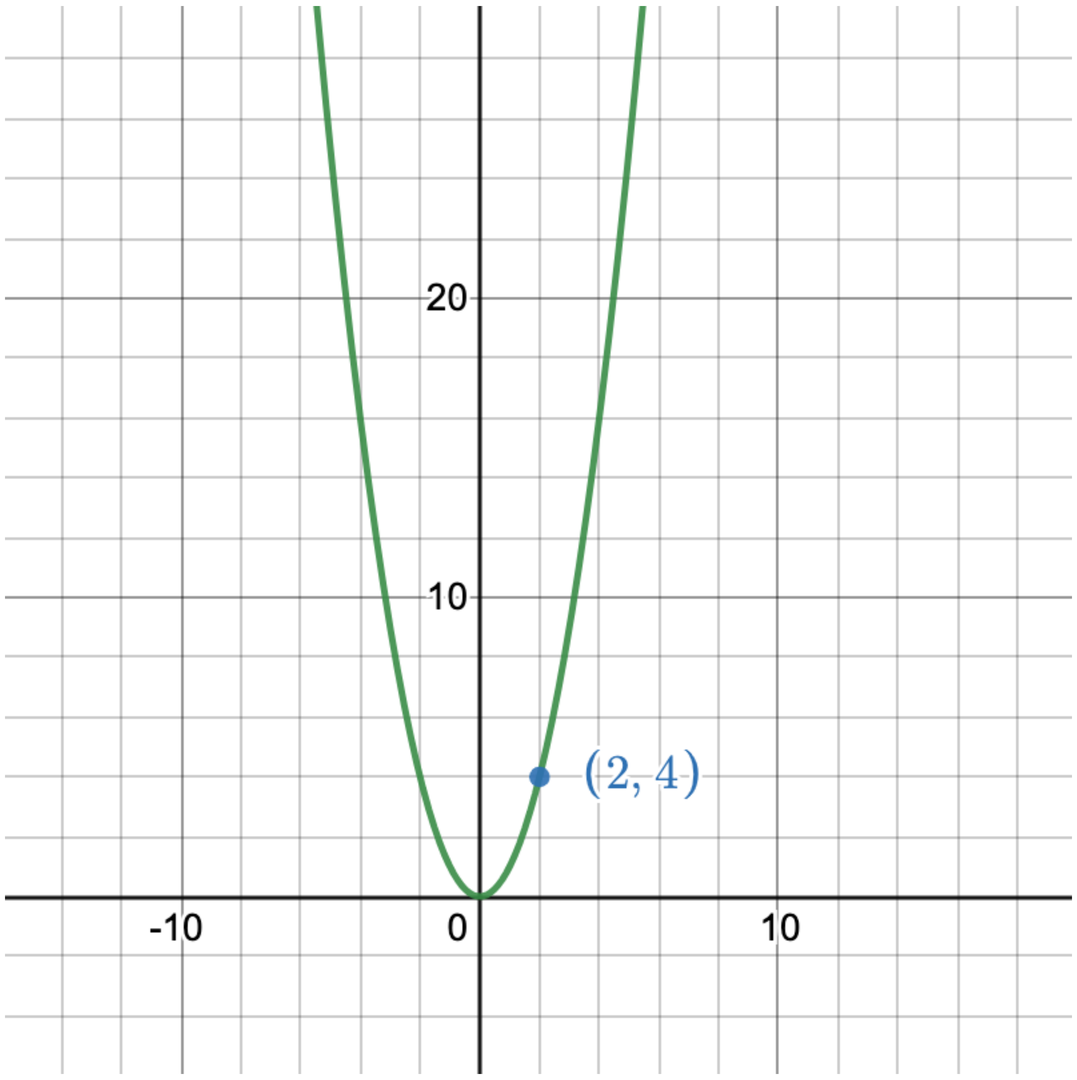
Let us define the tangent of a curve:

Tangent (of a curve): Suppose you have a curve f with a point (x,y) that is on f. The tangent of f at point (x,y) is the line that touches f once at exactly (x,y) and has the same slope (or derivative) at x.

Similar to the definition of intersection, the definition of tangent of a curve has a natural geometric interpretation which we will explore in the following example.

Suppose you want to find the tangent of a curve:

$$F: y = x^2 \text{ at } (x,y) = (2,4)$$



In order to find *T*, the tangent of *F* at point *(2,4)*, we must solve the equation:

$$T: 4 = 2m + b$$

which is nothing more than standard line notation ($y = mx + b$) that you learned in high school.

To solve this equation, we must first find *m*, the slope of the line at $x = 2$. We can find *m* from F by taking its derivative at $x = 2$ as the slope of *F* is equal to the slope of *T* at $x = 2$:

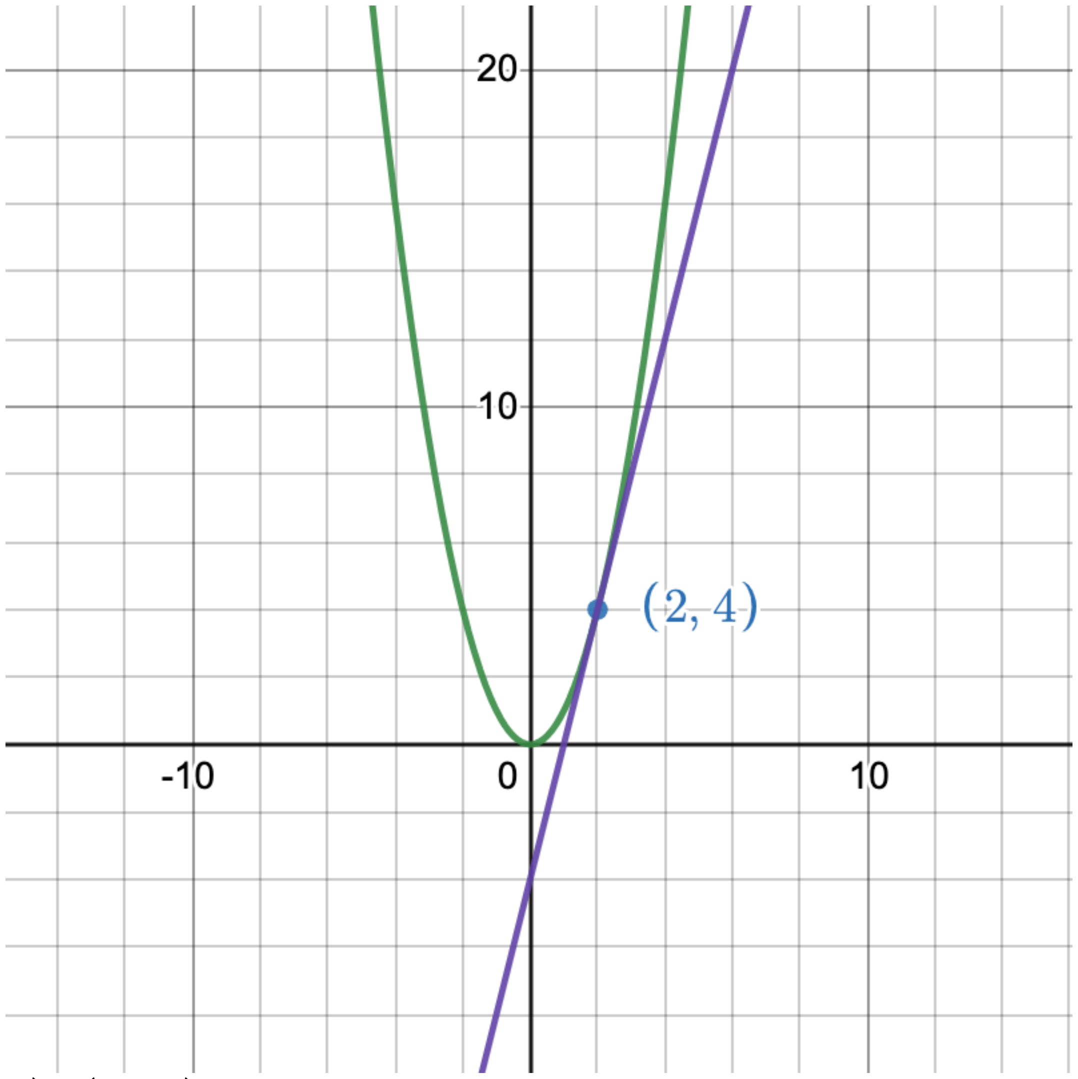
$$F' = 2x$$
$$m = F' \text{ at } x = 2$$
$$m = 4$$
$$T: 4 = 2(4) + b = 8 + b$$

Now that we have found the slope *m* of our line *T*, we must find *b* – the point at which *T* intersects with the *y-axis*. To solve for *b*, we can substitute *x = 0* into our equation for *T*:

$$T: 4 = 8 + b$$
$$T: -4 = b$$

Finally, we have determined *T*, the tangent line of curve *F* at point *(2,4)*:

$$T: y = 4x - 4$$



F (green), T (purple)

Elliptic Curves

Let us define an elliptic curve:

Elliptic Curve: An elliptic curve E set of solutions to the equation (called the Weierstrass equation):

$$E: y^2 = x^3 + Ax + B$$

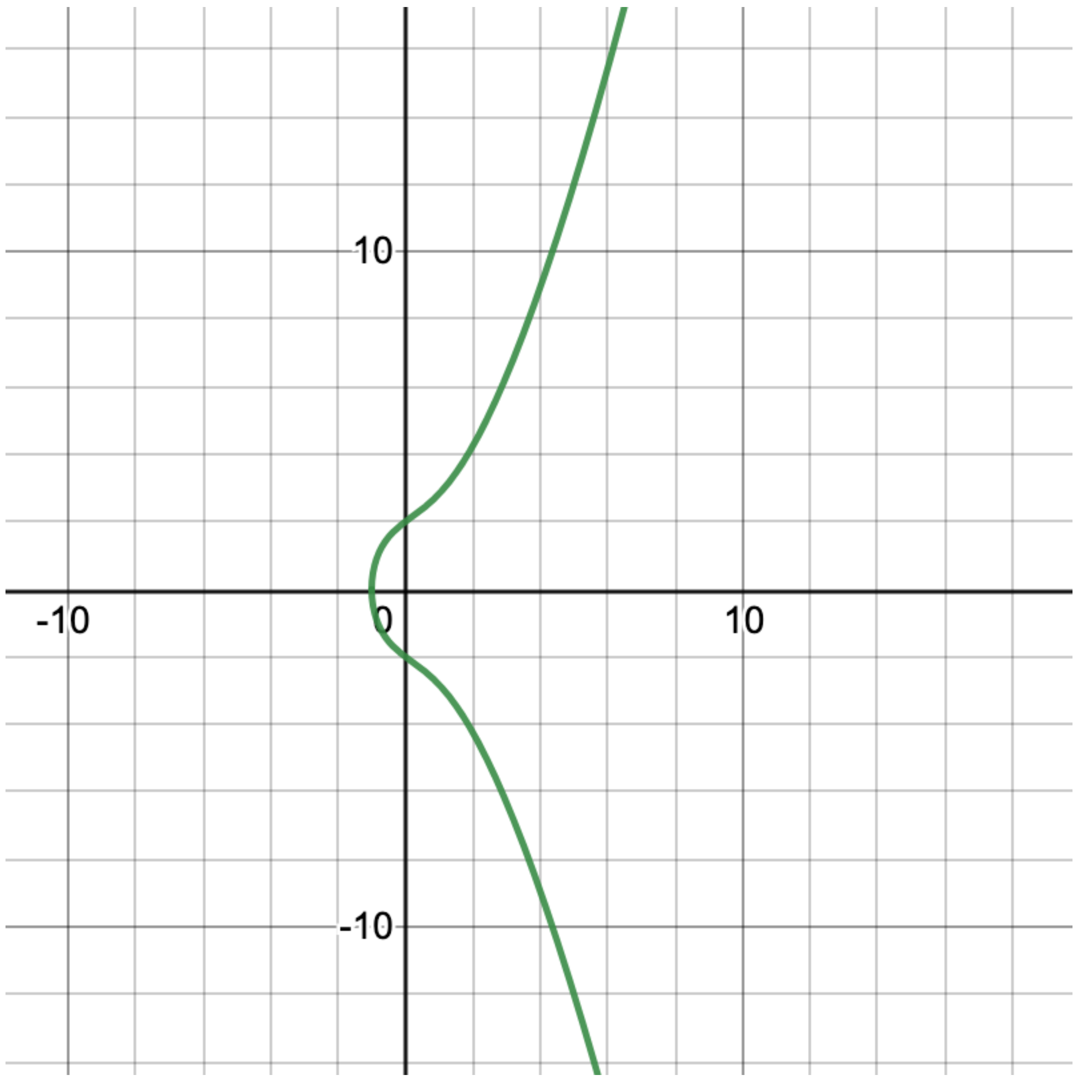
together with an extra point $\mathcal{O}$ at infinity where A and B are integers (positive or negative whole numbers) and satisfy the condition:

$$4A^3 + 27B^2 \neq 0$$

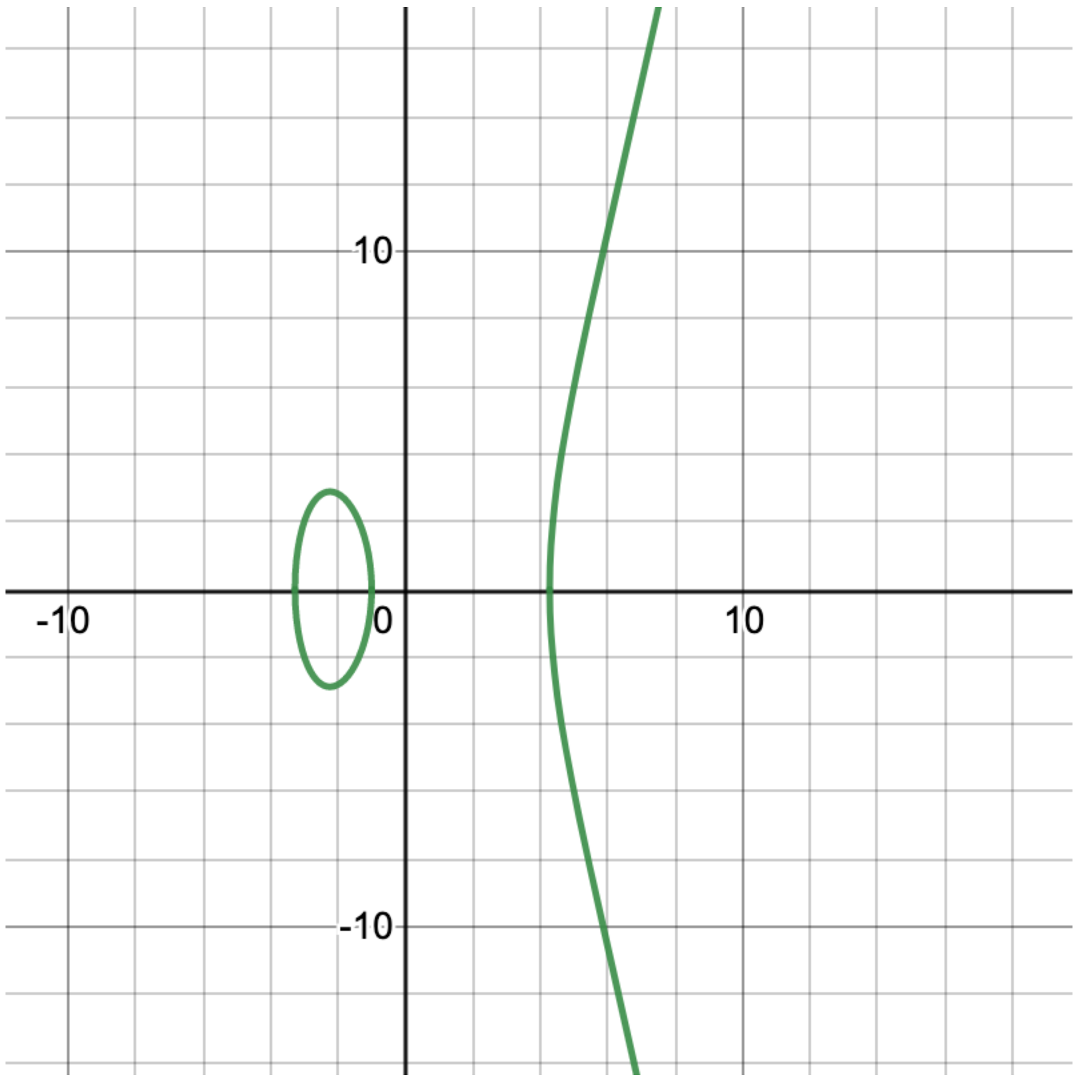
For those who are not thrilled by mathematics, please do not spend too much time ruminating on the “together with an extra point $\mathcal{O}$ at infinity” part of the definition. For all intents and purposes, $\mathcal{O}$ serves the same role as 0 in the integers – we will explore this in the next section. For the mathematically inclined, a geometric derivation of this definition can be found in Chapter 6.1 in *An Introduction to Mathematical Cryptography* by Hoffstein 2014.

What do elliptic curves look like? Let us look at two examples:

$y^2 = x^3 + 3x + 4$



$y^2 = x^3 - 15x - 14$

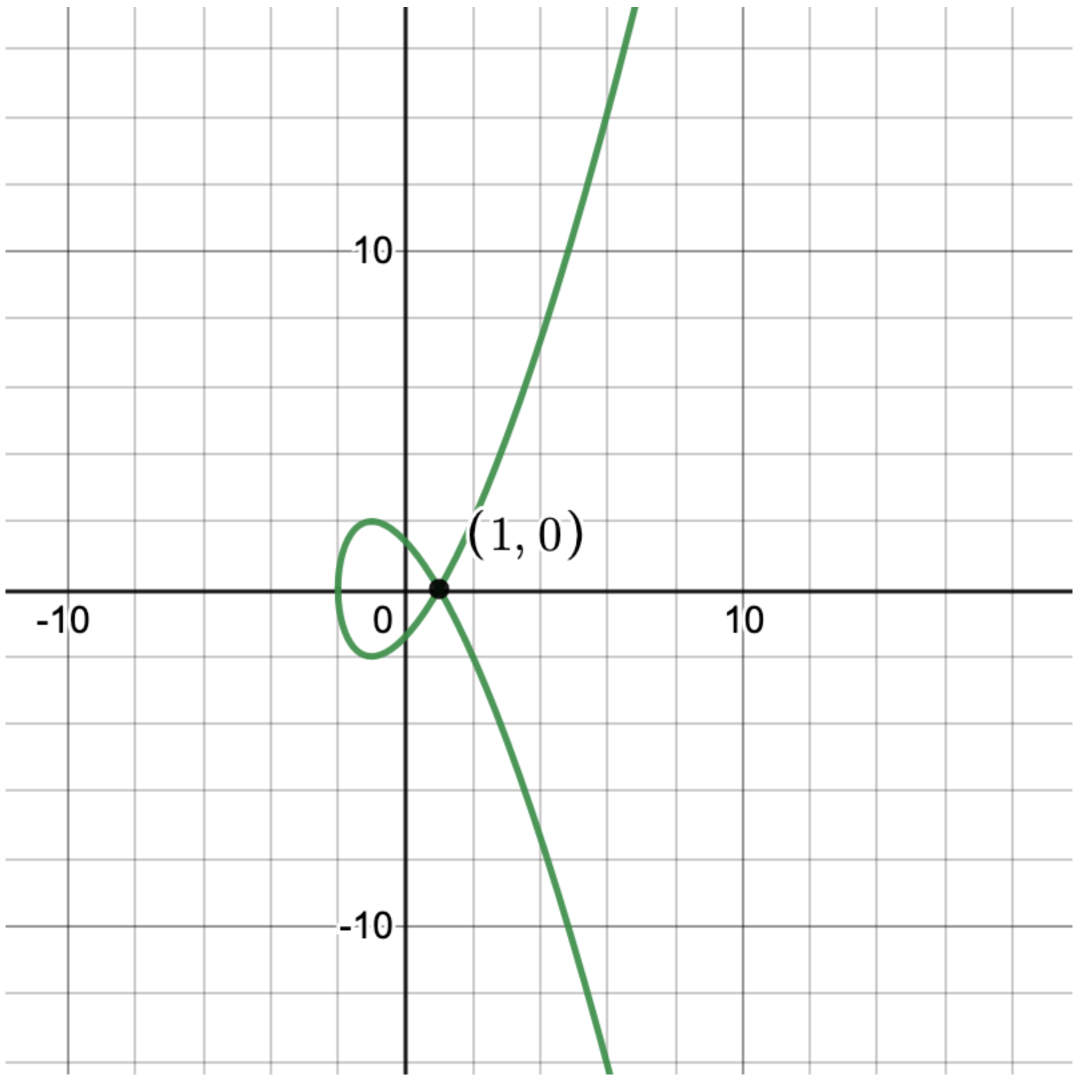


Both of these equations satisfy the condition:

$$4A^3 + 27B^2 \neq 0$$

Now let us look examine a curve that on the surface looks like an elliptic curve but does not satisfy our condition:

$y^2 = x^3 - 3x + 2$ where $4(-3)^3 + 27(2)^2 = 0$



Notice the strange point at *x = 1*?

At *x = 1* we see the curve intersect with itself. This point is called a *cusp* or a type of *singular point*. At such a point, the tangent of the curve (i.e., derivative) does not exist. If you try drawing a tangent line at a *cusp*, you will find that it is impossible for the tangent line to intersect the curve only once (which does not satisfy our definition of a *tangent*). This presents a lot of serious problems when we try to perform elliptic curve addition (next section); thus, we will avoid such curves at stick to our definition of elliptic curve.

Elliptic Curves over Finite Fields

Recalling the *set of Integers modulo n* from Chapter 2:

$$\mathbb{Z}_{\mathrm{n}} = 0,1,2, \dots, n-1$$

where there are *n* integers including 0. In simpler terms, the *set of Integers modulo n* represents a set of integers of size *n* (i.e., there are *n* elements in the set).

For example, the *set of Integers modulo 13* is as follows:

$$\mathbb{Z}_{13} = 0,1,2,3,4,5,6,7,8,9,10,11,12$$

For all intents and purposes in this guide, a finite field can be defined as follows:

Finite Field: The set of integers Modulo n. A finite field of size p prime is denoted as $\mathbb{F}_p$.

