CS 5433 Blockchains Crypto Smart Contracts

FINAL STUDY GUIDE

Topics

1. ~~Hash functions~~ ✓ ~~Applications~~ ✓
2. ~~Bitcoin Mining~~ ✓
3. ~~Byzantine Agreement Protocols~~ "~~Reconstruct Proof~~ Lecture on Zoom"
4. Key Management
5. ~~Smart Contract Vulnerabilities~~
6. DeFi Applications / NFT

Sub important

1 ~~Encryption and Digital Signatures~~ ~~4. ZK Proofs~~
~~2. Bitcoin Protocols, UTXO, Lightning Network~~ 5 Ethereum
3. ~~Proof of Work vs Proof of Stake~~ 6. Oracles
~~7. Commitment Schemes~~
~~8. Random Oracle Model~~ ✓
~~9. Merkle Trees~~
~~10. Nakamoto Consensus~~

Intro to Bitcoin

. 4 Key Ingredients for physical/digital currency
1. Creation 2. Forgery Prevention 3. Validity of Ownership 4. Transferability

Bitcoin combines ledger of asset ownership via the blockchain + secret key ownership of tokens

Bitcoin = Registered + Bearer Instruments and has the property pseudoanonymous (Ex. Captain Kirk)

Rules of Blockchains:

1. Strict Ordering 2. Rule based write, Global Read
3. No modification 4. Message Signatures used to Track Balances via UTXO model

Blocks

• Batch of transactions + digital signatures that are verified and chained

Central Authourities: Can fail to honor transactions on block or delete sever or go bankrupt, hyperinflation

Main Con → * Can Suppress Transactions

The Consensus Problems

Idea: Naive Consensus "broadcast to everyone else".
Cons: Nodes not always online, Message Delivery Varies
Idea: Vote often: Sybil Problem - people forge votes for benefit

Solution: Bitcoin Mining / POW

BTC Mining

- Solve Sybil by making users invest in computation system
- Miners race to find community signature for block of current ordered batch of transactions (Slot Machine)
- Miners Rewarded 6.25 BTC, 21M BTC in system for anti-inflation
- Proof of Work Computation power secures network

Hash Functions ~ the work horses of modern cryptography

Mining Goals

- To mine Miners find an input to Hash Function that has leading zeros so output is smaller difficulty threshold. To do this they try different nonces until they get it

Question

- Blocks per day ~ 24 Blocks × 6.25 BTC Reward × 37,000 USD ≈ 5.5 Million (Mining Pool Profit)
- The total Hash Rate 180 Million THashes/sec ~~onds~~ × 600 seconds / Mine a block ~ 6 ~~blocks~~ BTC/block per block

≈ 18,000 xhashes for a single BTC

Definition of Hash Function: Finger Print of the message that is compressed

"Message" of any length → [Hash] → Digest (Hex) n = 256 fixed length

Features of a Good Hash Function:

- H(x) should be easy to compute
- Preimage Resistance: Can't find input (preimage, infinite) given random output (image, finite) * has to be random
  "Pigeon Hole Principle"
- Collision Resistance: No pair of inputs map to the same output
  $W, X \not\to Y$ $H(w) = H(x) \neq Y$

Random Oracle Model → defines the perfect Hash Function tha cannot be achieved, used to prove security of Hash Functions

"Defines the ideal Model", Has the answer in an infinite long cells that contain n-bit string → 256 Hex digest Hash

Calculating Collision Resistance: 2^{256} is a huge number of outputs

If you can compute $2^{256}+1$ output hashes you're guaranteed to find a collision.

Rom implies preimage resistence, gives you a randomly selected output image, you need to use Brute Force and keep asking Oracle until you get output.

Question

Suppose Image is 256 bits, find x so $H(x) \leq \frac{2^{256}}{(d \cdot 2^5)}$

How many guesses on avg for d=1, d=32?

Max guesses for d=1 (Difficulty of Puzzle)

$H(x) \leq \frac{2^{256}}{1 \cdot 2^5} \leq 2^{251}$ $\quad H(x) \leq \frac{2^{251}}{2^{256}}$ ← your chance / ← Possible guesses

$H(x) \leq \frac{1}{2^5}$ ← probability of hit on one trial = $\frac{1}{32}$

$E[H(x)=1] = \frac{1}{32} \cdot 32 = 1$ so 32 trials on expectation

Avg guesses of $d=32 = \frac{1}{2^{10}}$ probability = 2^{10} trials on expectation

1024

No maximum number guesses because you are dealing w/ probs

* Hash Function is not an encryption algorithm.

