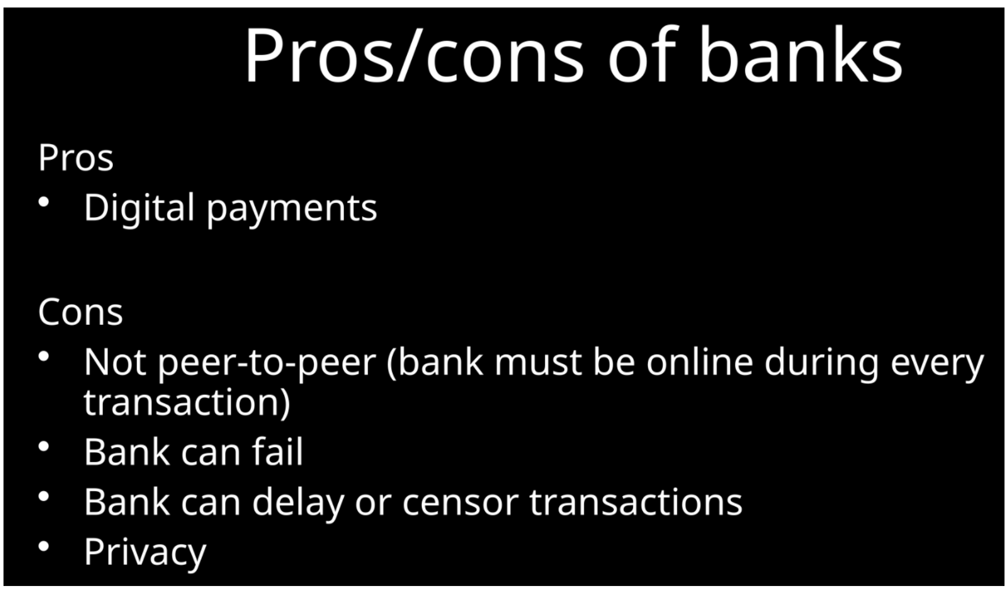
Lecture 1 - Signatures, Hashing, Hash Chains, e-cash, and Motivation

Resources

- Course Site on OCW
- Course's github

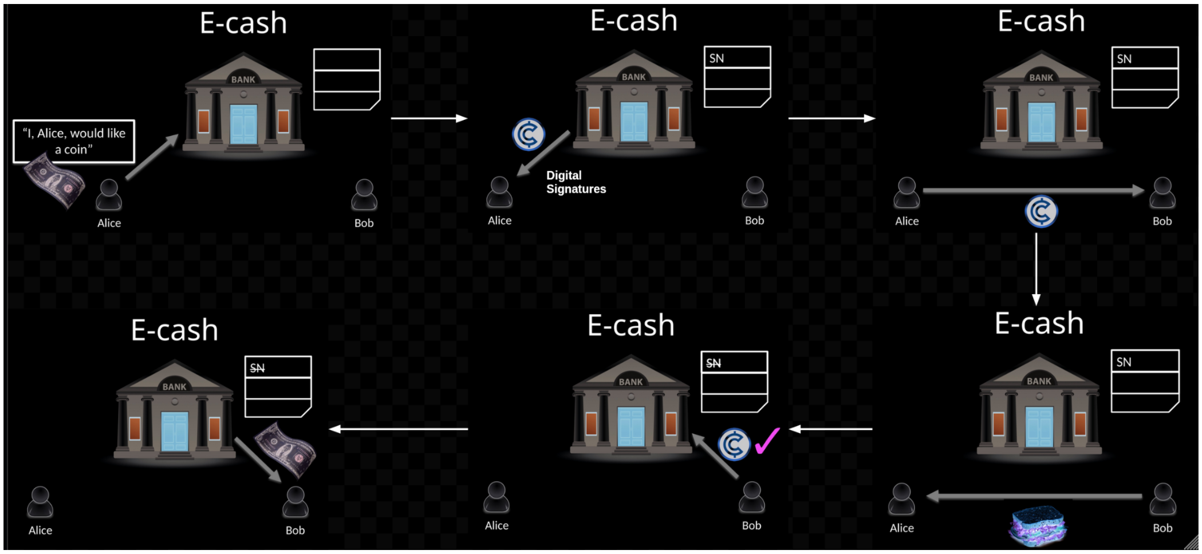
Money, Banks & E-cash

- Money / Tokens are valuable because people believe they are valuable
- **Banks**:



- **E-cash**
  - A fundamental problem of designing digital currencies is that people can't double spend it, since bits can be copied
  - *E-cash*: Alice gives some money to the bank so the bank generates a coin with a unique number (serial number - SN) which Alice can give to Bob to get like a sandwitch. Bob can go back to the bank and get

the money



Pros/cons of simple e-cash

Pros
- Digital payments
- Peer-to-peer

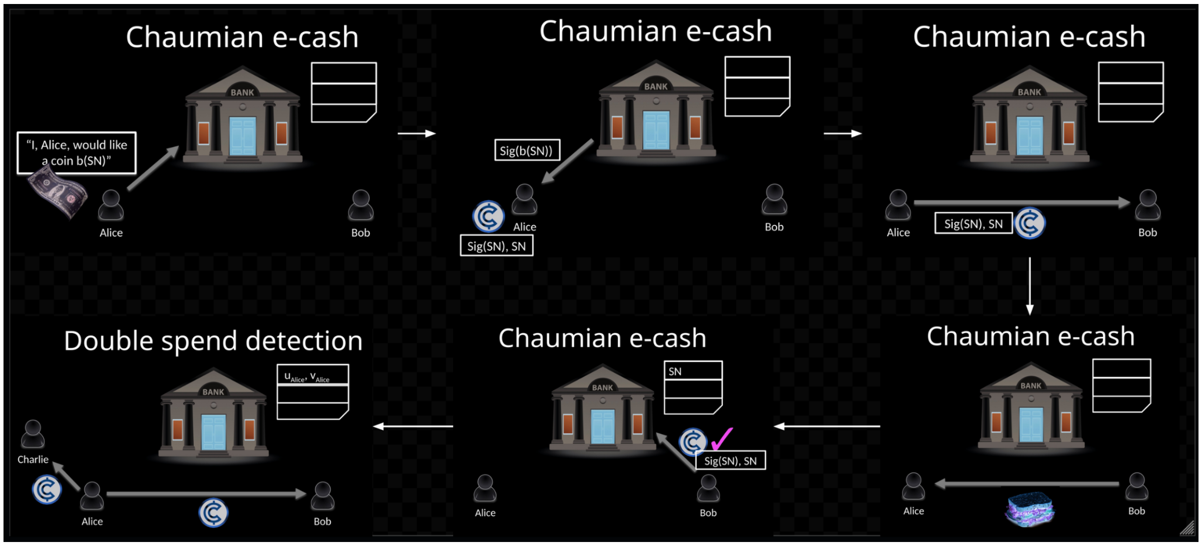
Cons
- Bank needs to be online to verify
- Bank can fail
- Bank can delay or censor transactions
- Privacy

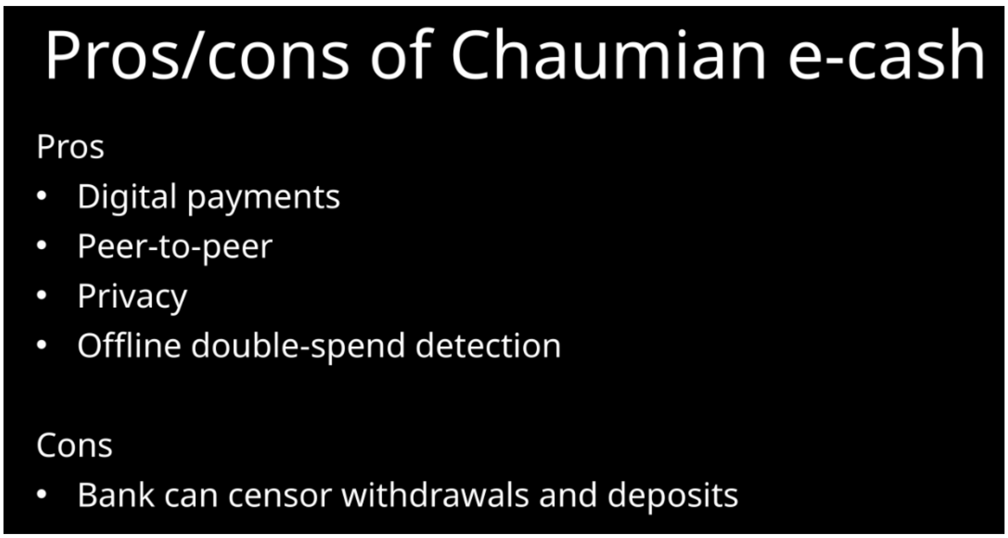
- **Chaumian e-cash**

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

`SN`: serial number. `b` is the blinding factor that Alice can apply and remove. `Sig`: the bank's digital signature



