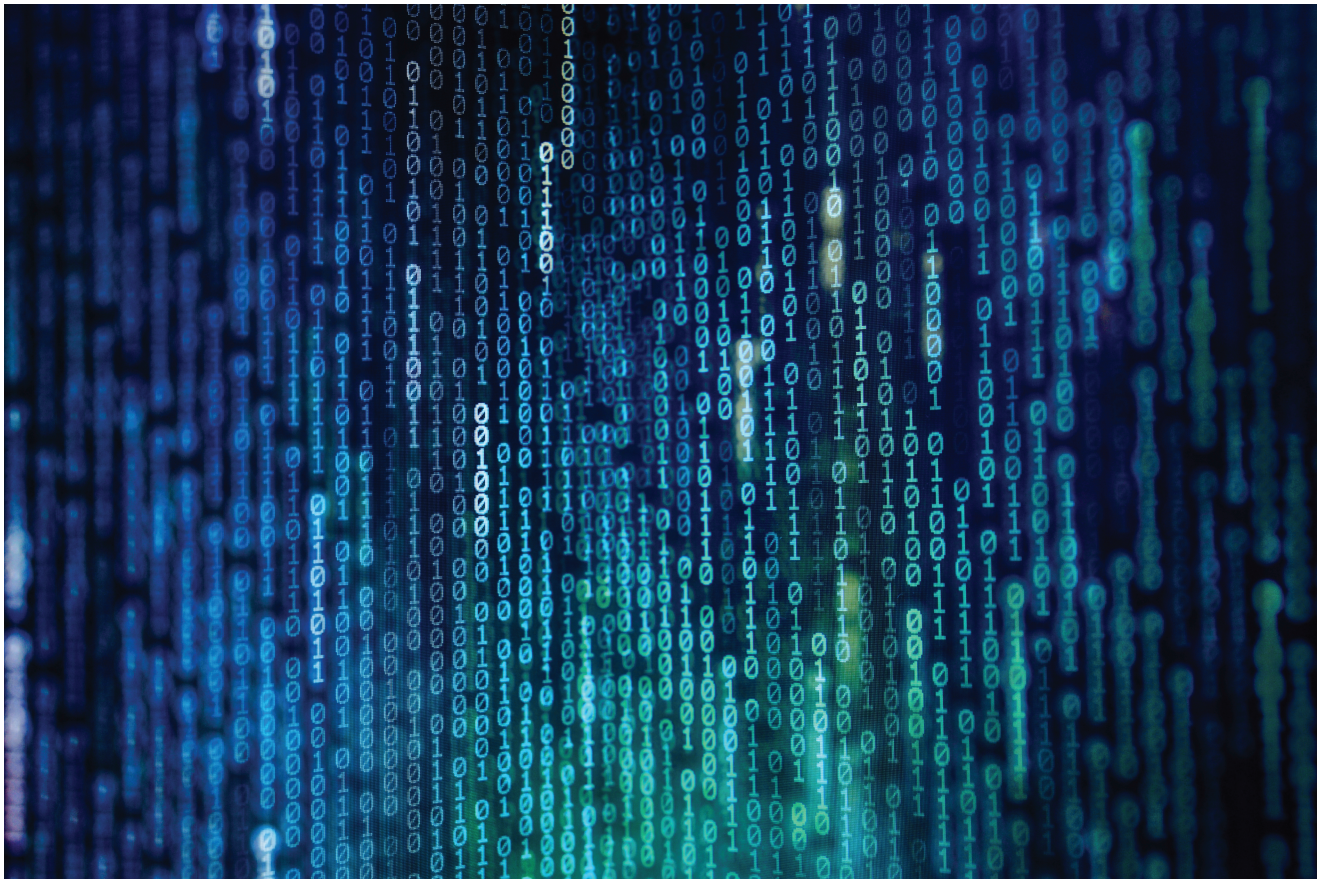
Blockchain Viewed from Mathematics



Yan X Zhang

Much of human activity involves *assets*, objects that hold value that can be transferred between users, such as gold, cash, or stocks. In the modern era, we want some assets to be *digital*, or stored entirely as computer data. Since there are no physical items (like gold bars) to represent such an asset, its existence is equivalent to having a computerized *ledger* (or state) of the asset, which keeps track of how much of the asset each user owns, and a *protocol*, a set of rules that define how the ledger changes and govern user actions involving the asset.