Birthday Paradox: How many people so collision prob $\geq 1/2$

X_{ij} denote ball j lands in bin i, collision $\sum X_{ij} \geq 1$

$$E[X] = \frac{C(K,2)}{N} = \frac{K(K-1)}{2N} \approx \frac{K^2}{2N} \approx K = \sqrt{N}, \quad K = 23 > \sqrt{365}$$

For collision, by birthday paradox collision w.p $\approx 1/2$

requires 2^{128} throws ← 128 bit security is considered strong

In homework we reduced the time by storing seen hashes and increasing p(collision $X_{ij} \geq 1$) as we iteriated and added coins to each hash until one had 4 collissions

Where Hashing is used:

1. Addresses – double hashed Public key
2. Transaction ID – because of collission resistance unlikely to find duplicates → Inclusion Proofs via Merkle Trees.
3. Chaining of Blocks – Hash/Block Headers (History of whole System)
4. Mining – Solving puzzle find input to Hash w/ leading zeros
5. Commitment Schemes – Alice can use H(message) to hide it

Commitment Scheme

- Choose random secret key r, $\{0,1\}^{128}$, Alice gives $C = H(m \| r)$ its hard to decipher m because it can be anywhere in ROM
- Property called computationally "hiding"
- Can verify C when Alice reveals m, r through decommitting
- Doesn't Allow 'cheating' because no two inputs = output C

$H(1\|r), H(0\|s)$ are not both equal to C.

by Collision Resistance property of Hash

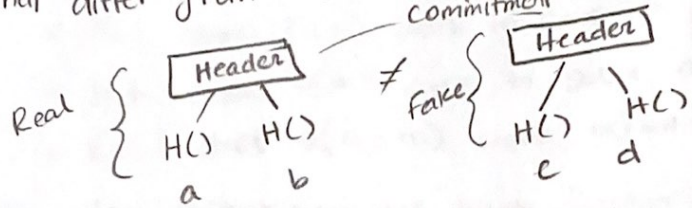
Properties of a Committment Scheme

1. Efficient to Compute C (Verification)
2. Hiding: Hard to compute m from commitment C (Pre Image Resistance)
3. Binding: Hard to change m for given C (Collission Resistance)

Merkle Trees: represent transactions in compact form, represented as hash pointers

How to prove Tx ID has been committed? Show a path to root node

It can't be forged because you will find pair of hash pointers that differ from real due to collission resistence



$H(a||b) \neq H(c||d)$ because of collision resistance

Steps from HW. 1. Generate Tree of leaves and hash pointers w Root = Commit

2. Generate the Proof String for a given leaf from leaf to root
3. Verify: Check that the leaf is in the proof string and commitment equals the root of proof string

Merkle Signature Scheme

Signature is the preimage of leaf along with path to root, each leave has a message.

1. Key Gen - generates secret, public keys and build Merkle Tree and outputs root PK
   "Preimag" "image"
2. Sign - generates signature path of sibling nodes with message

Public Key Cryptography → Key Sharing * Encrypt under Public key Encryption: PK used to initate Algorithm

PK Cryptography - Key Exchange: users and websites share sk k, over public web

Merkle Puzzles: A selects k puzzle and sends puzzles to Bob.

Bob solves puzzle revealing sk and an index, sends back. Alice knows which puzzle Bob solved and the corresponding secret key index

* Alice does $O(n)$ work to construct puzzles.
* Bob does $O(m)$ work to solve a puzzle
* Eve does $O(n \times m)$ work because she doesn't know index

Question: How might you use hash function to construct a puzzle?

Puzzle requires m hashes to recover key k

- Like a commitment scheme, pick random value k, r,

Compute $P = H(k \| r)$, Puzzle $P = H(k \| r)$ ← customized puzzle with salting

r, to make pre computing harder

Puzzle = [Enc[secret[i] || 0's, Puzzle Index] . . .]

- Shuffle and send to Bob who Dec $g \| 0^{n-s}(m)$ and gets g, Puzzle Index
- Alice looks up her key List for index establishing join key sharing

Diffie Hellman Agreement: HAVE TWO PARTIES COMPUTE g^{ab} USING DL Problem

Discrete log problem: $x = \log_g y$, x cannot can be computed in polynomial time allowing us to compute secretly in exp space

$y = g^x$ ← hidden

Num generation

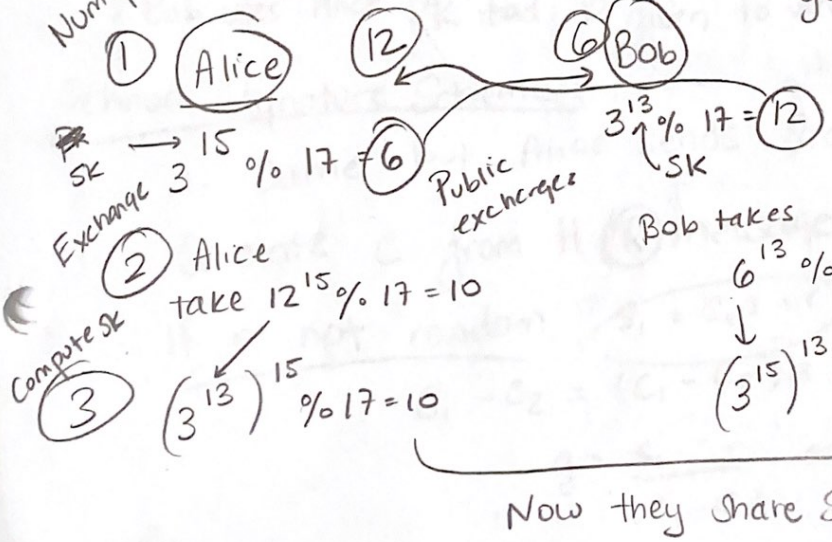
not hidden

generator = 3

mod = 17

Eve

- Doesn't know a, b from Alice or Bob
- She can only compute g^{a+b} not g^{ab}



Digital Signatures : Private Key used to start algorithm (Message Integrity)