- If Alice gives a coin to 2 people, the bank will be able to detect it and can punish Alice
- 

Primitives for Making a Cryptocurrency

Hash Functions

Informal Def

- `hash(data) = output`, where `data` can be any size. `output` is fixed size
- You can do almost anything, e.g. build a cryptocurrency with a hash function
- `output` looks like noise: half the bits are 1s, half are 0s
- *Avalanche effect*: change 1 bit of the input, about half the output bits should change

Formal Def

- *Preimage resistance*: Given `y`, you can't find any `x` such that `hash(x) == y` (brute-force would take 2^{256} ops for SHA256)
- *2nd preimage resistance*: given `x`, `y`, such that `hash(x) == y`, you can't find `x'` where `x' != x` and `hash(x') == y`
- *Collision resistance*: nobody can find any `x`, `z` such that `x != z` and `hash(x) == hash(z)` (can succeed with 2^{128} tries - birthday problem)

Note

- We don't have a mathematical proof of a true one way function that's existing. Hash functions are built based on heuristic
- If you can break SHA256 with less than 2^{256} attempts, e.g. 2^{240}, it's considered successful

Hash Usages

- *Hashes as file names*: If anything in the file content changes, then the name changes
- *Hashes as references*:
- *Hashes as pointers*:
- *Hashes are commitments*: E.g. you publish a hash of your prediction on something, then when the day come, you publish the prediction and people can truly see that the prediction was made beforehand and if it is correct
- *Data Structures*: e.g. Merkel tree = a binary tree of hashes. Blockchain = a chain of hashes

Hashes as commitments - An example

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

Add randomness so people can't guess my preimage - this method is called HMAC (Hash-based message authentication). This is a kind of proto-signature

Signatures

Signatures are messages signed by someone. There are 3 functions needed:

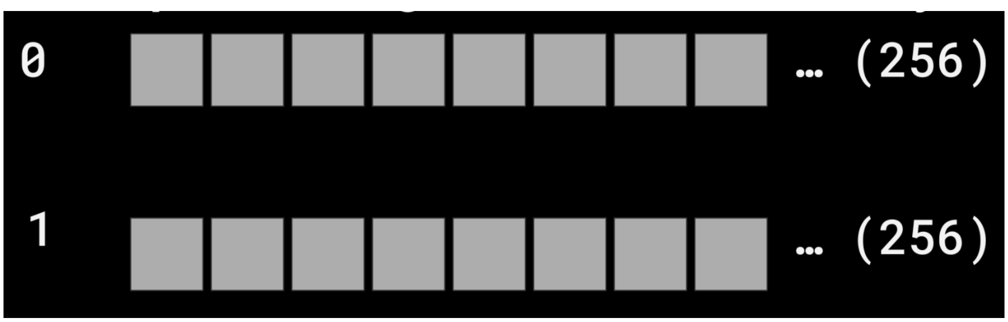
```
GenerateKeys()                            -> (secretKey, publicKey)
pair
Sign(secretKey, message)                  -> a signature
Verify(publicKey, message, signature)     -> bool if all 3 things
match up
```

Lamport Signatures: Signatures from Hashes

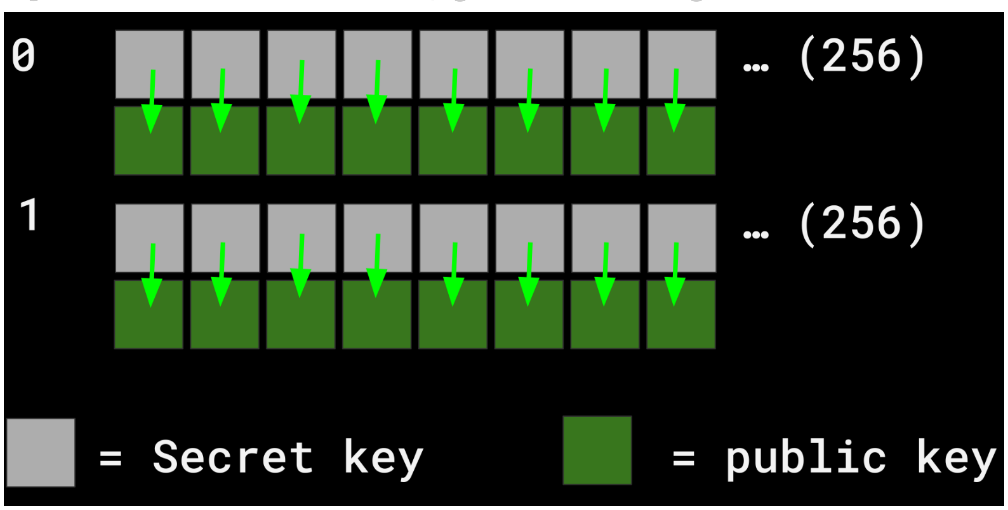
- We will implement this in the first pset

