Lab 1: Intro to cryptocurrencies.

> In this lab, we will try to build our own simple coin in an interactive way. Along the road, we will learn to appreciate the cryptographic primitives that are used in modern cryptocurrencies and the problems they are trying to solve.

Designing our own simple centralized coin.

> Let's assume we want to create a coin for the class. The goal of the coin is to not use actual money but to somehow recycle the good deeds of students and to incentivize them to work harder. Each student can get a coin by answering questions on piazza or answer questions in class. If a student wants someone to answer their post on piazza, that student can offer coins in exchange for the answer. To make things even sweeter, a student can exchange some of their coins for grades. [This is a toy coin. We will not adopt such a scheme for the real class]

Does this coin have value for the students in the class? Can it have value for students that go to BU? What about outside the University?

Who should control the coin creation?

Who should control the balances of who owns what?

How can we transfer one coin from one student to another?

Can we transfer a coin from one student to another without going through the central server?

Can we decentralize coin creation?

First attempt to create the simple coin. The coin is controlled by the professor. The professor is going to create a server and host some code on it. What functions should our coin code support?

In [ ]:
```
# The professor has a server.
# The professor will host the code on their own server.

# dictionary holding each student's balance, {"student_id", "amount"}
students = {}

class DogeClassCoin:

  # who should call this function?
  def create_account(student_id, password):
    pass

  # who should call this function?
  def mint(amount, student_id):
    pass

  # who should call this function?
  def tranfer(sender, receiver, amount):
    pass
```

What about account authentication? A student must only send the coins that belongs to them!

In [ ]:
```
# The professor has a server.
# The professor will host the code on their own server.

# dictionary holding each student's balance, {"student_id", "amount"}
students = {}

# dictionary holding each student password, {"student_id", "password"}
passwords = {}
class DogeClassCoin:

  # who should call this function?
  def create_account(student_id, password):
    pass

  # who should call this function?
  def mint(amount, student_id):
    pass

  # who should call this function?
  def tranfer(sender, receiver, amount):
    pass
```

Assume that for some reason students can't access the server. Can you think of a way to make a transaction offline between two students?

Availability and Denial of Service Attacks

Accounts as Public Keys

> The server controlled by the professor can go down for several reasons (overload of students doing transactions, attack by an evil student trying to keep the average low, the professor forgot to pay for their server...). This will prevent a student from using their coin. Can you think of a solution to this problem?

1. Can we make transactions offline? would digital signatures help? What are the tradeoffs?
2. Can we replicate the server and host it on many servers? How would that work?

In [ ]:
```
# The professor has a server.
# The professor will host the code on their own server.
# server can go down
# use digital signatures to help make transactions offline.
# punish cheaters? (notify professors?)

accounts = {}

class DogeClassCoin:

  # who should call this function?
  def mint(amount, public_key):
    pass

  # who should call this function?
  def tranfer(sender_public_key, receiver_public_key, signature, amount):
    pass
```

What other benefits do we get by using accounts as public keys?

- Simpler server side code: Server doesn't have to handle account creation.

- Pseudo anonymity? In case students are hosting some of the replicated servers. They won't know what public key corresponds to what student.

Immutability, Auditability

Blockchains as append only data structures

> Consider the scenario where the server controlled by the professor got hacked.

1. What can we do to limit the damage of the hacker? Can we detect it? Can we revert back the damage done by the hacker?
2. What if the professor is doing things that are not ethical. Can we audit it?

In [ ]:
```
from hashlib import sha256
import json


# The professor has a server.
# The professor will host the code on their own server.
# server can go down

accounts = {}
# records all transactions that ever existed
transactions = []

class DogeClassCoin:

  class Transaction:
    counter = 0
    def __init__(self, sender_public_key, receiver_public_key, signature, amount):
      DogeClassCoin.Transaction.counter += 1
      self.id = DogeClassCoin.Transaction.counter
      self.sender = sender_public_key
      self.receiver_public_key = receiver_public_key
      self.sig = signature
      self.amount = amount

  # who should call this function?
  def mint(amount, public_key):
    # create a transaction and save it to the transactions
    pass

  # who should call this function?
  def tranfer(sender_public_key, receiver_public_key, signature, amount):
    # create a transaction and save it to the transactions
    pass

  # who should call this function?
  def get_transactions_digest():
    # get a hash of all the transactions that happened
    transaction_history = json.dumps(transactions).encode("utf-8")
    return sha256(transaction_history).hexdigest()

  def get_new_transactions_digest(old_digest, new_transaction):
    # get a new hash that the hashes the old hash and the new tranasction
    new_transaction_bytes = json.dumps(new_transaction.__dict__, sort_keys=True).enco
    return sha256(old_digest.encode() + new_transaction_bytes).hexdigest()

digest = DogeClassCoin.get_transactions_digest()
DogeClassCoin.get_new_transactions_digest(digest, DogeClassCoin.Transaction(
    "sender_public_key", "receiver_public_key", "signature", "amount"))
```

Out[ ]: '7281c3cfbbe1667ab31434b70864df6619573e823442965bd9c039bd3da0f335'

Smart contracts

What happens if a student doesn't send another student coins even after answering their question on piazza?

1. Make the professor punish them
2. Come up with an automated solution

In [ ]:
```
# The professor has a server.
# The professor will host the code on their own server.
# server can go down

accounts = {}

class DogeClassCoin:

  # who should call this function?
  def mint(amount, public_key):
    pass

  # who should call this function?
  def tranfer(sender_public_key, receiver_public_key, signature, amount):
    pass

  # who should call this function?
  def complain(accused_account, sender_public_key, receiver_public_key, signature, am
    pass

  # automated solution?
  # maybe charge the student when asking a question?
  # if someone answers the question then the coins get transferred to them automatica
```

Decentralizing coin creation.

The class has ended. The students are using the coin in campus beyond the class.The professor thought about it and they think that ethically they shouldn't be in charge of it anymore. However, they don't want to simply shut it down.

> Who creates the new coins?

> Can we let the students govern and maintain the coin together?

Key takeaways:

- Cryptocurrencies are just software running a program that tracks the balance of each user.

Cryptocurrencies have value if there is demand for them. Demand can be engineered.

- A cryptocurrency is decentralized:

> 1. If there exist no single entity that controls the coin software running on all servers.
> 2. Users would still be able to use the cryptocurrency normally even if any number of servers running the coin software stop working.
> 3. All coin software running on all different servers are on an agreement over the state of the software.

- If an entity controls the ability to mint coins of a cryptocurrency exclusively and at will, then that entity has a lot of leverage over that cryptocurrency.
- Decentralizing coin minting in a “fair” way is hard.
- If the servers running the coin software are trusted to run the software in an honest way then the cryptocurrency is centralized.
- A transfer from one party to another consists of decreasing the amount of coins from the sender and increasing it for the receiver.
- Every operation that modifies the state of the coin software running on the servers is called is called a transaction.
- To detect hacks and unfairness, all transactions should be recorded and backed up. The more parties that monitor those transactions and backs it up, the more safe it is.
- Fees are important because they make it harder for malicious users from flooding the servers with useless transactions. Transactions that will make it costlier for servers to to run the coin software.
- If digital signatures are used to authorize transactions between parties, and the servers running the coin software don’t maintain the secret keys of the users, then it would be harder for anyone controlling the servers (including hackers) to fake a transaction without being detected.