Typically, *n* will be replaced by a prime *p* in cryptographic applications.

An elliptic curve over a finite field can be defined as such:

Elliptic Curve over a Finite Field: An elliptic curve over a finite field, denoted $E(\mathbb{F}_p)$, is an elliptic curve whose elements only consist of points (x,y) that are in the finite field.

This definition may be confusing so let us examine an example:

Let $\mathbb{F}_{13} = 0,1,2,3,4,5,6,7,8,9,10,11,12$ be a finite field of size 13. Let E: $y^2 = x^3 + 3x + 8$. You can check that *E* is indeed an elliptic curve by checking whether or not it satisfies the condition (it does):

$$4A^3 + 27B^2 \neq 0$$

The elliptic curve over finite field $E(\mathbb{F}_{13})$ is the set of all points (i.e., *(x,y)*), where *x* and *y* are integers in $\mathbb{F}_{13}$, that lie on *E*.

To find $E(\mathbb{F}_{13})$, we can plug in different *x* values from $\mathbb{F}_{13}$ to find the two corresponding *y*-values using the equation $y^2 = x^3 + 3x + 8 \bmod 13$. Let's start with $x = 1$:

$$y^2 = (1)^3 + 3(1) + 8 = 12 \bmod 13$$

From the equation above, we know that $y^2 = 12 \bmod 13$. Since we are working modulo 13, we need to find two numbers that when multiplied by themselves (squared) equal *12 mod 13:*

$$5^2 = 25 = 12 \bmod 13,\ 8^2 = 64 = 12 \bmod 13$$

So, if $x = 1$, our two corresponding *y*-values are *5* and *8*. This leaves us with points *(1,5)* and *(1,8)* in $E(\mathbb{F}_{13})$.

However, not all *x*-values will have corresponding *y*-values that lie on $E(\mathbb{F}_{13})$. Take $x = 2$:

$$y^2 = (2)^3 + 3(2) + 8 = 4 \bmod 13$$

From the equation above, we know that y^2 = *4 mod 13*. Since we are working modulo 13, we need to find two numbers that when multiplied by themselves (squared) equal *4 mod 13*. However, no matter what numbers you try in $\mathbb{F}_{13}$, you cannot find two numbers that equal *4 mod 13* when they are squared. From this, we know that there are no points on *E*($\mathbb{F}_{13}$) with *x = 2*.

Continuing on, we can find all of the points in *E*($\mathbb{F}_{13}$):

***x*-Value**	***y*-Values**	**Points**
0	5, 8	(1,5), (1,8)
1	3, 10	(2,3), (2,10)
2	Not on *E*($\mathbb{F}_{13}$)	N/A
3	Not on *E*($\mathbb{F}_{13}$)	N/A
4	Not on *E*($\mathbb{F}_{13}$)	N/A
5	Not on *E*($\mathbb{F}_{13}$)	N/A
6	Not on *E*($\mathbb{F}_{13}$)	N/A
7	Not on *E*($\mathbb{F}_{13}$)	N/A
8	Not on *E*($\mathbb{F}_{13}$)	N/A
9	6, 7	(9,6), (9,7)
10	Not on *E*($\mathbb{F}_{13}$)	N/A
11	Not on *E*($\mathbb{F}_{13}$)	N/A
12	2, 11	(12,2), (12,11)

Not forgetting to add the extra point $\mathcal{O}$ at infinity, we can now describe the set *E*($\mathbb{F}_{13}$):

E$(\mathbb{F}_{13}) = \{\mathcal{O}, (1,5), (1,8), (2,3), (2,10), (9,6), (9,7), (12,2), (12,11)\}$

Elliptic Curve Addition

Integer Addition Law

Let us define "addition" on the integers:

Elliptic Curve Addition Law: Let Z be the set of integers and let x, y, and z be integers. The integer addition law has the following properties:

1. *Identity: x + 0 = 0 + x = x*
2. *Inverse: x + (-x) = 0*
3. *Associative: (x + y) + z = x+ (y + z)*
4. *Commutative: x + y = y + x*

This "addition law" is nothing more than a fancy (i.e., formal) way of expressing the regular addition you are used to in your everyday life.

Elliptic Curve Addition Law

Now, we will define the "addition law" on an elliptic curve:

Elliptic Curve Addition Law: Let E be an elliptic curve and let P, Q, and R be points on E. The elliptic curve addition law has the following properties:

1. *Identity: P + O = O + P = P.*
2. *Inverse: P + (-P) = O.*
3. *Associative: (P + Q) + R = P+ (Q + R).*
4. *Commutative: P + Q = Q + P.*

Notice the similarities between the two addition laws?

Elliptic Curve Addition Algorithm

Let us define an addition algorithm on an elliptic curve:

Elliptic Curve Addition Algorithm: Let E: $y^2 = x^3 + Ax + B$ be an elliptic curve and let P, Q, and R be points on E.

a) *Suppose P = O.* Then, P + Q = Q.
b) *Otherwise, suppose Q = O.* Then, P + Q = P.
c) *Otherwise, let P = (a,b) and Q = (c,d)*
d) *If a = c and b = -d, then P + Q = O*
e) *Otherwise, define λ such that:*
    a. *If P = Q, then* $\lambda = \frac{3a^2+A}{2b}$
    b. *If P$\neq$Q, then* $\lambda = \frac{d-b}{c-a}$

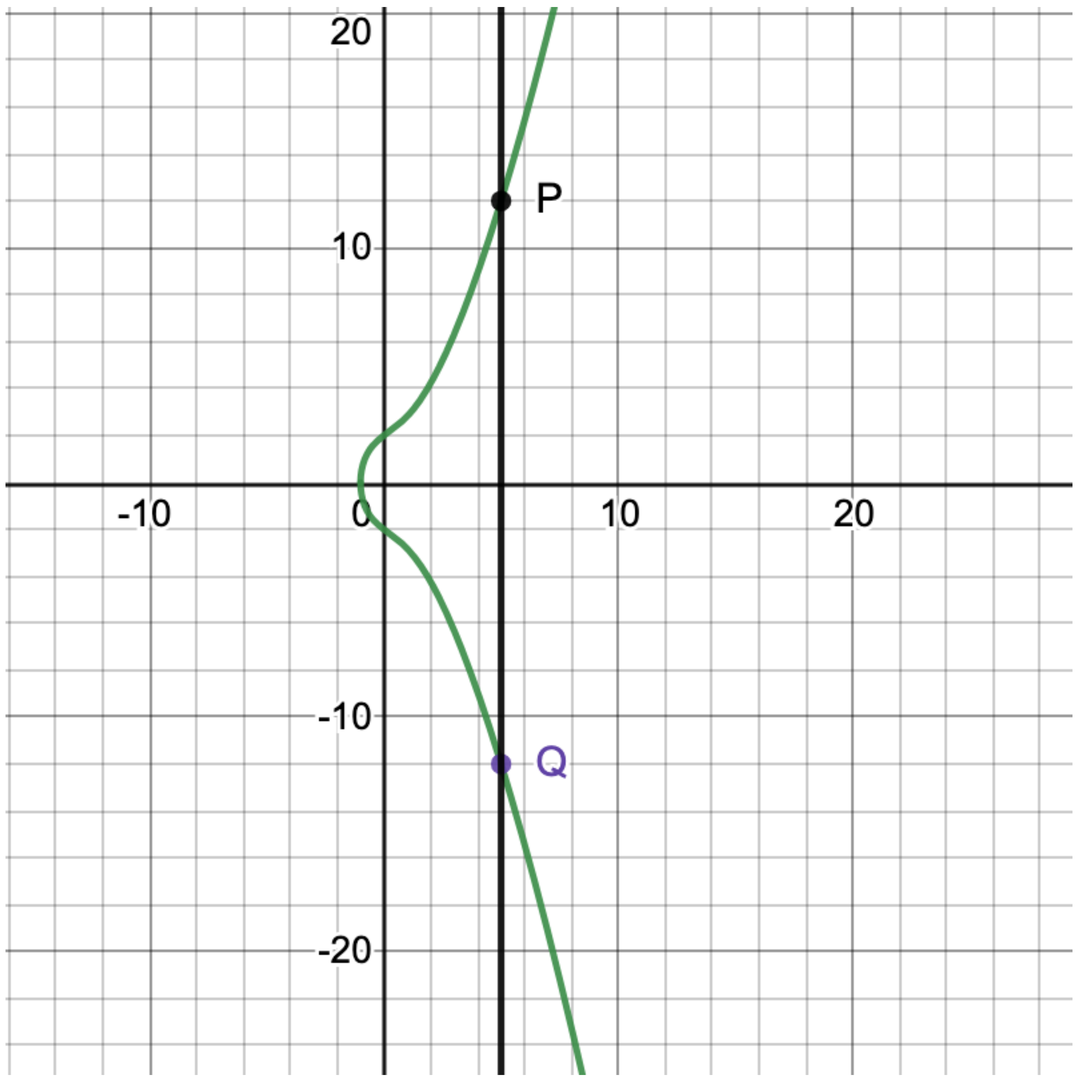
    Let e = λ^2 - a - c and f =λ (a - c) − b.
    Then, P + Q = (e, f).

This addition algorithm can be very confusing and scary as it is not similar to our idea of "regular" addition so let's draw some graphs to help develop a geometric understanding of what is going on.

a) *Suppose P = O.* Then, P + Q = Q.
b) *Otherwise, suppose Q = O.* Then, P + Q = P.

These two operations are intuitive as they are the same as "adding zero" in integer addition.

