Introduction

• Who we are
  – Neha Narula
  – Tadge Dryja
  – James Lovejoy (TA)

• Digital Currency Initiative

• Course
  – Lectures (20%)
  – Labs (40%)
  – Final project (40%)
Cryptocurrency Engineering and Design

• What is a cryptocurrency?
• How is it different than a regular currency?
• What does it mean to build one?
What we are not going to do

• How to ICO
• Trading advice
• Permissioned blockchains
Origins of Money

Images of sheep, grains, various ancient and modern currencies © unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/fairuse.
Traditional payments

“I, Alice, would like to send Bob $5”

Alice

Bob

| Alice: $10 |
| Bob: $0 |
Traditional payments

Alice: $5
Bob: $5

Bob, I sent you $5!
Traditional payments

Alice: $5
Bob: $5
Traditional payments

Alice: $5
Bob: $5
Pros/cons of banks

Pros
• Digital payments

Cons
• Not peer-to-peer (bank must be online during every transaction)
• Bank can fail
• Bank can delay or censor transactions
• Privacy
The bank can fail

<table>
<thead>
<tr>
<th></th>
<th>Alice: $10</th>
<th>Bob: $0</th>
</tr>
</thead>
</table>

Alice
Bob
The bank can delay or censor

“I, Alice, would like to send Bob $5”

No!

<table>
<thead>
<tr>
<th></th>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>$10</td>
<td>$0</td>
</tr>
</tbody>
</table>

Alice: $10
Bob: $0
"I, Alice, would like a coin"
E-cash

Alice  Bob
E-cash

Alice → Bank → Bob

SN
E-cash

Alice

Bob

SN

BANK
E-cash
E-cash

Alice

Bob
Pros/cons of simple e-cash

Pros
• Digital payments
• Peer-to-peer

Cons
• Bank needs to be online to verify
• Bank can fail
• Bank can delay or censor transactions
• Privacy
Chaumian e-cash

• Alice can choose SN
• Alice “blinds” her message to the bank so bank can’t see SN
• When Bob redeems, bank doesn’t know payment came from Alice
Chaumian e-cash

“I, Alice, would like a coin b(SN)”
Chaumian e-cash

Sig(b(SN))

Sig(SN), SN

Alice

Bob
Chaumian e-cash

Alice → Bob
Sig(SN), SN
Chaumian e-cash
Chaumian e-cash

\[ \text{Sig}(\text{SN}), \text{SN} \]
Double spend detection

Charlie

Alice

Bob

\[ u_{\text{Alice}}, v_{\text{Alice}} \]
Pros/cons of Chaumian e-cash

Pros
• Digital payments
• Peer-to-peer
• Privacy
• Offline double-spend detection

Cons
• Bank can censor withdrawals and deposits
How to build decentralized digital token transfer?

1MHePtrqAxZ
Primitives for making a cryptocurrency

Hash functions

Signatures
Hash functions
Simple, right? But powerful.
hash(data) -> output
data can be any size; output is fixed size
Hash functions

Important. You can do everything* with just hash functions.

*can’t do some fun stuff with keys

(Key exchange, signature aggregation, etc)
Hash functions

Any size input, fixed output... output is “random” looking

What’s that mean? Deterministic, no randomness

But the outputs look like noise; half the bits are 1s, half are 0s
Hash functions
Somewhat more well defined - “Avalanche effect”: change 1 bit of the input, about half the output bits should change
Hash functions
Well defined: what it shouldn’t do
preimage resistance
(2nd preimage resistance)
collision resistance
preimage resistance

given y, you can’t find any x such that hash(x) == y

(you can find it eventually, but that will take $2^{256}$ operations ($10^{78}$))
2nd preimage resistance

given x, y, such that hash(x) == y, you can’t find x’ where

x’ != x

and hash(x’) == y

(this one is a bit of a mess so let's leave it at that)
collision resistance
nobody can find any $x, z$ such that $x \neq z$

$\text{hash}(x) = \text{hash}(z)$

(again, you can find them eventually. And in this case, not $2^{256}$)
resistances

Practically speaking, collision resistance is “harder”;
collision resistance is broken while preimage resistance remains.
Examples: sha-1, md5
usages
hashes are names
hashes are references
hashes are pointers
hashes are commitments
Commit reveal
Commit to something secret by publishing a hash
Reveal the preimage later.
Example: a1c089bf65e852cf2ba2010d2ba84e2025ec937b5f8b9dac682c35dcf498aef4
Commit reveal

a1c089bf65e852cf2ba2010d2ba84e2025ec937b5f8b9dac682c35dcf498aef4

Reveal:

I think it won't snow Wednesday! d79fe819

$ echo "I think it won't snow Wednesday! d79fe819" | sha256sum

a1c089bf65e852cf2ba2010d2ba84e2025ec937b5f8b9dac682c35dcf498aef4  -
Commit reveal

$ echo "I think it won't snow Wednesday! d79fe819" | sha256sum

a1c089bf65e852cf2ba2010d2ba84e2025ec937b5f8b9dac682c35dcf498aef4 -

Add randomness so people can’t guess my preimage; HMAC

This is a kind of proto-signature
Linked list with hashes

We could call this a “hash-chain”
Also, it’s basically git
Binary tree with hashes

How can 2 inputs go to 1 output?
Not a collision. Concatenate then hash: $h(a,b)$
What's a signature?
Signatures are useful! Messages from someone. 3 functions needed:
GenerateKeys()
Sign(secretKey, message)
Verify(publicKey, message, signature)
3 functions
GenerateKeys()
Returns a privateKey, publicKey pair
Takes in only randomness
3 functions
Sign(secretKey, message)
Signs a message given a secretKey.
Returns a signature.
3 functions

Verify(publicKey, message, signature)

Verify a signature on a message from a public key. Returns a boolean whether it worked or not.
Signatures from hashes
It’s doable! In fact, you’ll do it!
First pset is to implement a signature system using only hashes. This is called “Lamport Signatures”
Lamport Sigs: Generate key

0

1

Make up 256*2 random 256 bit numbers
Lamport Sigs: Generate key

Get hashes for each
Lamport Sigs: Generate key

0

= Secret key

1

= public key

... (256)
Lamport Sigs: Sign

Hash string to sign.
“Hi” = 8f434346648f6b96df89dda901c5176b10a6d83961dd3c1ac88b59b2dc327aa4

Pick private key blocks to reveal based on bits of message to sign
Lamport Sigs: Sign

Hash string to sign.
Pick private key blocks to reveal based on bits of message to sign 01101110
Lamport Sigs: Verify

Hash each block of the signature
Verify that it turns into the block of the public key
Lamport Sigs: Signing again

Signing more than once reveals more pieces of the private key
Lamport Sigs: Signing again

Signing more than once reveals more pieces of the private key
Lamport Sigs: Signing again

1 sig: can’t forge anything
2 sigs: \(\sim \frac{1}{2}\) bits constrained
3 sigs: \(\sim \frac{1}{4}\) bits constrained
pset01: Lamport signatures

In golang
On github
Most of the signing code is written
Tests implemented
Also public key with 4 signatures; try to forge another!
Office hours / messages on slack