- Everyone knows what your Public Key Ring looks like
- It can only be produced by someone with your Ring (Private Key)
- Anyone can verify authenticity of seal, but only you can create one

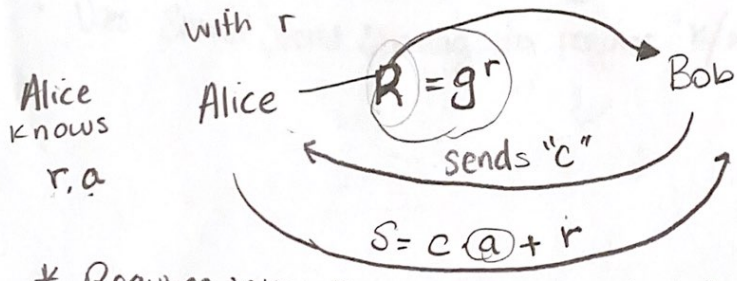
1. Key Gen (Public, Private), 2. Sign (Message, Signature)
3. Verify (M was signed by SK)

Goal: Prevent tampering of message, only signed by Message Owner

Schnorr Identification Scheme (Identifying someone in public)

1. Alice generates a random r, 2. Bob sends a challenge to Alice c
3. Alice takes c and makes $s = c \cdot a + r$ and hides her private key. (sends g^r to Bob)
4. Bob computes $g^s = g^r \cdot g^{ac}$ verifying Alice.

and Alice removes secret key from exponent but blinds it with r



* Requires interaction so not a signature scheme
* Bob uses Alice PK and R given to verify s.

Bob computes and checks

$g^{ca+r} = R \cdot A^c$ (A: given, Public; c: known to Bob)

If this holds Bob knows it's Alice w/o her revealing sk a ↑ hidden private key

Schnorr Signature Scheme

1. Same but Alice sends Signature (R, s)

Generate c from $H(R \text{ message})$

If r not random? $s_1 = c_1 a + r$ $s_2 = c_2 a + r$

$s_1 - s_2 = (c_1 - c_2)a$ ← p is the same!

$a = \frac{s_1 - s_2}{c_1 - c_2}$ and a is revealed!

Bitcoin Scripts / Protocols

ECDSA + Schnorr: Uses curve and subgroup to create signature

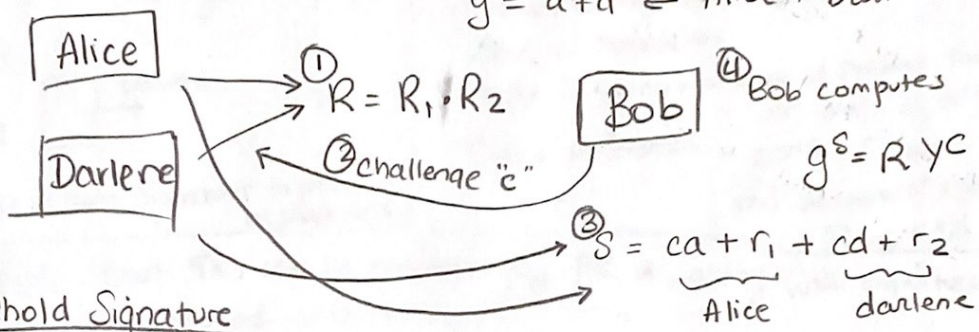
$$s = k^{-1}(e + xa)$$

- k^{-1}: blinding
- e: message
- x: blinding
- a: private key

Nice property where signers can easily generate joint signatures

Joint Schnorr Identification: $Y = g^y$ ← joint public keys

$y = a + d$ ← Alice + Darlenes Private key



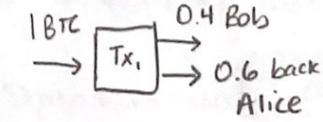
Threshold Signature

- Any k out of n (3/4) can sign with joint signatures
- group has a public key shared, private key shared
- Uses Shamir Secret Sharing to require K/N to use signature.

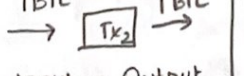
Bitcoin Scripts / Protocols

UTXO Model: Unlike account based system, no explicit balances only sets of transactions that contain Bgs of Money

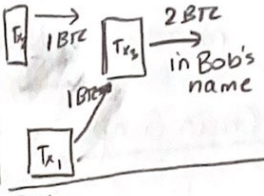
Change Address: Must be consumed entirely and money returned to self



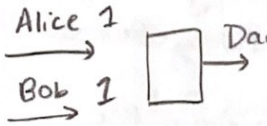
Verification:



Consolidation of Funds: Combine UTXO to 1 address



Joint Payments: Inputs from two people



* Needs Alice + Bobs Signatures to prove legitimate send

Transaction Meta Data:

- Transaction Hash ID
- Lock time
- Size of Transaction
- Input References – hash of previous Transaction an index of output, signature
- Output References – array [Value, Output ≤ Input] and address of Recipients Pubic key

Bitcoin Scripts: Goal: This can be redeemed by PK X, along with signature of PK

Lets Recipient spend UTXO money.

