Blockchain and Smart Contract Mechanism Design Challenges

What are we talking about today?

Cryptoeconomics is about...

- Using **cryptography** and **economic incentives** to achieve information security goals
  - Cryptography can prove properties about messages that happened *in the past*
  - Economic incentives defined inside a system can encourage desired properties to hold *into the future*

*Credit to Vlad Zamfir for this characterization

Claim: it is not proof of work, nor decentralized money, nor linked-list data structures, but specifically cryptoeconomics that is the single key fundamentally transformative idea that came out of Satoshi’s code and whitepaper.

Note on public vs consortium chains

- The cryptoeconomic approach is more useful in public chain applications, as in restricted-identity applications there are often legal/social ways of penalizing bad actors
- However, there are sometimes parallels
  - “Fault accountability” in consensus

Applications of cryptoeconomics

- Consensus layer
  - Proof of work
  - Proof of stake
- Second layer
  - Smart contract mechanisms
  - Gadgets (mechanisms that get used by other mechanisms)
  - Channel constructions (lightning, Raiden, Truebit, etc)

The first is cool, but today we focus on the second.

Two ways to look at on-chain applications

- **Separated concerns approach**: assume bottom layer (consensus) works perfectly. Ensuring correct operation of the consensus layer is the consensus layer’s responsibility. Using this assumption prove that second layer works fine.
- **Integrated approach**: look at and analyze attacks on both layers simultaneously.

Claim: both are useful. Separated concerns approach often works as an abstraction, but it is important to note where the abstraction is more likely to fail.

Desired properties of the consensus layer

- **Convergence**: new blocks can be added to the chain but blocks cannot be replaced or removed
- **Validity**:
  - Only valid transactions should be included in a block
  - Clock should be roughly increasing
- **Data availability:** it should be possible to download full data associated with a block
- **Non-censorship**: transactions should be able to get quickly included if they pay a reasonably high fee

Security models

- In traditional fault-tolerance research, we make an **honest majority assumption**, and use this to prove claims about correctness of algorithms
- In cryptoeconomic research, we make assumptions about:
  - Level of **coordination** between participants
  - **Budget** of the attacker
  - **Cost** of the attacker

Security models

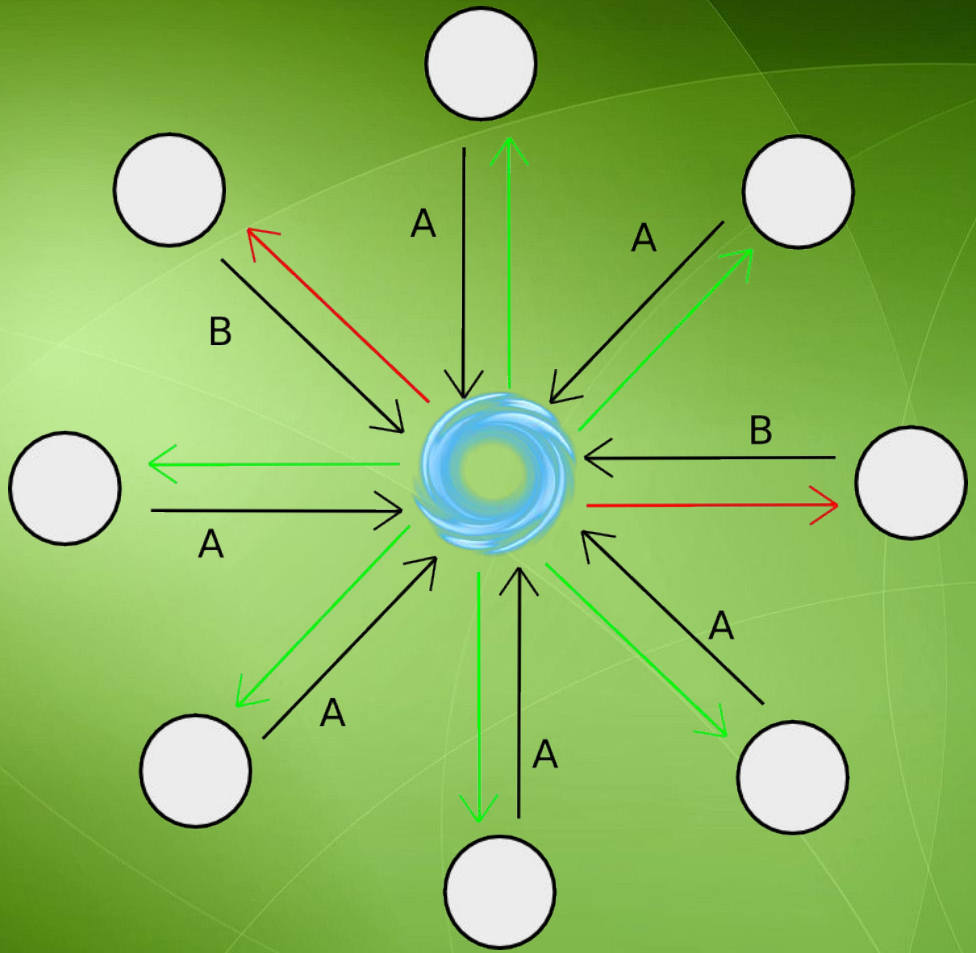
- **Uncoordinated majority**: all actors make choices independently, no actor controls more than X%
- **Coordinated choice**: most or all actors are colluding, though in second-layer systems we may rely on free entry from non-colluding actors
- **Bribing attacker:** all actors make choices independently, but an attacker can add their own money to influence participants' payoff matrices