Yan X Zhang is an assistant professor of mathematics at San José State University. His email address is yan.x.zhang@sjsu.edu.

Communicated by Notices *Associate Editor Steven Sam.*

DOI: https://doi.org/10.1090/noti2365

For most currencies such as the USD (US Dollar), banks already accomplish digital assets by storing the ledger (say, how many USD each user owns) as numbers in some database; people can withdraw/transfer funds by running software that interacts with the database according to protocol. However, we might want digital assets to have additional "democratic" properties:

1. The asset should be *decentralized*: there is no single central authority that could render the protocol unusable by failing and/or purposefully denying service to its users.
2. The asset should be *permissionless*: anyone should be able to become a user of the asset if they follow the protocol.

Banks fail these objectives. For example, if a bank is shut down (via computer failure, government coercion, etc.),

the users lose access to their assets immediately. We call a digital asset that fulfills these additional properties a *cryptocurrency*. The word *blockchain* refers to an instantiation of a cryptocurrency-inspired protocol; it has also become the name of the field of research for these protocols.

Even as a mathematician with little interest in financial speculation, I enjoyed the mathematical content lying underneath the digital gold rush of cryptocurrencies and blockchain. I wrote this specific paper to answer the question, **"what aspects of blockchain might be interesting for a mathematical audience?"** What I mean by "mathematical" also includes fields adjacent to mathematics (basically, fields where practitioners prove theorems):

1. Since a blockchain saves some state across different users, it is natural to study it with *distributed systems* theory, especially *consensus*, which studies: "how do we get n nodes to agree on something, where the network and/or some of the nodes can be faulty?"
2. *Game theory* and *mechanism design* are very relevant as blockchain participants are usually dealing with money. These fields predict user behavior and align the incentives of the users with those of the protocol designer.
3. Many primitives in blockchain depend on understanding what is "easy" or "hard." In particular, blockchain tries to serve large numbers of users while keeping certain actions difficult to prevent attacks on the system. These are the fundamental problems of *cryptography*.
4. Cryptocurrencies (or currencies built on top of cryptocurrencies!) are financial assets. *Mathematical finance* studies these objects and associated phenomena.

To start, let us try to create a cryptocurrency from scratch, leading to the main ideas of Bitcoin. After that section, I will introduce case studies of specific topics. As they are mostly independent, I invite the reader to skip around guided by their own interests and background; for example, "Consensus Meets Blockchain" would be more interesting for a reader who is already familiar with and/or interested in theoretical computer science.

We Build a Cryptocurrency

First, **we need to identify and authenticate users.** We do this with a *digital signature* protocol, which provides pairs of *public* and *private keys*. The public key acts as an "account" (a user can have many public keys) that owns some amount of the asset, and the private key is kept by the owner and can be used to *sign* messages. Digital signatures allow anyone to verify (with the help of the associated public key) that it is extremely unlikely for anyone other than the owner of the matching private key to have signed the message. Messages are to be *broadcast* (sent to all users).

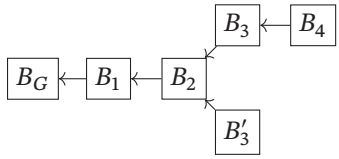


Figure 1. A representative visualization of many blockchains inspired by Bitcoin.

Second, **users need to be able to send assets.** We allow users to create *transactions*, which are messages of the form "x sends amount z to y" (signed by x), where x and y are public keys representing their owners. A *block* is a collection of transactions. It is dangerous for blocks to be "taken out of context" since they can be mutually exclusive (for example, maybe the transaction "x sends z Bitcoin to y" was put in two blocks B and B', so we do not want to double-count that as part of the same history). This motivates having each block refer to a particular state of the blockchain from which the block dictates the next steps to take; in other words, a block is a "state transition function" from a specific state to the next. Putting it together, each block should contain:

1. a reference to a *parent* block (if B_1 is B_2's parent, we say that B_2 is B_1's *child*);
2. a set of transactions;
3. a digital signature of the block publisher.

Complementing this design, at the beginning of our protocol, we can define a *genesis block* B_G, a block encoding an empty state with no parent. Then, we can visualize blocks as the nodes of a graph where each block has a directed edge towards its parent, which is a tree where each block B identifies a unique *chain* $B \to B_k \to B_{k-1} \to \cdots B_G$. Thus, each block B corresponds to a unique state defined by the empty state iteratively updated by the transactions chain in the reverse order $B_G, \ldots, B_k, B$.