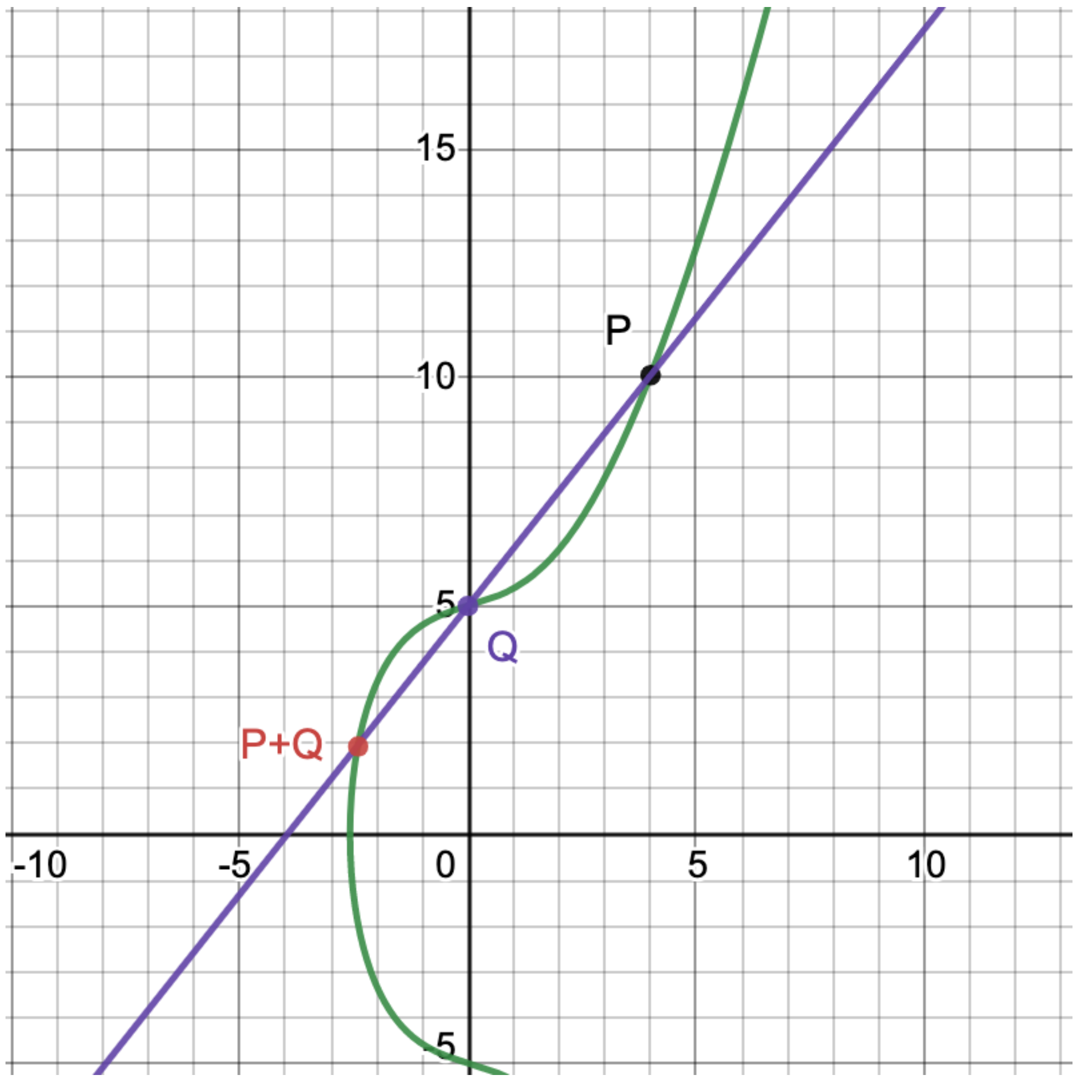
c) *Otherwise, let P = (a,b) and Q = (c,d)*
d) *If a = c and b = -d, then P + Q = O*



When we add such points, we get a vertical line which intersects with our point at infinity O.

e) *Otherwise, define λ such that:*
   a. *If P = Q, then* $\lambda = \frac{3a^2+A}{2b}$
   b. *If P$\neq$Q, then* $\lambda = \frac{d-b}{c-a}$

   Let e = λ^2 - a - c and f =λ (a - c) − b.
   Then, P + Q = (e, f).



Ignoring the algebra, you can get a sense of what is going on within the Elliptic Curve Addition Algorithm by looking at the geometry. To find *P+Q* with these conditions, we can draw a line that intersects both points and find the third point of intersection with the elliptic curve. This third point of intersection is *P+Q*.

Elliptic Curve Discrete Log Problem

Discrete Log Problem

From the previous chapter, we know the following definition:

Discrete Log Problem: Let x be a primitive root of a prime p. Let z be a positive integer modulo p. The Discrete Logarithm Problem (DLP) is the mathematical problem of finding an integer y such that:

$$x^{y} = z \bmod p$$

Elliptic Curve Discrete Log Problem

There is an equivalent problem to the DLP on elliptic curves:

Elliptic Curve Discrete Log Problem: Let E be an elliptic curve and let P, and Q be points on E. The Elliptic Curve Discrete Logarithm Problem (ECDLP) is the mathematical problem of finding an integer n such that:

$$nP = Q$$

That is, the ECDLP is the problem of finding an integer *n* such that *n* additions of *P* is equal to *Q*:

$$Q = P + P + P + \ldots + P = nP$$

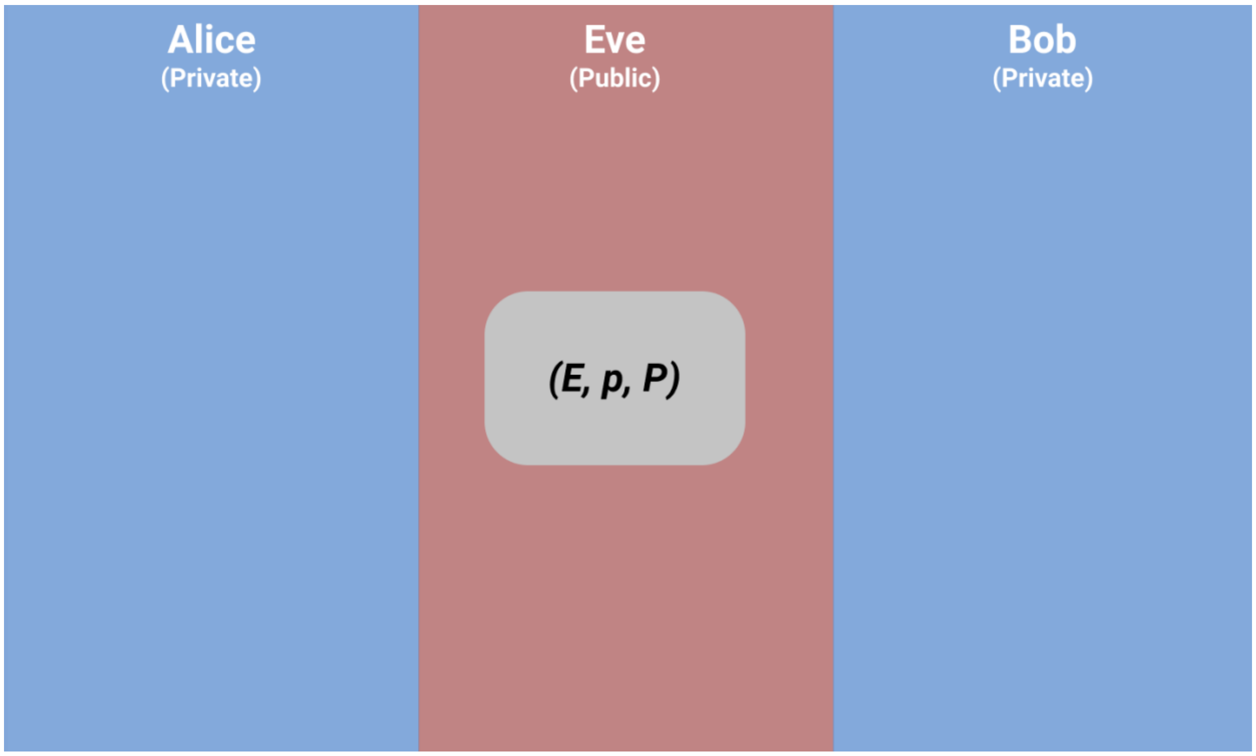
Elliptic Curve Cryptography

Elliptic Curve Diffie-Helman Key Exchange

Well, let us examine the setup of the Diffie-Helman Key Exchange:

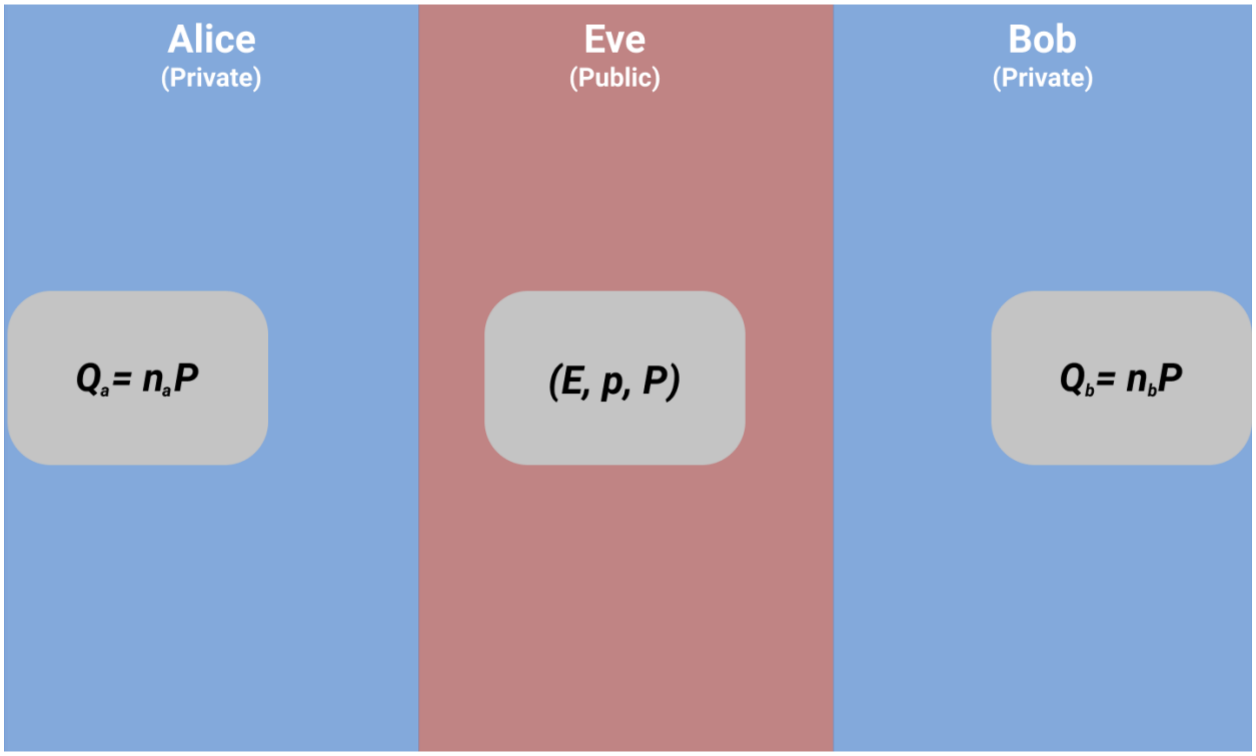
Creation of Public Parameters

12. Bob and Alice do not share a secret key through a private channel.
13. Alice or Bob chooses and publishes a (large) prime *p*, an elliptic curve over a finite field $E(\mathbb{F}_p)$, and a point *P* in $E(\mathbb{F}_p)$. These values are public and shared by Alice and Bob.