1. Generating the Keys

- **Secret key**: the secret key = 2 rows of 32-byte blocks, each row has 256 blocks

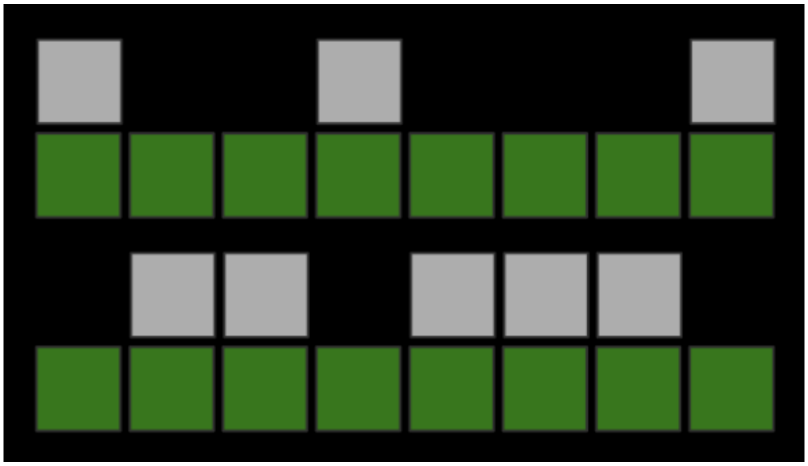


- **Public key**: For each of the block, get the hash (green arrow = hash func)



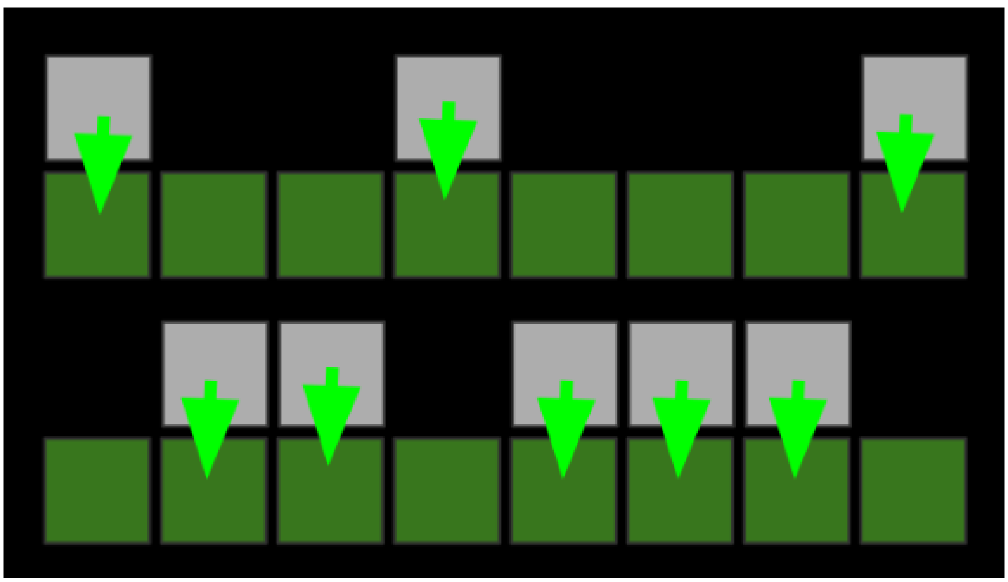
2. Sign

- Pick *private key* blocks to reveal based on bits of message to sign
- E.g. `message=01101110`



3. Verify

Hash each block of the signature on the message and Verify that it turns into the block of the public key



- For Lamport Signatures, you can determine the message just from the signature and the public key

Security

- No one can forge a signature from a public key, since no one knows the pre-images except for the one who holds the secret key

Lamport Signatures' Cons

> ⚠ **Warning**
>
> Signing more than once reveals more pieces of the private key and then people can start forging signatures
>
> - 1 sig: can’t forge anything
> - 2 sigs: ~½ bits constrained
> - 3 sigs: ~¼ bits constrained