- A combination of **scriptPubKey** and **scriptSig** to validate Transaction
  - scriptPubKey: output script specifies an address of PK/Recipient
  - scriptSig: input script: specifies signature so user can pay with UTXO

Pay 2 Script Hash: When recipient is using MultiSig your can pay 2 a script by providing correct data

Script Address A = H(S) → S + Sig / Executes secondary script / Complexity pushed to input script

Escrow Transactions

A needs to send Money for B's goods. Add person C, require 2/3 signatures Judy can arbitrate and sign to release escrow money.

Question Alice pays 10 BTC into 2/3 MultiSig Transaction, If Bob delivers both Alice/Bob sign w/o involving Carol. If one cheats, Carol signs w/ the other to either return or pay the non-cheater.

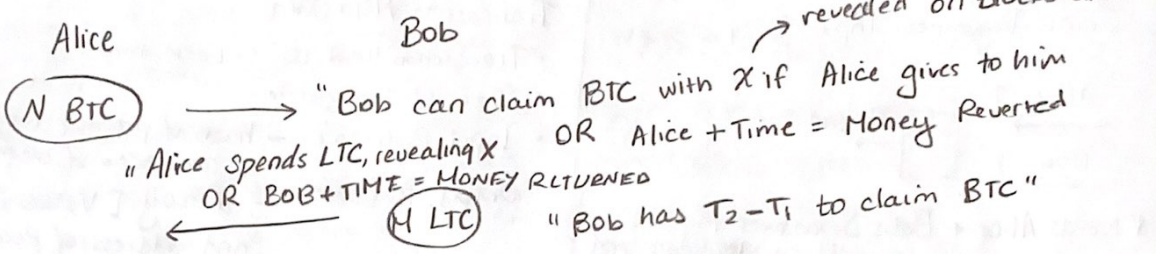
Green Addresses: A trusted third party / bank that pays recipient and guarantees it doesnt double spend money. No longer used

Micropayments: paying for subscriptions, create a multisig that allows Alice ~~to borrow a little bit at a time~~ prefund and withdraw a little at a time, Bob only publishes final transaction

Lock time enforces that Alice gets money back if Bob doesnt claim

By Lightning Network : built on bidirectional payment channels. where most transactions are off chain, money flows in either direction and channel closed after time Δ. There is a dispute period
How paths are discovered? Monetary capacity of nodes to fund movement of money. Network of payment channels to connect participants

Atomic Cross Chain Swaps : Trade different crypto currencies "fair exchange"



Zero Knowledge Contigent Payments : Fair exchange knowledge for money
Ex. Sudoku, crossword, private key, password. Use Public Key Encryption

1. Alice posts challenge f
2. Bob comes up w/ answer X

Zero Knowledge Proof: Alice verifies Bob knows X via ZK

3. Alice pays Bob under condition X is revealed on spent transaction or time runs out w/ Alice signature
4. Bob spends N BTC

Byzantine Agreement : Consensus protocol for Blockchain a distributed system

Goals 1. All loyal generals reach same decision 2. Commanding general obeyed if loyal

Byzantine Broadcast Protocol: must work no matter subset of corrupt nodes as long as upper bound by $f < n$, assume synchronous message int

Two Requirements : Regardless of corrupt nodes

Consistency: If two honest nodes output b and b' respectively then $b = b'$ (agree on same action)

Validity: If sender is honest and receives input b, all honest nodes should output b (players must agree on commander b)

How to attack naive majority vote protocol? Assume $n = 2k+1$, node 1 is corrupt

1. Divide $2k$ honest nodes into S_0, S_1 and send opposite bits to both groups 2. Have S_0, S_1 vote for opposite 3. Send majority of attacker vote to both groups. S_0 and S_1 think majority = 0, 1 achieving output inconsistency.

Player Model: Each P_i can behave honest or corrupt, $n-f$ = honest

Adversarial Model: Adversary A controls all f faulty players

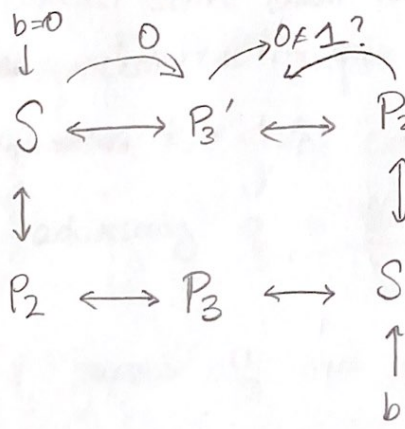
Communication: Synchronous and node-node via channels.

Byzantine Agreement only possible w/ $f < n/3$, Impossibility Proof

Proof by Contradiction, create a situation where its ambiguous who's faulty

Suppose three players S, P_2, P_3, and copies ~~all and none are honest~~

Two instances of Π with $f = 1$



① * Has validity S' and P_2' output 1 by validity, possible fault is P_3'

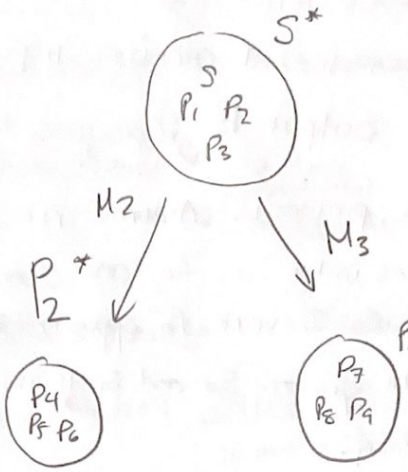
② * Second situation, S, P_3' output 0 by validity P_2' possible fault