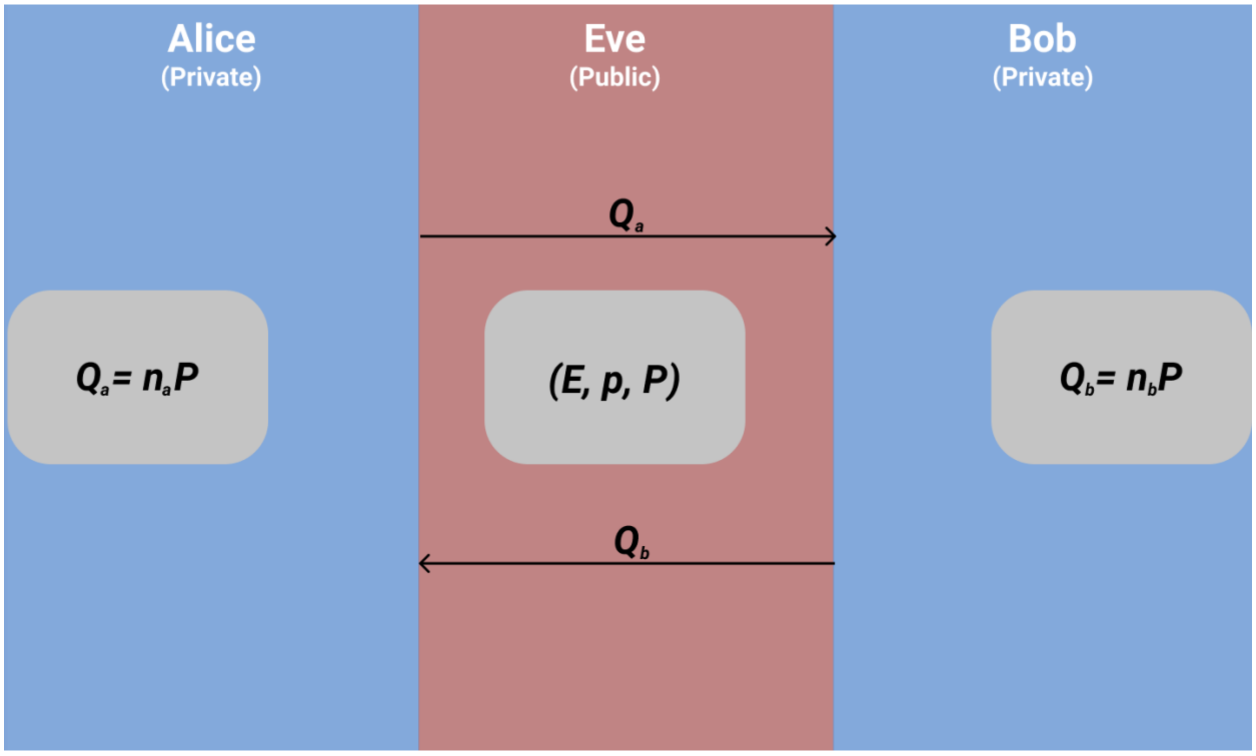
First Private Computations

14. Alice chooses her own secret integer n_a.
15. Bob chooses his own secret integer n_b.
16. Alice computes $Q_a = n_aP$.
17. Bob computes $Q_b = n_bP$.



Public Exchange

18. Alice sends Q_a to Bob via a public channel.
19. Bob sends Q_b to Alice via a public channel.



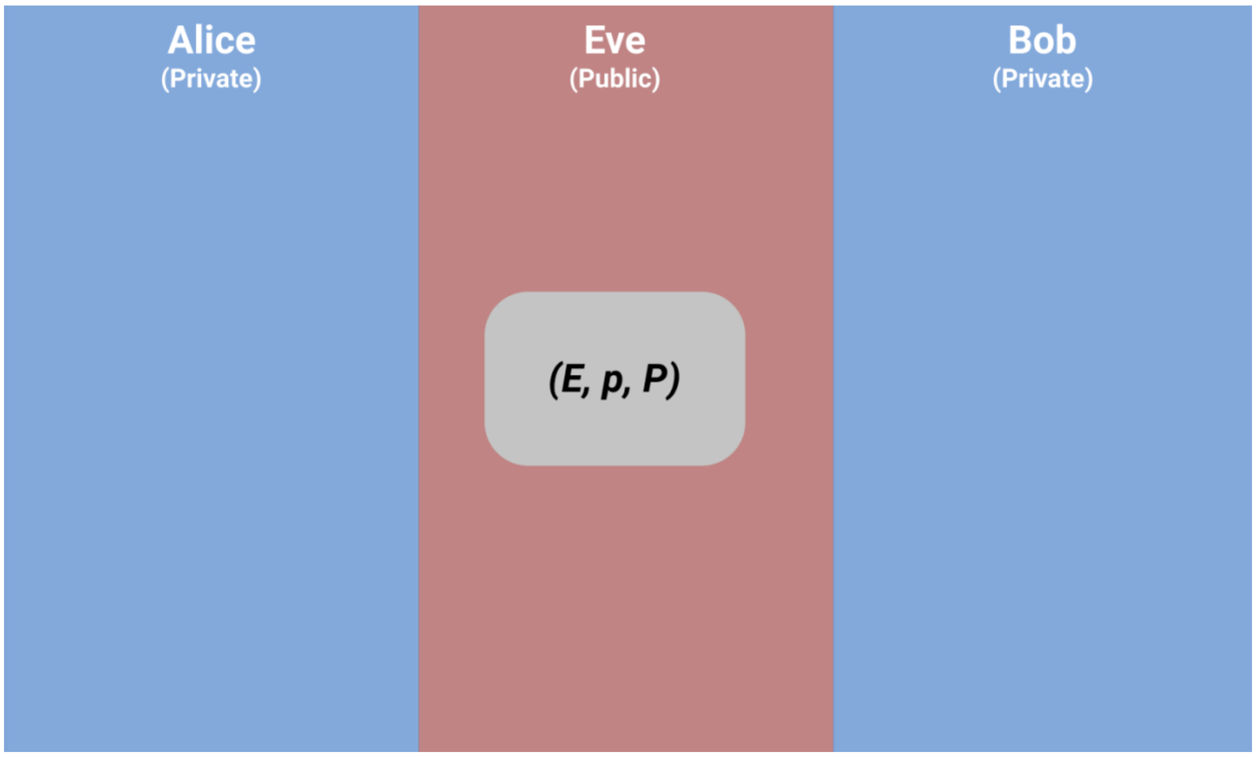
Second Private Computations

20. Alice computes n_aQ_b.
21. Bob computes n_bQ_a.
22. By the properties of elliptic curve addition, $C = n_aQ_b = n_bQ$. So, C is their shared secret key that they can use in further computations without exposing n_a or n_b.

Elliptic Curve Elgamal Public Key Cryptosystem

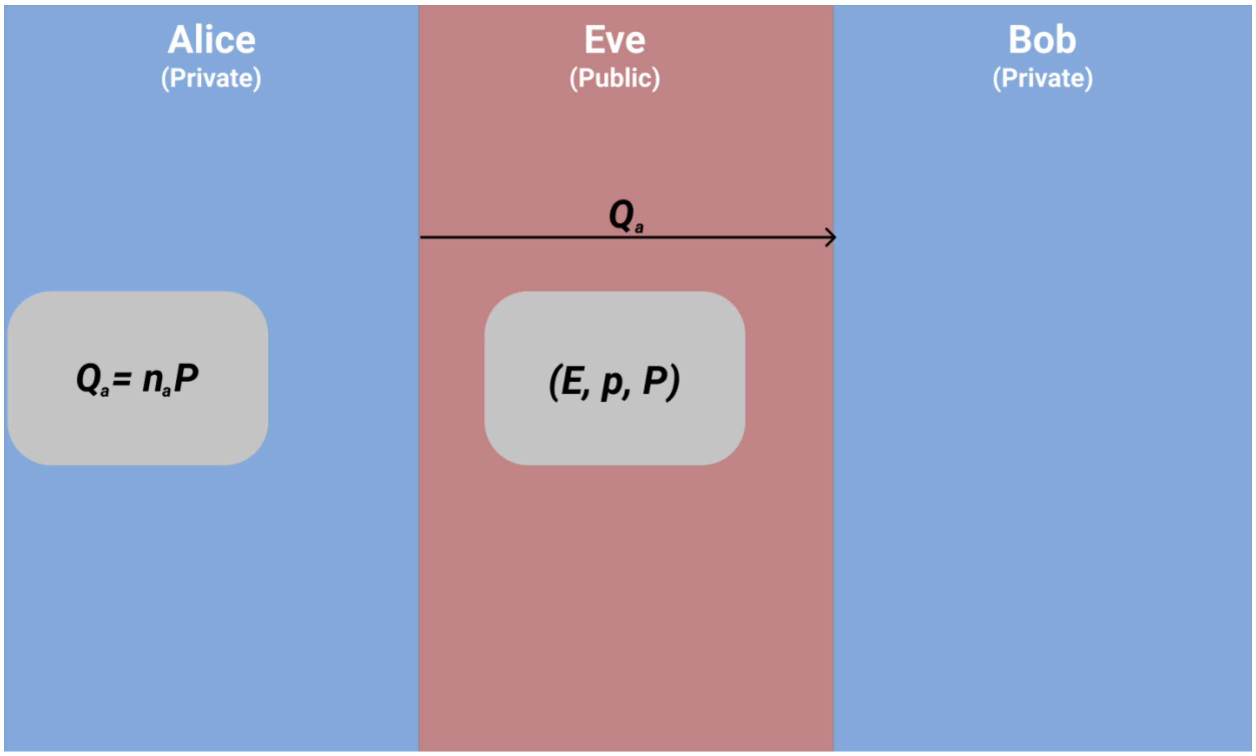
Creation of Public Parameters

1. Bob and Alice do not share a secret key through a private channel.
2. Alice or Bob chooses and publishes a (large) prime p, an elliptic curve over a finite field $E(\mathbb{F}_p)$, and a point P in $E(\mathbb{F}_p)$. These values are public and shared by Alice and Bob.



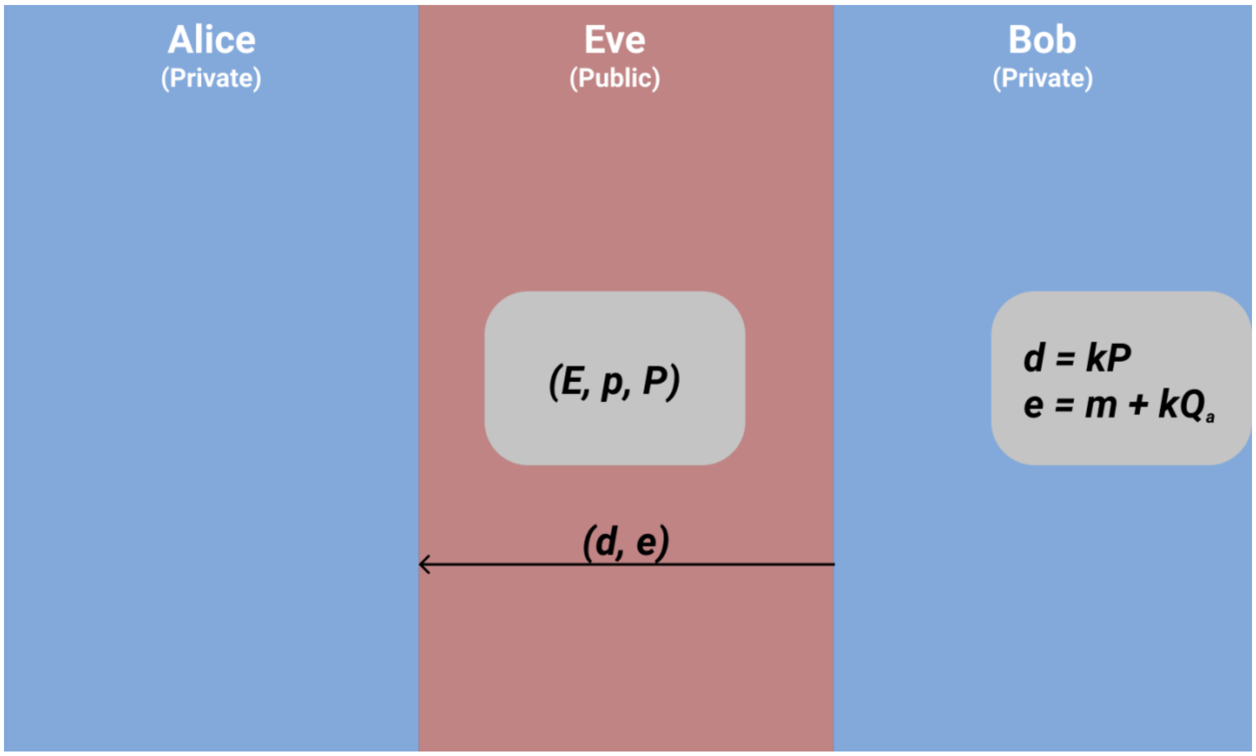
Alice: Key Creation

10. Alice chooses her own secret integer n_a.
11. Alice computes $Q_a = n_aP$.
12. Alice sends Q_a to Bob.



Bob: Encryption

13. Bob chooses a plaintext message m (which is an element in $E(\mathbb{F}_p)$) along with random integer k.
14. Bob computes $d = kP$ and $e = m + kQ_a$.
15. Bob sends d and e to Alice via a public channel.