Note that our blockchain can *fork*, which happens when we have two different child blocks with the same parent. We say that a block B' is a *descendant* of another block B if there is a directed path from B' to B (resp., from B to B') in this graph. We say that two blocks *conflict* if neither are descendants of the other (equivalently, when the two blocks appear in different paths in a fork). Two conflicting blocks correspond to different and possibly incompatible states; in particular, the *leaf* blocks (blocks with no children) are the candidates for "latest" states of the blockchain. We now have a problem: **"how do we resolve forking?"**

To resolve forking, we need a rule that tells a user which block to consider as the "true state." We need to consider users having potentially different *views* (the set of blocks they have seen), which can happen because of network latency, attackers, etc. Abstractly, we need a *fork-choice rule* that takes as input (V, B), where V is a user's view and B is a block in the view with at least one child, that outputs a

unique child block of B. If we repeat the fork-choice rule starting at the genesis block, we are guaranteed to end up at a leaf block in V, which we call *the head of the chain* of the view. The protocol-following user is to consider the head of the chain as storing the "true history" of their view.

Forks are hard to avoid because they can be created non-maliciously. For example, in Figure 1, the fork between B_3 and B_3' may have occurred because an honest miner published block B_3' off B_2 before seeing B_3 in his view. On the other hand, malicious forking can be very damaging. The typical example of this is a *double-spending* attack, where the attacker x gets the other users to accept a block B that includes a transaction T where x paid some money, obtains some (real-life) service in return for T, and then gets the other users to accept a forked chain starting from the parent of B. If successful, then T is effectively forgotten and the attacker can spend the money again in the future.

With these considerations, it is very important to have a fork-choice rule that is "robust" to forking. We now list a few intuitive approaches to fork-choice rules and why they fail:

1. Favor the "earliest" created child block when we see a fork: timestamps are easy to fake.
2. Take an arbitrary function of the child blocks (represented as integers/strings), and pick the block with the smallest function value: an attacker can eventually fork via a block with an even smaller value.
3. Pick the child block that would eventually lead to the longest chain of blocks: an attacker can just create a very long chain.
4. Have some sort of "vote" on the child blocks: an attacker can create many different accounts to get a higher number of votes.

Many of these attacks can be implemented as *Sybil attacks*, a generic term for when an attacker creates many users to deceive other users about the true state of the chain, including (but not limited to) creating frequent forking. Sybil resistance is especially important for blockchain because we want cryptocurrencies to be permissionless. The protocol designer needs to balance two forces:

1. The easier it is to publish blocks, the more susceptible the blockchain is to forking and Sybil attacks.
2. The harder it is to publish blocks, the slower the rate of transaction processing becomes.

Finally, we have not yet addressed the "leader selection" problem of **"who publishes the blocks?"** Each blockchain addresses all of these issues differently, starting with Bitcoin.

Bitcoin; Proof-of-Work

Bitcoin [Nak19] is the first and still-dominant cryptocurrency and blockchain. Its approach to Sybil resistance is making block publication difficult, with the idea that if **we make it costly to do a useful action, then it is costly to abuse/fake that action**. This approach also addresses the leader selection problem: **the user who performs the difficult action fastest gets to publish blocks**.

First, if we make block publication "difficult," then only certain users with access to computational resources would attempt to publish blocks; these users are called *miners*, though there is no "official" protocol labeling of miners and any user can theoretically be a miner. To incentivize miners to include their transactions and also to perform the "difficult" block publication, users are to include a *fee* in each transaction they create, which goes to the miner who includes the transaction.

Now, our goal is for block publishing to be "difficult" but for verification of a valid block to be "easy" (for the sake of other users). The clever idea that Bitcoin uses is to use a *(cryptographic) hash function*, which is a function which is fast to compute but whose outputs look "essentially random," so that it is hard to find an input that enforces some property on the output. Specifically, if we have the output of such a function h be represented as binary strings, then if we ask a miner to some input x to the function that has an output starting with k 0's for some "difficulty parameter" k, the best thing the miner can do is to repeatedly try random inputs and then getting a satisfactory output roughly $1/2^k$ of the time, which requires an expected time of 2^k rounds. This design is called *proof-of-work* because whenever a miner finds a successful x, it "proves" that the set of all miners did some (random) total amount of work (with expected total computation of 2^k rounds). This repetitive process feels like and is thus referred to as *mining*, which is also why the block publishers are called *miners*. Under this design, the next user to mine a block will essentially be randomly chosen, with each miner having the probability of being selected to publish a block proportional to their computational power.