③ * If it were to have consistency ← faulty msg P_2' and P_3' should from S output same bit but $0 \neq 1$

and we have contradiction and ambiguity of who's faulty

Theorem 2 $n>3$, No BB protocol for $f \geq \lceil n/3 \rceil$

Suppose Theorem 2 does not hold, and Π^* is secure $f \geq \lceil n/3 \rceil$



If S^* is faulty:

Consistency holds among non faulty nodes

$P_2^* = P_3^*$, Π^* is consistent

Validity: If S^* honest, P_2^*, P_3^* are honest

But Π^* being consistent and valid then violates Thm 1

proving Thm 1 and thm 2 Π not secure for $f \geq \lceil n/3 \rceil$

Phase King rotate through leaders till find honest leader and use guidance unless super majority vote

- Protocol is secure for $n//4$ faulty with $n+1$ phases

Proof of Stake Solutions How do decide block creator fairly?

Mining Economics

$\frac{\$X}{MWh}$ is mining profitable?

Given:

Difficulty = 200M TH/s Cost of ASIC: $10,000

Price = $40,000 per BTC Specs: 110 TH/s Power: 3000 W, 24 months

Steps

(1)

Fraction Total Mining = $\frac{110\ TH/s}{200,000,000\ TH/s}$] Our ASIC

Expected Block Reward: F. 6.25 BTC · 40,000 = $\frac{13.75\ \$}{block}$

Daily Gross Profit: $\frac{24hrs \times 60min}{10min \times 13.75\$}$ = $20 per day

Cost Equipment/Day = $\frac{\$10,000}{24 \times 30\ Days}$ = $13.69/day

20 − 13.69 = $6.30 per day w/o electricity

(2) $\frac{3000W}{day}$ = 0.003 MW × 24 = 0.072 MWh for day

Elec X < $\frac{\$6.30}{0.072}$ for mining to be profitable

= $87.5/MWh

Nakamoto Consensus

Select longest active chain to add verified blocks by Miners

- Monetarily incentivized to pick longest fork, no final chain
- Six Confirmation Rule: for confirmed TX's

If adversary $q > \frac{\beta = chainlength}{\lambda_h - block\ production\ rate}$, adversary controls

Forking lowers q and allows adversary to take chain

Bitcoin Mining Economics

→ w.p 1.6%

ASIC = 73 TH/s, 31 years to mine, replaced 6-12 months

Mining Pools: Aggregate mining power and share rewards.

Mining requires $H(\text{blockheader} + \text{nonce}) \leq D$

Question

① How can unsuccessful pool manager prove he did work?

A: Provide nonces s.t $H(\text{blockheader} + \text{nonce}) \leq D', D' > D$ for some difficulty D'

② How do pools prevent running off with block reward?

- Send block reward to admin and worker can't change the address of tx and prove she did work for block.

Asic Resistant (Memory Hard) Hash Functions making hard to create ASIC

Proof of Stake vs Proof of Work

- Replace computational resource with money.
- Make Stakeholders lock capital and then use as trustful board to verify & txs.
- Use randomness to choose commitees / identify block creators

weights stake Amounts

* "Good Randomness is hard to find"

P2P Network

nodes in P2P broadcast transactions and blocks to entire network & nodes after running scripts like validation of tx's / blocks. They also store ledgers of the blockchain. Supernodes: denoting full node that is holds DB

Lifecycle of Tx

1. Tx created and signed in wallet
2. Tx broadcasted in P2P network
3. Tx picked up by miners and placed in blocks
4. Block B mined, P2P propogates B
6. B added to everyones blockchain and Tx ✓

Proof of Stake Solutions How do decide block creator fairly?

- prevent cheating and stalling

RANDAO

1. Players send commitments w/ deposit on chain smart contract
2. Players decommit, R computed 3. Deposits returned to players

Verifiable Random functions: generate randomness locally w/ proof correctness

VRF(SK, coin#) $< d$ ← calibrated to choose enough players

↑ = a private key ← staked coins

$VRF(a, x) = z = x^a$ given $A = g^a$

In POS, nothing at stake to fork but Ethereum slashes comp

Key Management

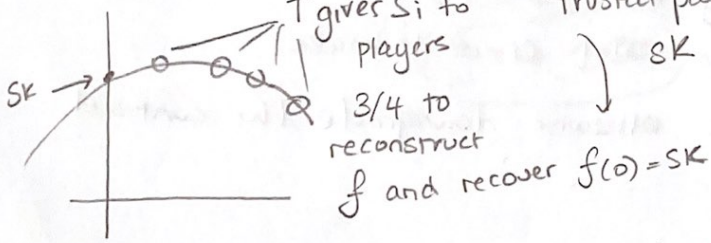
- Security vs Recoverability tradeoff
- Wallet identity with ECDSA digital signatures SK, PK
- K/N multisig is only works if K signatures but cost more
- Secret Sharing: SK is split into N shares but needs K players to reconstruct