Alice: Decryption

16. Alice computes $e - n_a d = m$.

Chapter 4: Hashing and Digital Signatures

Downfall of Electronic Signatures

Suppose you were asked the following questions:

1. What is an electronic signature?
2. What is a digital signature?
3. Are electronic signatures the same as digital signatures?

Maybe you think that electronic and digital signatures are the same, so your answers look something like this:

1. When you electronically sign a document with your signature, duh!
2. The same as above.
3. Yes.

We interact with electronic and digital signatures every day yet many of us do not know that these terms are not interchangeable.

An electronic signature is when you type, write, or upload a photo of your analog (i.e., paper) signature onto a digital document. For example, John Smith needs to fill in and sign an electronic document (e.g., like a media release form); he completes the document and signs with the following:

John Smith

Electronic signatures have a major problem: there is no way to verify that the document was not changed or tampered with after John Smith submitted the document.

Additionally, anyone that has a copy of John Smith's signature can now sign documents on his behalf without his knowledge. The consequences of someone tampering with or signing an electronic document without the consent of John Smith may be innocuous for something like a media release form for a small charity gathering; however, they can be devastating for contracts that involve more serious legal and financial consequences.

Digital signatures solve both problems. Closer to a notary stamp, digital signatures involve creating a digital certificate through a cryptographic signature scheme. If John Smith signs with a digital signature, both he and the receiver can verify whether a document was changed or tampered with after he submitted it. Additionally, no one can recreate John Smith's digital signature without important technical information.

With the digitization of many business functionalities, electronic documents provide massive amounts of creation, storage, and transportation efficiency gains when compared to analog documents. Very few offices still use fax machines over email. However, a problem with digitization is the inherit lack of scarcity in a digital world. As stated previously, a nefarious party can just illegally tamper with or sign on behalf of John Smith without his knowledge. Worse, there is no way to prove that John Smith did not make changes to or did not create a document if you have his electronic signature. It is a problem that would have to be disputed in court.

However, digital signatures allow the recipient of the signed document to verify the integrity of the document via software. With the appropriate identity verification methods in place, both John Smith and the recipient can rest assured that the electronic document has not been tampered with or created by an adversarial third party.

The utility and security of digital signatures has led to the downfall of electronic signatures. No one would use the Bitcoin protocol if it used electronic signatures instead of digital signatures. Similarly, there are many digital applications that are only feasible with the use of digital signatures.

For more information on the difference between electronic and digital signatures, see The Importance of Having a Secure Digital Signature Platform.

Hashing

Strings and Documents

In computer programming, every number, letter, and punctation mark is a character. For example, "1", "one", and "!" are all characters. Furthermore, there are many characters that are invisible; some examples include the *space* and *tab* characters. Taking these characters together, we can create a new type of data called a *string*:

String: A data type which is an ordered sequence of characters.

Strings are extremely common in most programming languages. A string can contain any type of character, visible or invisible, with repetitions. An important characteristic of strings is their length: “Hi!” is a string of length 3 while “” or the empty string has a string of length 0.

Together with other data types, strings make up a large portion of digital *documents:*

Document: Any type of digital document. Some examples include a computer file, photo, or video.

Documents are extremely useful in everyday life. However, some documents can be quite large and impractical to use within cryptographic applications. It would be useful to have a method of taking documents of arbitrary size and turning them into fixed-size values. This is where the notion of hashing comes into play.

Hashing

Hash Mapping ($h: D \rightarrow H$): Let D be a document of arbitrary size and H be a short bit string. A Hash Mapping $h: D \rightarrow H$ takes an input D and returns an output H.

A hash mapping should have the following properties:

1) It is deterministic – hashing the same *D* results in the same *H*.
2) It is very difficult to find two different *D*’s that result in the same *H*.
3) A small change in *D* results in a large change in *H* such that the new *H* seems uncorrelated to the old *H*.
4) It is quick to compute for any *D*.
5) It is a one-way mapping – it is easy to convert *D* to *H* but very difficult to convert *H* to *D*.

These 5 properties can be summarized in the following sentence: hashing maintains the integrity of a document. What does this mean? This means that hashing is a great tool to build cryptographic systems where one needs to verify whether a document is belongs to a specific individual or has been altered. Hashing is exactly the tool we need to begin our discussion of digital signatures.

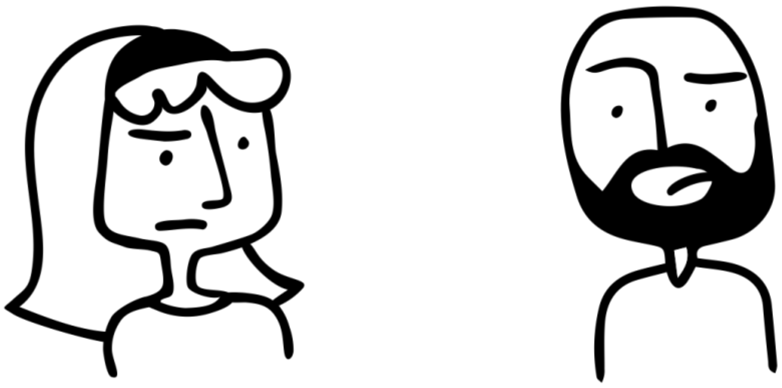
Digital Signatures

Digital Signatures

The primary objective of cryptographic schemes is to exchange communication over an adversarial or insecure network.

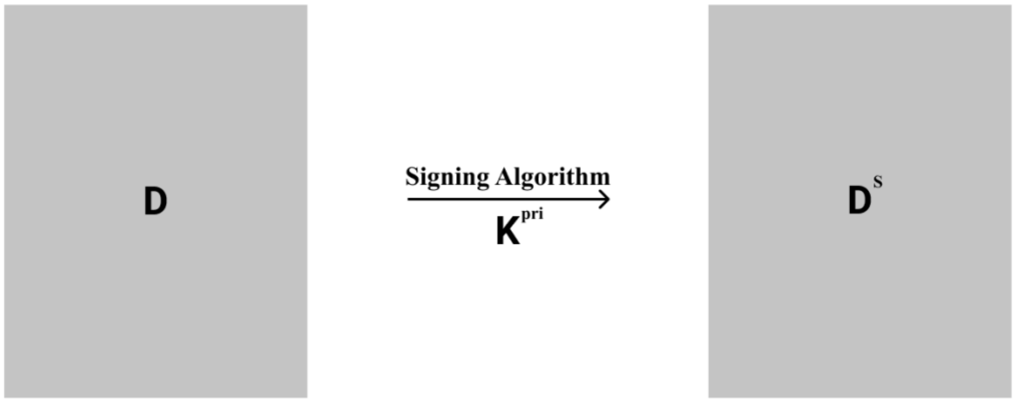
The primary objective of digital signatures is to solve the problem of digital ownership or agreement.

Suppose Samantha wants to electronically sign a legal document and securely send it to Victor, her lawyer. She has two problems: Problem 1) How can she “sign” the electronic document so that Victor can verify that it is indeed her document? Problem 2) How can she securely send the signed document to Victor?

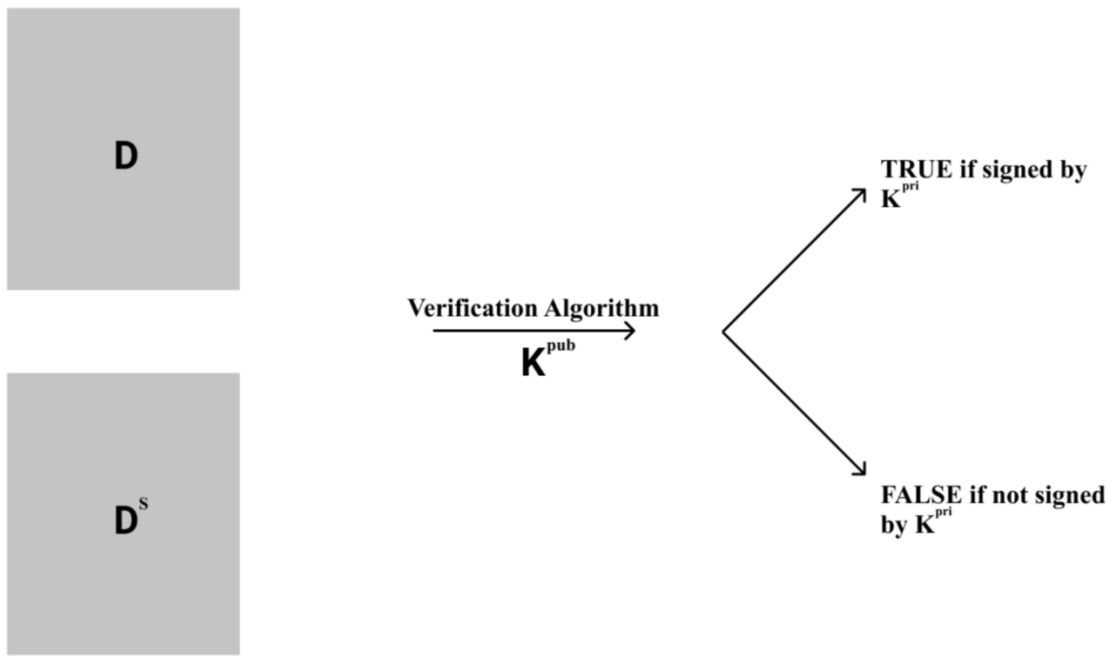


Problem 2 is solved by cryptographic schemes such as the Elgamal Cryptosystem. Let us focus on Problem 1:

Let *D* be the unsigned document that Samantha wants to sign and send to Victor. Let D^S be the signed document – the original legal document *D* with some additional information that constitutes the digital signature. Here is the signing process:



Once Samantha sends both the unsigned and signed documents to Victor, he must have some way to verify whether or not the Samantha approves or signed the document. To do this, he must use a verification algorithm that takes K^{pub}, *D,* and D^S and returns one of two outcomes: 1) if D^S is the signed version of *D* with private key K^{pri}, the algorithm will return TRUE or 2) if D^S is the not signed version of *D* with private key K^{pri}, the algorithm will return FALSE. Since the verification algorithm does not have direct access to Samantha's private key, the digital signature scheme must have some way to easily verify a digital signature using the public key.