Proof-of-work[1] leads to the narrative that Bitcoin uses a lot of electricity—people are incentivized to buy machines optimized for computing these hash functions. In exchange, this "wasteful" plan is an elegant response against Sybil attacks: if making a block were free, then the network would be flooded with valid blocks, and the users would be hard-pressed to find consensus.

Specifically, miners follow the following protocol:

1. We define the following fork-choice rule: each user (in their view) is to **consider the leaf block with the highest total difficulty in its chain to be the head of the chain**. This rule is often called the (misleading) *longest chain rule* (it should really be renamed

[1] *We academics are well aware of the "academic proof-of-work": research papers are like blocks where we cite previous papers as (multiple)"parents," where we pay the "work" of peer-review to have papers published, else the archive of academia would be flooded with (more) useless papers.*

"heaviest chain rule"). The miner uses the longest chain rule to obtain their head of the chain B_P.
2. The miner then computes a *difficulty* k, a value dynamically set by the protocol that changes with time to target a specific rate of block generation for the blockchain.
3. The miner collects the transactions they want into a block (usually, the miner just picks a maximum number of transactions in the "pool" of outstanding transactions, sorted by the highest fees). We represent this set of transactions by some string[2] T.
4. The miner looks for a string (typically called a *nonce*, something to be used just once) S such that the concatenation $S||T$ satisfies

$$h(S||T) = \underbrace{00\cdots0}_{k\ 0\text{'s}}\underbrace{b_1b_2b_3\cdots}_{\text{any bits}},$$

so that the first k bits of the hash result are equal to 0. Here, h is a protocol-agreed hash function and k is the aforementioned difficulty.
5. The miner publishes a block B with content $(S||T)$ and parent B_P.
6. The miner receives a *block reward* (new Bitcoin generated and given to the miner directly from the protocol) and the fees from the transactions included in the block.

This protocol aligns the incentives for the blockchain: the users are incentivized to tip the miners with fees to include their own transactions and the miners are incentivized to do the computational work in order to receive the block award and the fees.

Most cryptocurrencies take on the same basic skeleton of Bitcoin. I mention two more (and only two, to not exceed the *Notices* bibliography item limit):

- Ethereum [W+14], the second largest blockchain, also implements a ledger for its cryptocurrency, Ether. However, the blockchain also stores *smart contracts* (executable source code), making it more of a "virtual computer." Thus, the state of Ethereum also contains statements such as "x has smart contract y" on top of statements such as "x has y Ether." Transactions can also be of the form "associate the smart contract y to x" or "run the smart contract associated with x." As a consequence, many types of "software" have been written on Ethereum, including games, decentralized autonomous organizations (where users can put in stake to get voting rights), and decentralized exchanges (where users can trade assets built on top of Ethereum).
- Zcash, based on [SCG+14], is one of the leading blockchains for private transfers. Despite the name, "cryptocurrencies" usually do not offer privacy past the pseudonymity of public keys. In the vast majority of designs (including Bitcoin), anyone can, in principle, keep track of the balances of all addresses. The protocol in [SCG+14] provides one transparent and one private chain. In the private chain, zk-SNARKs (a cryptographic tool that we will introduce later) provide arguments that valid transactions have occurred, without leaking information about the sender, the receiver, or the amount, while being able to detect, e.g., invalid transactions.

Consensus Meets Blockchain

A traditional *consensus protocol*, as found in distributed systems theory, is a set of rules to be followed by a set of *replicas* (users) sending messages to each other and changing internal states in order to agree on some value. Faulty replicas might crash and malicious replicas might attack the system (in cryptocurrencies, this is especially relevant as greed attracts malicious users). The usual approach in the field is to assume that no more than some fraction (usually 1/3) of the replicas are *byzantine* (having completely unpredictable behavior) and then try to have the other *honest* (protocol-following) replicas agree. Specifically, during the execution of the protocol, a replica can be instructed to (irreversibly) *commit* to a value, and the goal is for all **honest** replicas to commit to the same value. Different consensus protocols optimize for different properties, such as:

- *safety*: it is impossible to commit two conflicting[3] values;
- *liveness*: it is always plausible to eventually have everyone commit values;
- *low complexity*: not too many (and/or too long) messages are sent over the network.