Procedure: Allows single cost of single transaction

Sharing 1. Trusted party generates S_{ALICE} key, S_{BOB} key $\oplus$ SK (XOR)

Reconstruction 2. T collects (S_{ALICE}, S_{BOB}) and computes $SK = S_{ALICE} \oplus S_{BOB}$

Sharing done via polynomial $f()$ degree $K-1$, $f(0) = SK$



If Alice AND Bob must participate and Carol OR Drew

make a (2,2)-sharing SK where

$S_1 = (2,2)$ sharing between S_{ALICE} & S_{BOB}

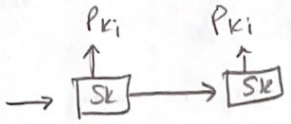
and $S_2 = (1,2)$ Sharing between S_{CAROL}, S_{DREW}

If T not trustworthy, Threshold sigs jointly generate sk and compute Sig w/o reconstr

Mnemonic Seeds: Memorable 12 words

to recover Secret key

Heirarchical Derivation: Derive many SK's from 1 seed



BAD, ONE LEAKED SK leaks all of them

Good, inversion resistance prevents leak and also protects all SKi

Cold vs Hot Storage

- off grid
- wallet, usb hardware
- 98% holding

- on network
- SK in device network

Paralysis Proofs

- If A, B, C share money in SC
- They send deadman switch check to make sure all are alive otherwise downgrade the contract

... third party w/public state

Privacy: K-anonymity due to crowd size

- multilayered by P2P network
- timing and stylometry
- address reuse undermines privacy
- Privacy being good or bad depends on circumstances

Mixers / tumblers

- People send BTC to mixer to get back untraced coins but they can take all money

CoinJoin

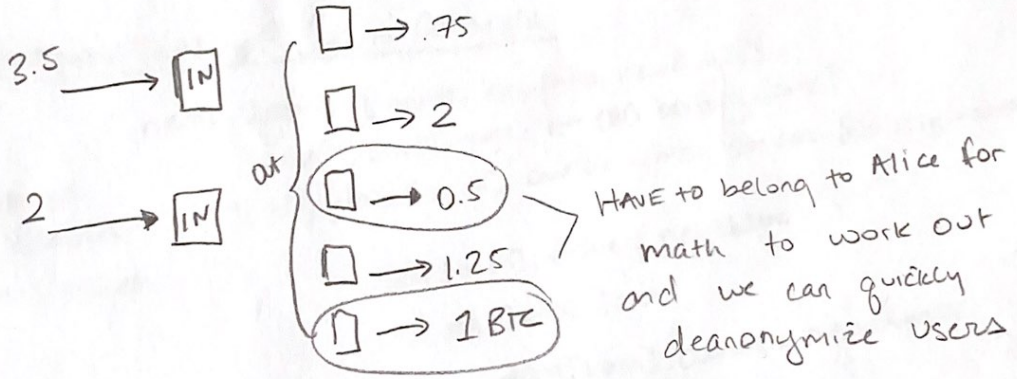
- involves distinct players who all sign

but Mixer can deanonymize user

A → IN → A'
B → IN → B'
C → IN → C'

Chaumian CoinJoin: blinded output address so Mixer doesn't deanonymize

Question Deanonymize users of CoinJoin



Smart Contracts : trusted third party w/ public state

- Solves the fair exchange / fair execution
- transparent, immutable, guranteed to execute as programmed.
- Replicated State Machines: collection of machines executes same sequence of inputs (transactions on blockchain)

Ethereum Virtual Machine: hardware that runs network (stack based script) where memory is very expensive

Gas and tx limits: creates economic computation costs

all transactions specify gas required ~ 21000 gas = $4.20 for Eth

Solidity: common language framework for Smart Contracts

modifiers → access control payable = accepts funds

ex. modifier onlyBy (address _account)
    require (msg.sender == _account)

SC fixed forever and sc calls required to update

- a.transfer(x) → calls other contracts sending X to address "a"
- foo.call.value(3).gas(20764) → call functions provide gas

Fallback Function: unamed and called if no function specified or non-existent function called.

Vulnerabilities of Smart Contracts

name Update (name, newValue, newOwner)

hash = sha3(name) ← can be any name?

→ if data[hash].owner = msg.sender && msg.value ≥ Update cost

- if sender is current owner and paid enough

data[hash].value = newValue

if newOwner != 0:

→ data[hash].owner = newOwner

this sets the new owners hash but doesn't delete the old one! which can still be looked up.

Pitfalls of smart contracts: RPS game needs simultaneous features to work so need a commitment scheme that is binding and hiding

Smart Contract Vulnerabilities

Safe practice steps

1. perform input validation and check on current state
   discard messages if validation fails
2. Update local state balances
3. Pass on interaction to trigger other contracts if all states valid

Switching → 2 allows for → 3 re-entrancy bug
infinite deposits by DAO attack

Zero Knowledge Proofs Steps (contigent payments)

1. Alice posts challenge f on chain
2. Bob sees f and comes up w/ x.
3. Bob says I know secret x such that $H(x) = Y$ and $f(x) = \text{True}$
4. Alice sends N_{BTC} → (Bob signs and reveals x) OR (Alice signs and T_1 expires)

Idea: given a function $f(x) \rightarrow$ True, False prove x such that $f(x) = \text{True}$ without revealing anything else
there exists efficient ways to prove knowledge and doesn't have to be interactive.