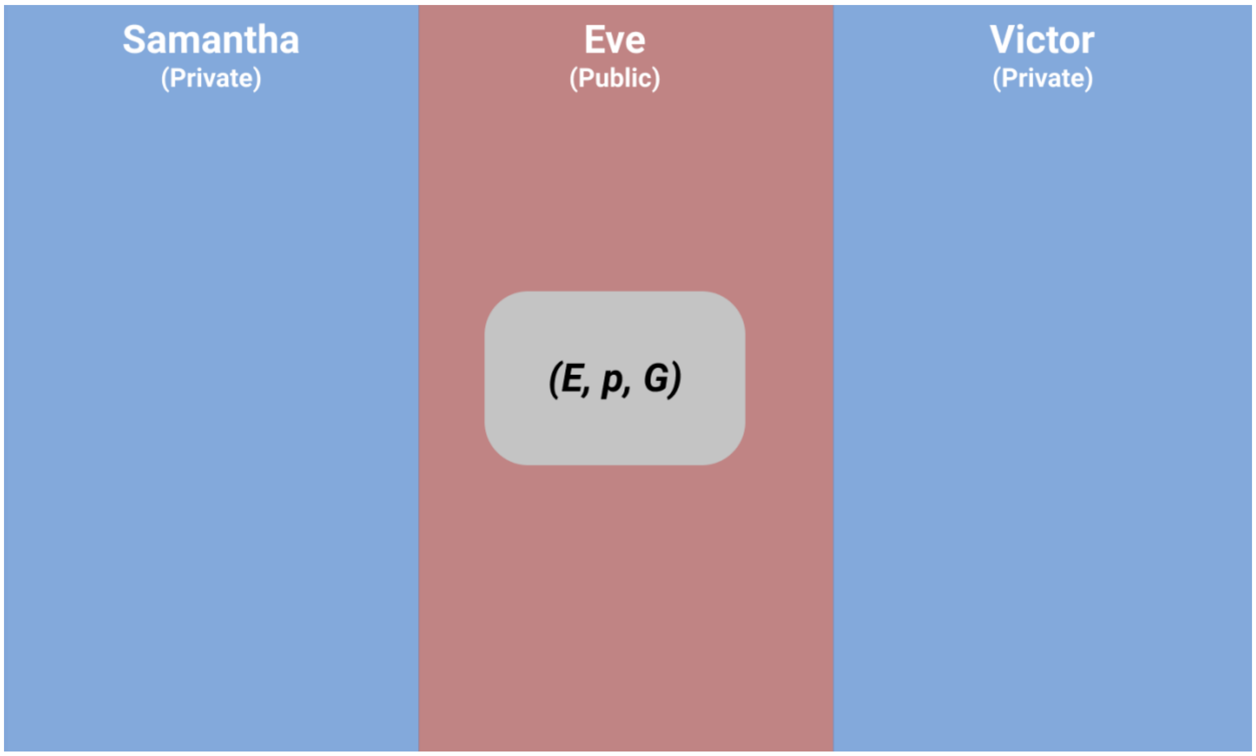


Digital Signatures on Elliptic Curves

Elliptic Curve Digital Signature Algorithm

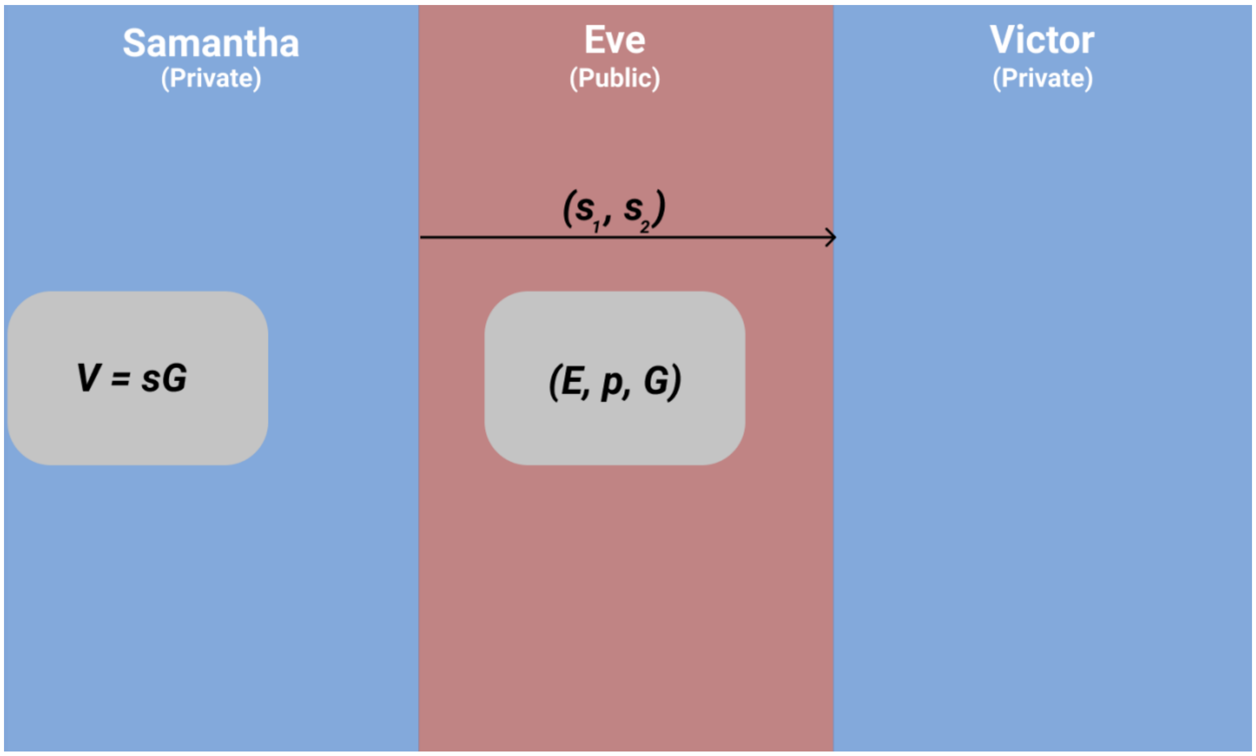
Creation of Public Parameters

3. Samantha and Victor do not share a secret key through a private channel.
4. Samantha or Victor chooses and publishes a (large) prime p of order q, an elliptic curve over a finite field $E(\mathbb{F}_p)$, and a point G in $E(\mathbb{F}_p)$. These values are public and shared by Alice and Bob.



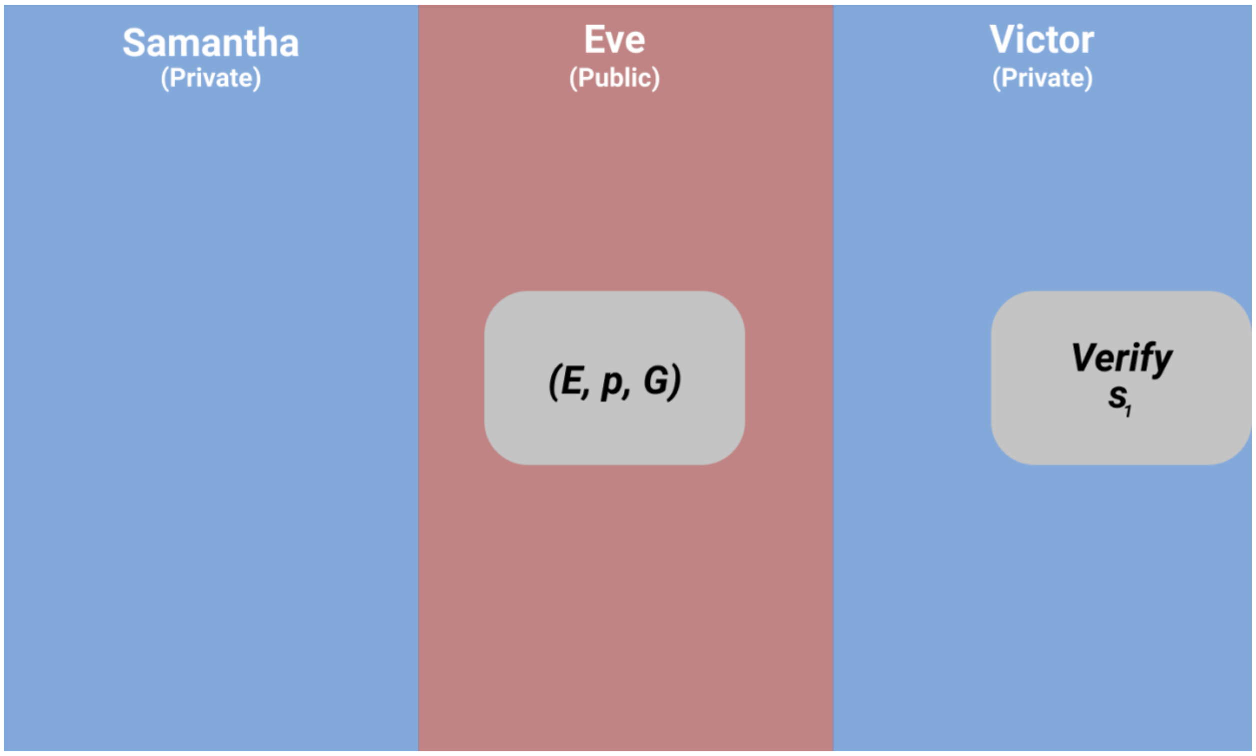
Samantha: Key Creation

17. Samantha chooses her own secret integer s such that $1 < s < q - 1$.
18. Samantha computes $V = sG$.
19. Samantha publishes the verification key V.



Samantha Signing

20. Samantha chooses a document *D mod q*.
21. Samantha chooses a random element *e mod q*.
22. Samantha computes $s_1 = x[eG] \bmod q$ and $s_2 = (D + s\, s_1)e^{-1} \bmod q$ where *x[eG]* is the x-coordinate of the point *eG*.
23. Samantha sends the signature (s_1, s_2) to Victor