There are a few differences between the usual assumptions of consensus and blockchain, such as:

- The typical use case of a distributed system is a database with a small number of replicas (such as 4). The permissionless nature of blockchain (in particular, a byzantine user can pretend to be many users) operates under higher scale.
- The non-byzantine replicas in distributed systems are typically "obedient" nodes; the non-byzantine users in blockchain are typically "rational" human actors who must be incentivized (such as by fees) to perform costly actions.

[2] *For the big picture it is sufficient to assume that T is just a concatenation of strings encoding individual transactions; the technical implementation involving Merkle trees is slightly different.*

[3] *This word depends on context. If the value is, e.g., a number, "conflicting" simply means "different." In more involved contexts, such as when the values represent the state of some machine, "conflicting" would mean representing incompatible states, analogous to conflicting blocks in different forks in blockchain.*

In most consensus protocols, there is a *leader* replica proposing values and "phases" of replicas acknowledging these values; the phases need to be carefully designed to avoid well-coordinated attacks by byzantine actors. Practical Byzantine Fault Tolerance (PBFT) [CL99] is representative of this school, guaranteeing safety when less than 1/3 of replicas are byzantine. It mainly[4] operates as follows:

1. Pre-prepare phase: a leader proposes a value and broadcasts it as a message.
2. Prepare phase: if a replica receives a value, it broadcasts a *prepare* message.
3. Commit phase: if a replica receives matching prepare messages from 2/3 of all replicas, it broadcasts a *commit* message; it internally *commits* the value if it receives matching commit messages from 2/3 of all replicas.

It is interesting (and exciting) that Bitcoin, in particular proof-of-work, provides a new approach to consensus that basically avoids the concepts of leaders or phases. This is why Bitcoin's proof-of-work design is also called *Nakamoto Consensus*, despite its motivation as a Sybil resistance mechanism.

Nakamoto Consensus does not have safety in the traditional consensus sense, though it is natural to interpret it as having "probabilistic safety": blocks on the longest chain are **likely** to be "safe," but it is plausible for any block to be "undone" if a chain with higher difficulty appears. As a case in point, [ES14] analyzes *selfish mining attacks*. The main idea is that if the attackers have a "secret chain" consisting of a valid chain of unpublished but successfully "mined" (that is, with the correct nonces computed) blocks, they can delay the publication of the chain until the rest of the miners catch up. This wastes the other miners' progress, so the attackers end up with more relative revenue. The tradeoff is that this delay may cause them to waste their own publication if the rest of the miners publish a competing block quickly.

We now give some details. In [ES14]'s setup, the attackers have total computation power (interpreted as a fraction of total computational power of all users) $\alpha \in [0, 1/2)$. With Bitcoin's design, this means that at any moment, the next block is mined (but not necessarily published) by the attackers with probability α and is mined (and immediately published) by a non-attacker miner with probability $(1 - \alpha)$, with the attackers being a minority. The authors then model the attacker's strategy as a Markov chain with states:

1. $0'$: the chain is forked, with one leaf block published by the attackers and one not (the authors assume that a non-attacker would mine on one of these two blocks randomly via some fixed Bernoulli distribution, while the attackers would mine on the attacker-published block);
2. 0: there is a unique public head of the chain B; the attackers have no mined-but-unpublished chains that would be longer than B if published.
3. $i \in \mathbb{N}$: there is a unique public head of the chain B; the attackers have a mined-but-unpublished chain (extending B or forking one of its ancestors) such that, if published, would be i blocks ahead of B.

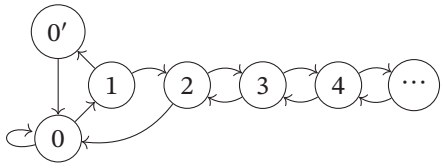


Figure 2. The Markov chain for selfish-mining.

State transitions happen when a block is mined, with nuances depending on the state and whether the attackers or non-attackers mine the block. For example, if a non-attacker publishes a block at state $i \geq 3$, the state moves to $i - 1$ since the attackers' unpublished chain's lead over the public chain has decreased by 1 block. However, if a non-attacker publishes a block at state 2, the proposed strategy is having the attackers publish their two blocks immediately, skipping past[5] state 1 to state 0. See Figure 2 (essentially [ES14, Figure 1]) for a visualization.