How to prove zero knowledge?

Show interaction can be simulated therefore user learned nothing. Whatever Alice is showing to Bob, Bob could produce on his own

Schnorr Identification. is ZK proof because Bob can run protocol on his own that looks valid but learns nothing.

Lecture 22: Oracles

Collateralized Lending Contract: Borrower deposits Eth and withdraws Stablecoins

* Collateral value drops below threshold, then contract liquidates
  - ETH gets auctioned

Contract needs to know value of Eth on the market to determine liquidation threshold.

Blockchains lack Internet Connections

def
- **Oracles**: Relay system ^ Off chain that relays data ^ to smart contracts on chain; every DeFi app uses an oracle
- **Generalized**: Off-Chain system that connects bc's with other systems

How to build an oracle? Idea: Build oracle into consensus protocol.

→ but the miner can lie or cheat about information from other data sources

→ **How to deal with integrity?** Digital Signature Scheme so 1 node doesn't tamper with data.

* Majority Honesty ⇒ valid flight information

→ **How to check if sending node goes down?** Sigle point of failure is not a problem due to other P2P nodes that can take over. Through decentralization. Multiple nodes deliver value to SC's. Allow for backup transmission.

→ **What if the Website goes down?** Allow for backup data sources. Nodes have access to multiple websites. Be careful sites dont always rely on a single source

→ **What if there is Heterogeneous data?** Nodes disagree on right value based on time. Take majority value.

May need to be adjusted for numerical data.

Compute the median: fact given a minority of bad values median is an honest value or bounded by honest values

Other problems in building Oracles

1. Ensure nodes get paid for service? Billing
2. Ensure oracle reports are mined in a timely way?
3. Ensure majority of nodes are honest?
   - → pay good behavior, penalizes bad
   - → use reputation

DEXES enable on-chain trading of various asset pairs

Exchange tokens for currency, vice versa

Side effect: current valuation of asset price

Mispricing reveals arbitrage opportunity

Price on DEXES should start to match centralized

We can use a DEX as a price oracle.

(Cryptoeconomic Oracles)

Pros:

- instant response, doesn't need oracle systems for contracts
- composable with other contracts
- economic assurance of correctness

Cons:

- only good for price information
- can be manipulated • move liquidity to swing price

FLASH LOANS (used for Arbitrage)

- Remarkable unique to smart contracts
- Borrow and pay it back in one transaction
- If you fail to pay back, transaction / loan reverts

Oracles Continued : Lecture 22

Flash Loans: gambling contracts don't return rewards in single transaction because they need to get good randomness
- Future blocks or off chain systems

NFT Based Games

- NFT = "Non fungible token", token that is unique
- In NFT games, pieces/characters are NFT's
- NFT's generated randomly (CryptoKitty breeding)

VRF: Verifiable Randomness function for NFT's

Oracle use SK to generate randomness

$VRF_{sk}(a, x) \rightarrow z = x^a$, proof of correctness

↳ can't be coded on a SMARTCONTRACT because private key will be revealed

- can combine with secret-sharing to achieve threshold
- Secret shared, K players compute VRF. No K-1 players can break VRF.

Wrapped Currency: Cross chain bridge, oracle locks token in one oracle and issues tokens on another chain

Oracle Privacy

- flight data sent to Oracle so its not in Smart Contract and not on chain.
- TEE a trusted execution environment where people can't tamper with execution of App x.
- program running in enclave can't see the data in program

MakerDao Exercise

Suppose you have 1.5 Eth, maximum you can make in vaults?

- 150% collateralization, 1 Eth = $3000

Step 1 1.5 Eth worth 4500, vault for 3000 DAI

→ 3000 = 1 eth, deposit in bank

→ $\frac{2}{3} \cdot 3000 = 2000$ Dai to buy $\frac{2}{3}$ eth

→ $\frac{2}{3} \cdot \frac{2}{3} \cdot 3000 = 1300$ Dai $= 0.433$

$$\text{Eth} = 1.5 + 1 + \frac{2}{3} + 0.433 \approx 4.5 \text{ Eth}$$

DAI collected $= 3000 + 2000 + 1300 = 9000$ DAI

Automated Market Makers

$X \sim$ balance of A Trade maintain invariant $X \cdot Y = \boxed{C}$

$Y \sim$ balance of B

Example Suppose $C = 10$, $X = 10$ Eth, $Y = 1$ BTC (market) Eth = $40 BTC = $400

What happens if I trade 1 Eth for BTC

trade 1 eth, $X' = 11$ ← market

after you trade (market) → $Y' = \frac{10}{11} = 0.9$ so you receive $Y - Y' = 0.1$ BTC

Suppose I trade ε units of A for $d\varepsilon$ units of B

to maintain invariant $(X + \varepsilon) \cdot (Y - d\varepsilon) = C$

$d = \frac{Y}{X}$ is the current exchange rate

Miner Extractable Value Duedate May 6th

- HFT characteristics
  - arbitrage bots/algo • frontrunning • investment in low latency
- Life is brutish, nasty and short
- Bots engage in unusual game, imperfect information
- Continuous time and partial all-pay

Grim trigger - one player will threaten another if other player doesnt play. ie. raise of gas prices can ~~cause~~ be punished

- Convergence over time toward equilibrium.
- Wall Street Latency Wars
  - Smart Contract allows multiple trades to execute atomically
  - Pure revenue trades
- In 2020, gas fee ~~trades~~ auction changed to Flashbots

Systemizing arbitrage → Flashbots - bypass mempool, move competition off chain
Arbitragers submit bundles to Miners - direct bribe.