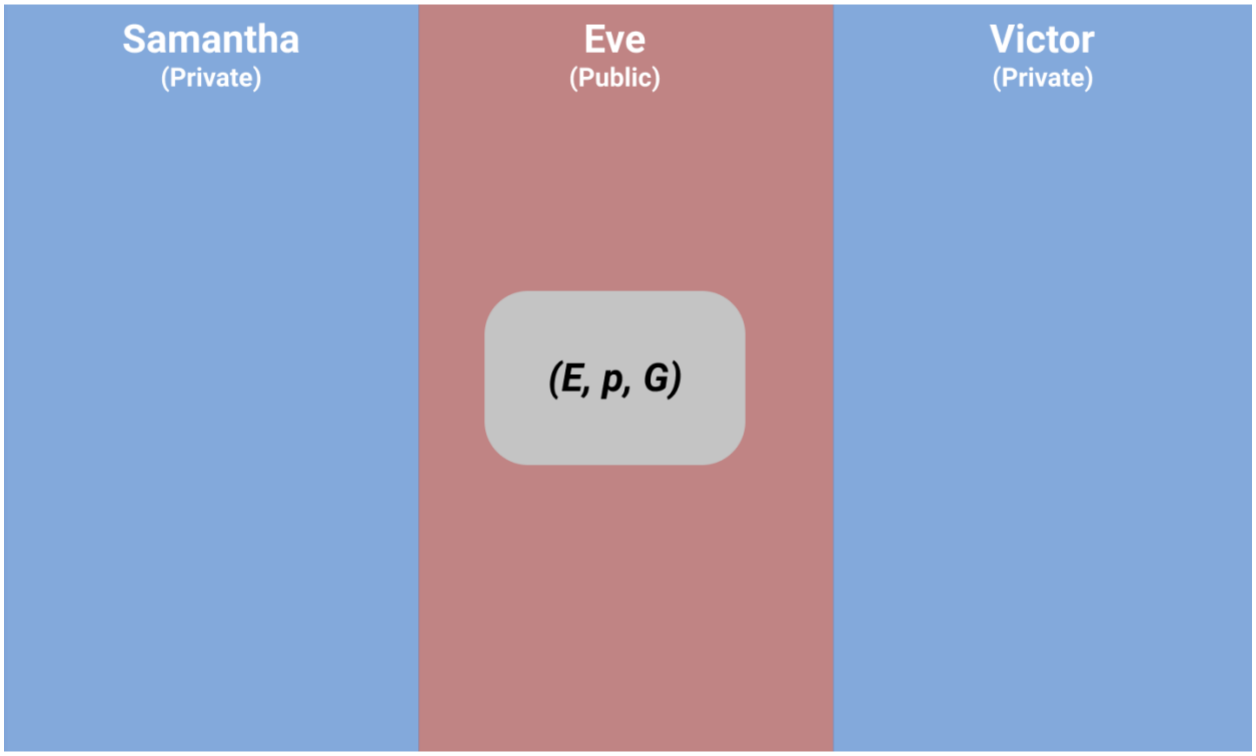
Victor: Verification

24. Victor computes $v_1 = d\, s_2^{-1} \bmod q$ and $v_2 = s_1\, s_2^{-1} \bmod q$.
25. Victor computes $v_1G + v_2G$.
26. Victor verifies that $x(v_1G + v_2G) \bmod q = s_1$.

Schorr Signatures

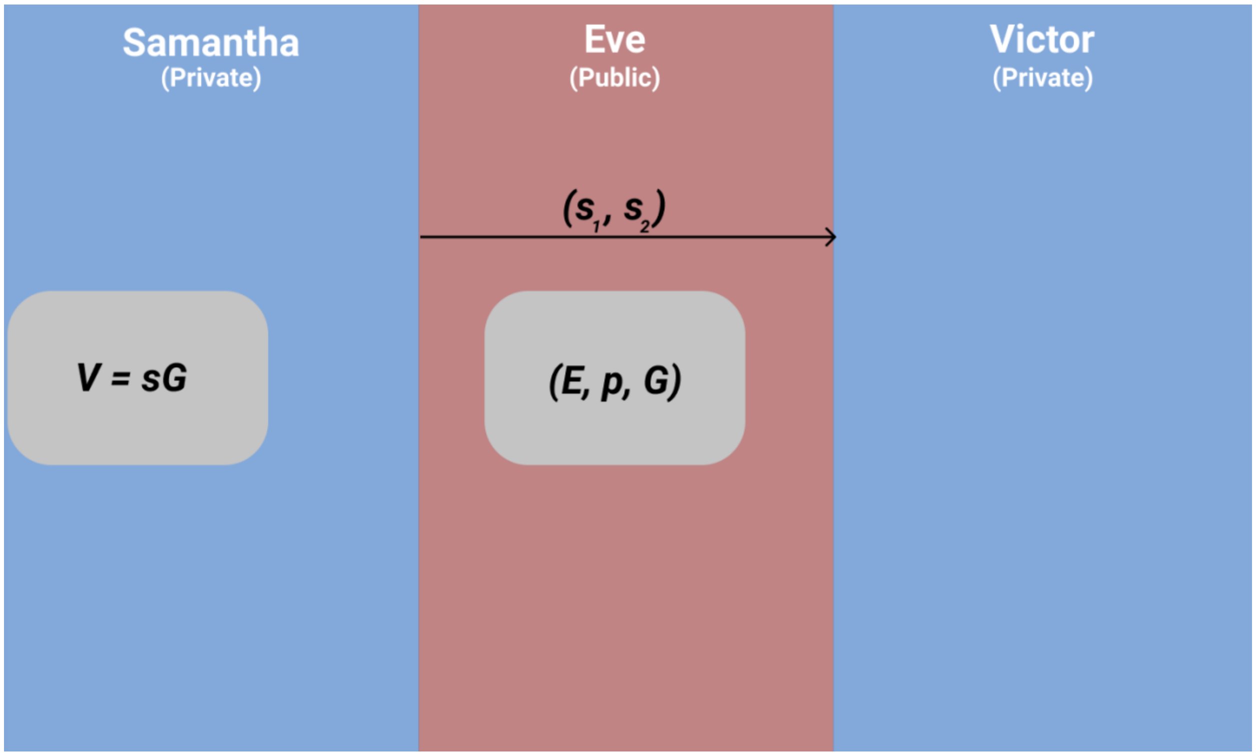
Creation of Public Parameters

1. Samantha and Victor do not share a secret key through a private channel.
2. Samantha or Victor chooses and publishes a (large) prime p of order q, an elliptic curve over a finite field $E(\mathbb{F}_p)$, and a point G in $E(\mathbb{F}_p)$. These values are public and shared by Alice and Bob.
3. Let m be the message hash.



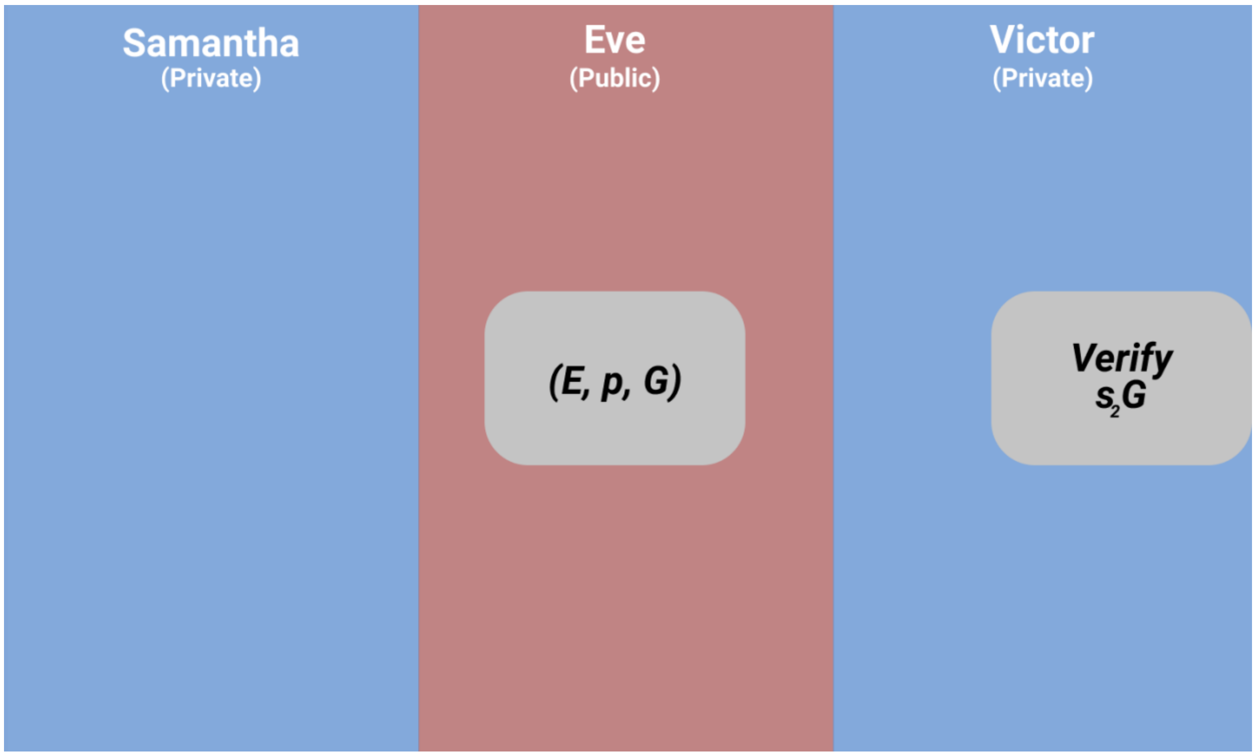
Samantha: Key Creation

4. Samantha chooses her own secret integer *s* such that $1 < s < q - 1$.
5. Samantha computes $V = sG$.
6. Samantha publishes the verification key *V*.



Samantha Signing

7. Samantha chooses a document *D mod q*.
8. Samantha chooses a random element *e mod q*.
27. Samantha computes $s_1 = x[eG] \bmod q$ and $s_2 = e + h(x[eG] \mid\mid V \mid\mid m) * s \bmod q$ where *x[eG]* is the x-coordinate of the point *eG*.
9. Samantha sends the signature (s_1, s_2) to Victor.



Victor: Verification

10. Victor computes *V from x[eG].*
11. Victor verifies that s_2G *mod q = eG + h(x[eG] || V || m)*V mod q*.

Utility of Schorr Signatures

If we compare ECDSA to the Schnorr Signature scheme, we see that ECDSA contains *inverses modulo q* while Schnorrr does not. What does this mean? This means that there are specific values in ECDSA that cause the algorithm to stop working properly. You can think of *inverses modulo q* as "division" in the algebra that you are used to working with. We know that you cannot divide a number by 0 as that operation in undefined; similarly, there are some values in ECDSA which will cause to operations within it to be undefined. With this in mind, we cannot easily perform algebraic operations such as addition or multiplication with ECDSA.

However, we can easily perform algebraic operations such as addition or multiplication in the Schnorr Signature scheme because it does not contain any "division". Schnorr is linear mapping:

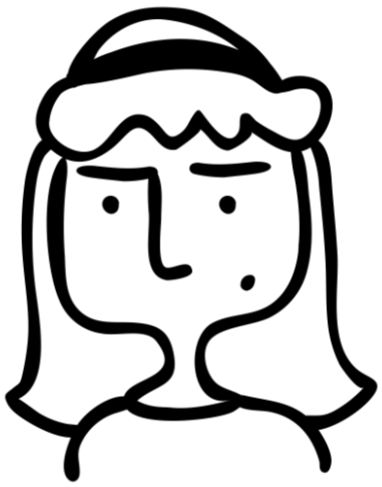
Linear Mapping: A mapping f(x) that satisfies two properties:

1. *Additivity: f(x+y) = f(x) + f(y)*
2. *Homogeneity: f(bx) = b f(x)*

where b is a constant.

One very interesting and useful application of Schnorr's linear properties is MuSig – a Schnorr-based multi signature scheme that is indistinguishable from single signature transactions on the blockchain. Through key aggregation (see preprint), MuSig greatly improves the efficiency and privacy of Bitcoin multi-sig transactions.

Epilogue



"And if you gaze long enough into an abyss, the abyss will gaze back into you."

Friedrich Nietzsche, *Beyond Good and Evil*

Bitcoin is an abyss whose lure strengthens as you delve deeper into its unknowns.

It may be intimidating to stare into the darkness of the abyss. As we look, the concepts and implications of Bitcoin Cryptography that we neither understand nor appreciate challenge our unarticulated presuppositions and paradigms of thought.

It may be even more intimidating to enter the darkness, to try and make sense of Bitcoin and its implications.

Those that enter the abyss may find themselves forever changed in an unpredictable manner.

Do not fear what you find in the abyss.

Do not fear if what you find changes you.