By computing the transition probabilities and keeping track of the probabilistic block rewards, the authors of [ES14] quantify the attack in terms of additional revenue to the attackers. This work is one of the first mathematical approaches to Bitcoin's safety. In a similar vein, [SZ15] analyzes Bitcoin's susceptibility to double-spending attacks and proposes GHOST, a replacement for the longest chain rule with nicer properties.

The main idea of *proof-of-stake*, an alternative design paradigm to proof-of-work, is to disincentivize bad behavior by threatening to *slash* (take away) a user's *stake* (money that they commit to the protocol). In effect, proof-of-stake **tries to obtain the benefits of proof-of-work through game theory**, replacing "do work to earn the right to an action" with "do an action for free, but be susceptible to losing money." This is analogous to allowing a tenant to live in an apartment but seizing their deposit if they commit bad behavior. Proof-of-stake protocols are highly desired

[4] *I am omitting some details, such as the"view change" operation for when a leader takes too long and the other replicas select a new leader. This is not just a matter of convenience; the protocol needs to protect against potentially byzantine leaders.*

[5] *The authors do not explicitly justify why the attackers do not stay at state* 1*. My interpretation is that if they do, they have two block rewards at risk if the non-attackers catch up (which advances the state to* $0'$*, from which the attackers are likely to lose the rewards). In contrast, if the attackers arrive at state* 1 *from state* 0 *instead of* 2*, only a single block reward is at risk if the attackers stay at* 1*, so the cost-benefit analyses differ. It may be helpful to envision an additional state to disambiguate the two state* 1*'s.*

because they avoid "useless" work, similar to the pursuit of "clean energy."

Proof-of-stake protocols tend to be similar to traditional consensus protocols.[6] They also have their own weaknesses; in particular, [BCNPW19] shows that after making some assumptions, every "longest-chain variant" proof-of-stake protocol contains one of two properties, either of which makes the protocol vulnerable to a different attack. This result has an "impossibility theorem" flavor reminiscent of Arrow's Theorem [Arr50] from voting theory.[7]

With proof-of-work inspiring, e.g., Nakamoto Consensus and proof-of-stake inspiring, e.g., HotStuff[8] [YMR+19], it is natural to ask the following question.

> **Question 1.** Using concepts blockchain popularizes such as proof-of-work and proof-of-stake (and also VDFs, SNARKs, etc. which we will see later), can we innovate designs in consensus and other subfields (for example, *self-stabilization* studies how replicas in a potentially corrupted state can guarantee returning to a "good" state) of distributed systems?

We end with a final voting analogy—when we do not have a central authority, the users need to have some sort of "voting." Standard consensus protocols such as PBFT use acknowledgment messages as votes to ensure safety, where voting power is proportional to the number of users (assuming there are not enough byzantine replicas to break protocol by, e.g., voting twice). Sybil attacks in a permissionless blockchain make such systems ineffective, so we must find an alternative to "voting with number of users." Indeed, proof-of-work is "voting with computational work" and proof-of-stake is "voting with staked money"!

Mechanism Design; Gas Markets

Mechanism design and game theory are everywhere in blockchain since we must always consider the incentives of the various parties. For example, game theorists study *auctions* where participants bid money to win one or more rewards. Most proof-of-work blockchains involve transaction creators bidding fees to win the service of being included in a block, so they can naturally be modelled as auctions.

An Ethereum transaction includes a *gas limit* (in abstract units of *gas*), which estimates how complex the operation is for the Ethereum "computer," and a *gas price*, which is the amount the user is willing to pay per unit of gas. For example, a simple Ether transfer has a gas limit of 21000. If the user bids a gas price of 2×10^{-7} Ether for such a transaction, then the user would pay the product 0.0042 Ether to a miner including the transaction in a block.

From the miner's perspective, the miner is incentivized to just take the maximum number of transactions they can fit, sorted by the highest gas prices. This means the highest gas price bidders "win" and then pay exactly what they bid, a design called a *first-price auction*. Finding optimal strategies is difficult in a first-price auction, especially with limited information. In practice for Ethereum (everything discussed so far also applies to Bitcoin and similar proof-of-work designs), participants often overbid, causing sharp spikes in gas prices.