Fault tolerance of Bitcoin

Model	Fault tolerance / security margin
Honest majority[1]	~½ (as latency approaches zero)
Uncoordinated majority[2]	~0.2321
Coordinated majority	0
Bribing attacker	~13.2 * k **budget**, 0 **cost**

1. http://bravenewcoin.com/assets/Whitepapers/Anonymous-Byzantine-Consensus-from-Moderately-Hard-Puzzles-A-Model-for-Bitcoin.pdf
2. http://fc16.ifca.ai/preproceedings/30_Sapirshtein.pdf

Example: Schellingcoin


A
A
B
B
A
A
A
A
A

Example: Schellingcoin

- Uncoordinated choice: you have the incentive to vote the truth, because everyone else will vote the truth and you only get a reward of P if you agree with them
- Why will everyone else vote the truth? Because they are reasoning in the same way that you are!

Example: Schellingcoin

- Coordinated choice: security margin exactly zero, because total payoff is the same regardless of result

P + epsilon attack

A bribing attacker can corrupt the Schellingcoin game with a **budget** of P + ε and zero **cost**!

Base game:

	You vote 0	You vote 1
Others vote 0	P	0
Others vote 1	0	P

With bribe:

	You vote 0	You vote 1
Others vote 0	P	P + ε
Others vote 1	0	P

Are coordinated choice models realistic?

Yes.

Are bribing attacker models realistic?

- Subsidized mining pools (eg. to influence segwit vs BU voting)
- Subsidized stake pools in PoS
- Exchanges offering interest rates, participating in coin voting on users’ behalf

Smart contract applications

- Outsourced computation and storage
- Provably fair random number generation
- Providing true info about the real world (“oracles”)
- Governance (DAOs)
- Stable-value cryptocurrencies (“stablecoins”)
- Bounties for solutions to math or CS problems
- Telling the time

Outsourced computation, case 1: problems in NP

(see also: https://eprint.iacr.org/2015/460.pdf by Andrew Miller et al)

```
def accept_solution(soln):
    if correct(soln):
        send(msg.sender, self.balance)
```

```
def commit_solution(solnhash):
    self.commits[msg.sender] = {
        hash: solnhash,
        validBlock: block.number + 10
    }

def accept_solution(soln):
    if correct(soln) and \
            block.number >= self.commits[msg.sender].validBlock and \
            sha3(soln + msg.sender) == self.commits[msg.sender].hash:
        send(msg.sender, self.balance)
```

Outsourced computation, case 2: general computation

Simple idea: save intermediate states

- Suppose we can represent $y = f(x)$ as $y=f_n(f_{n-1}(...(f_1(x))...))$
- Submitter sends intermediate states of computation:
  - $S_1 = f_1(x)$
  - $S_2 = f_2(S_1)$
  - ...
- Each f_i can be computed within a transaction
- Submitter also submits a deposit

Simple idea: save intermediate states

- Within some challenge period, anyone can submit a “challenge index” i
- If S_{i+1} != $f_{i+1}(S_i)$, then the challenger gets the submitter’s deposit
- If no challenges are made within the challenger period, submitter gets their deposit back plus a reward

Is it profitable to cheat?

- Let: c = cost of computing, D = deposit, r = reward

(submitter, challenger)	Submitter computes fairly	Submitter cheats
Challenger checks and challenges if needed	(r, -c)	(-D, D)
Challenger does nothing	(r, 0)	(r, 0)

Finding the Nash equilibrium

(submitter, challenger)	Submitter computes fairly	Submitter cheats
Challenger checks and challenges if needed	(r-c, -c)	(-D, D-c)
Challenger does nothing	(r-c, 0)	(r, 0)

- Let: P_s = prob submitter cheats, P_c= prob challenger checks

$$R_s = r\text{-}c + P_s(c - DP_c) \qquad R_c = P_c(DP_s - c)$$

$$P_s = c/D \qquad P_c = c/D$$

In many situations, there will be an inherent tradeoff between capital efficiency and correctness