613M has been obtained from flashbots, Miner Extractable Value

The arbitage in crypto exceeds HFT on wall Street.

ICO - initial coin offering (tokens unlock/available

NFT Drops - collection of NFT's displayed to market)

Time-bandit attacks

MEV cause forks to be more profitable and Miners can use ~~multiple~~ latent block to subsidize attack by bribing Miners or wreck Mining power.

Lecture 25: Non Fungible Tokens

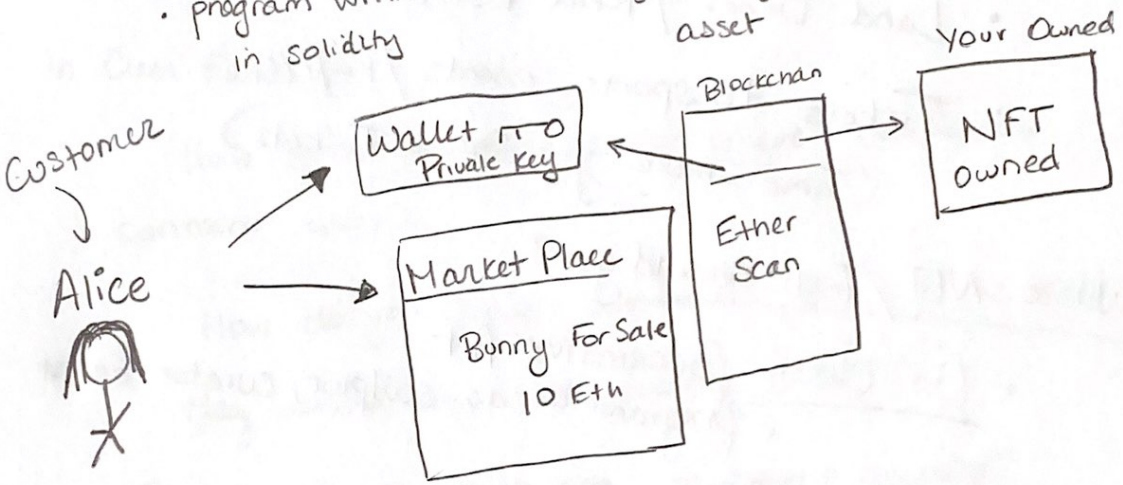
Philosophy of NFT's

Bored Abe Yacht Club and Cartoon Punks Groups for NFT's

What are NFT's?

① Consensus
- program written in solidity

② The State
- some digital asset



Why Buy NFT?

- Social Signaling (Prada handbag, Rolex) on online mechanisms far widespread than physical means.
- Can't be Stolen, if you believe in the concept of land ownership, you believe in the concept of NFT
- Modern generation has accepted digital ownership (Fortnite, Apps) Etc.

NFTs are not ~~art~~ just art

Fungibility: mutually interchangeable dollar bills

Everything non-fungible/semi fungible in universe

Ex. Fine Art, Profile Pics, Collectibles, Crowdfunding

Examples of NFT

- Loot, Fractional Ownership, Identity Tokens
- NBA Top Shot → video of NBA game that you can own.
- Game Studios will sell NFT characters.
- Drivers License on Blockchain
- Land Deeds / Real Estate on Blockchain
- Tickets to sports events / Raffles (Time share for Season Tickets)

Where NFT / Tech should go?

- On-Chain Generative Art
  - Programmer as sculptor, curator, artist
- On-Chain Purity
  - Art stored on contract itself to improve ownership philosophy
- Mutating Meta-Data
  - Changing picture based on transfer of sale/owner
  - Ex. growing / evolving crystal art
- Meta-data that sources from real world
  - Picture changes form during day / night or by season based on time zone.

Lecture 28: Criminal Smart Contracts

Example 1: Leakage of Secrets.

Example 2: forging certificates / Key Theft

Key Theft: Create multitarget contract commission fairness ≠ fair exchange
that may release reward for all Certificate Authorities

Example: Calling Card Crimes

Verifies Assasination happened, Who is responsible?

In Class Exercise:

How might you make a non unique calling card
contract work?

How do you prevent adversaries from guessing true cc?

May assume, say, 1% guessing probability

Require deposit > 10,000, to prevent guessing

Defenses against Criminal Smart Contracts

1. Ban decentralized smart contracts
2. Anonymity revocation 3. Blacklist Coins

HARD PROBLEM

Decentralized Identity: making digital drivers license tied to your Public key

- Improve user experience
- Reduce Fraud
- Empowers users to manage their own data
- Enterprises don't have to manage sensitive data

Problem 1 : Credential Issuance

- Who are the authorities?
  - DMV and people need to switch over

Problem 2 : Key Management

Users can't manage keys

Coinbase just keeps your private keys

Problem 3 : Sybil Resistance

User takes multiple identities