Extended idea: multi-step game

- Submitter submits (S_0, $S_{512,}$ S_{1024}) + deposit
- Challenger disagrees with one of these answers (WLOG say the first), submits (S_0, $S_{256,}$ S_{512}) + deposit
- Submitter disagrees with one of these answers (WLOG say the second), submits (S_{256}, $S_{384,}$ S_{512}) + deposit
- …..
- Challenger submits (S_{314}, $S_{315,}$ S_{316}), result verified on-chain

Interactive games and trust assumptions

- Interactive games (incl. all of the above, channels, lightning, Raiden) **lean very heavily on the non-censorship property of a blockchain**
- Normally, censorship implies denial-of-service
- Here, censorship implies theft

Challenge flood attacks

- Send a very large amount of challenges at the same time
- Victims do not have enough block space to reply to all challenges in time
- Attacker unfairly “wins” in at least some situations
- This works on **any** interactive protocol

Challenges

- Can we detect censorship and have online full nodes reject censoring blocks?
- Can we make it impossible to censor some things without censoring everything?
  - “Ethereum is resistant to soft forks” … but only somewhat
    - http://hackingdistributed.com/2016/07/05/eth-is-more-resilient-to-censorship/
    - https://pdaian.com/blog/on-soft-fork-security/
  - More resistance via in-protocol scheduling

Challenges

- Can we detect flood attacks in-protocol and automatically delay challenge periods?
  - Doable in ethereum: if a block is X% full, count it as being worth only 1-X of a block
- Can we dual-use deposits in interactive games with deposits in proof of stake?

Auctions and Privacy

Usual second-price auction

- Phase 1: everyone submits sealed bid
- Phase 2: everyone unseals bid, top bidder wins and pays second highest bid

Crypto challenges

- To prevent submitting very many sealed bids and only opening the ones you want, a sealed bid should have a deposit
- How large is the deposit?
- If the deposit is the size of the bid, this reveals info about the bid size
  - Destroys incentive compatibility

Possible solution

- Allow deposits to exceed size of bid (refunding excess at reveal time), then distribute 0.1% of auction revenue to all bidders *in proportion to excess deposits*
- Goal: encourage “fake submissions” with very low value but high deposits
- An attacker can bribe depositors to reveal their values, but this invites even more people to make fake submissions
- TODO: formalize all of this

Intuition: mechanism design often relies on a party that you can trust for both correctness and privacy. A blockchain can be trusted for correctness, but not privacy. Hence, there are additional challenges in designing incentive-compatible mechanisms that can run on a blockchain.

TODO: formalize all of this

Randomness

PoW randomness

- Idea for coin flip game: both parties put in 10 ETH, if next block hash odd party A gets 20 ETH, if even party B does
- Problem: exploitable by miners!
  - If I play the game **and** am a miner, and I create the next block, then I can selectively not publish it if I dislike the outcome

EV(honest) = -10

EV(cheat) = 10 * 0.5 + (-10) * 0.5 - 5 = -5

Cataloguing attacks on randomness gadgets

- Arbitrary selection (you set the result to what you want)
- Dice re-rolling
- Influence (eg. shift probability of heads from 50% to 52%)

PoW randomness

- Single block
  - Re-rolling cost = block reward
- Majority function of N blocks
  - Cost of influence ~= O(sqrt(N)) * block reward

See http://www.cs.technion.ac.il/~idddo/CoA.pdf and other works by Iddo Bentov

PoS-style randomness

- RANDAO (http://github.com/randao/randao)
- N parties submit hashes + deposit
- N parties all submit preimages
- Result is xor of preimages
- If any party does not send their preimage in time, game restarts, absentee’s deposit lost
- Economic security property: can force a re-roll at cost of one player’s deposit

Timelock randomness

- Compute some non-parallelizable function of, say, a recent block hash
  - Iterated hashes (eg. SHA3)
  - Iterated modular square root (eg. Sloth https://eprint.iacr.org/2015/366.pdf)
- Intent: it is not impossible to compute the function of a value made available at time T until time T+x for some known x
- Can add a cryptoeconomic game to incentivize revealing ratio of problem hardness to time

Other challenges

Other challenges

- Stablecoins
  - Two challenges: (i) price oracle, (ii) mechanism, see https://github.com/rmsams/stablecoins and http://makerdao.com/
- Provably fair games
  - Games with private random info tend to be hardest, eg. poker (see literature on “mental poker” protocols)

Other challenges

- Incentivized data storage
  - Paying for download vs paying for availability
- Can we incentivize geographical decentralization?
  - One idea: incentivize being very close to at least some of the users of the system, with greater incentives for users who are underserved; assume that users are geographically decentralized