In auction theory, such situations can often be improved by *second-price auctions* which enjoy better theoretical properties. In our case, "second-price" actually means that each transaction included in a block would pay the miner the **lowest** bid among them. However, blockchain miners can do things traditional auctioneers cannot; they can include fewer transactions or create "fake" transactions themselves. For example, the blockchain might want a miner to include the maximum of three transactions in each block, but the top three gas prices have values $\{10, 2, 1\}$, which would offer value 3 to the miner including all of them. Then the miner is incentivized to include just the first transaction or make two fake transactions at 10 each (e.g., to pretend they are processing the maximum number of three transactions), both with a profit of 10. In summary, **auction theory works differently under blockchain, so we need new ideas**.

EIP-1559 is a proposal for revising the Ethereum gas market. Its main idea is to incentivize transaction creators to bid close to a default value known as a "base fee," so the user experience is closer to, e.g., Amazon where a buyer simply pays a posted price. Counterintuitively, it also asks the base fee portion to be destroyed permanently instead of being given to the miner (otherwise, the miner could collude with users to simulate a first-price auction).

1. From a state of the protocol, one can deterministically compute a *base fee*. The idea is that the base increases under high trading activity and decreases under low trading activity.
2. A transaction replaces the concept of the single *gas price* by a *tip* and *fee cap*. If a transaction has a tip t, fee cap f, and gas limit g is included in a block with base fee r, then the creator of the transaction pays $g \times \min(r + t, f)$.

[6] *A heuristic reason for this similarity is that if we incentivize (via rewards and punishment) "rational" validators to act as "obedient" honest nodes in the traditional protocol, we reduce to a similar setup as a traditional consensus protocol.*

[7] *Tangent: one interpretation of Arrow's Theorem is "if we do not want a dictatorship, then our voting system will have some inadequacies (which dictatorships avoid)." Analogously, if we do not want a central authority, then our blockchain will have inadequacies (which central authorities avoid)!*

[8] *HotStuff* [YMR+19] *is a consensus protocol modelled after proof-of-stake blockchain protocols; the paper also presents an abstract framework that generalizes several existing protocols.*

3. The miner receives $g \times \min(t, f - r)$; note this corresponds to the base fee being destroyed.

The report [Rou20] contrasts this design with the status quo. The positive predictions are that there would be variance reduction, easier user discovery of optimal fees, resistance against fake transactions, and (beneficial) deflation. However, [Rou20] argues that the average transaction fee will not change significantly and in periods of high demand the situation can revert to that of a first-price auction, where users might start inflating their tips.

> **Question 2.** [Rou20] mostly analyzes single blocks. What can we say about EIP-1559 dynamics (base fee changes, collusion, etc.) over longer time periods?

Verifiable Delay Functions

With all this talk of proof-of-work and proof-of-stake, a fun mathematical tangent happens with "proof-of-time," which would be a way to prove that a certain amount of time[9] has elapsed. This is almost what happens in proof-of-work, though a miner running 10 mining machines would on average use 1/10 the time to mine a block.[10] Thus, what we want is a family of functions such that:

1. given a parameter T, we can generate a specific function F such that it takes around T time to compute, and this cannot be easily parallelized;
2. after computing F, we obtain a short proof that allows others to verify the computation very quickly.

If both conditions are satisfied, we call such a function a *verifiable delay function (VDF)*.

Why would we want such a function? It is actually not to replace proof-of-work.[11] The primary application of VDFs to blockchain is generating **randomness** on the blockchain, a problem much trickier than it appears to be. As an example, suppose we want to generate a random number, but the result may benefit some users and hurt others (such as a lottery, settling some bet on the blockchain, or choosing the block proposer in proof-of-stake). One naive way to generate the randomness would be to, e.g., hash the data of a block and look at its first bits. However, a miner who mines B will immediately know whether the result benefits her. This means she might decide to *censor* (not publish) the block, spoiling our goal.

If we instead feed the data in the block to a VDF before hashing, then the miner cannot figure out the answer in advance. If the miner still wants to publish the block (e.g., for the block reward), the miner would mine the block. In summary, **we have incentivized fair randomness by making sure that the last actor cannot peek ahead at the answer.** It is also important for the function to be efficiently verifiable so that the other users of the chain can be convinced that the function was computed correctly.

Two of the earliest VDFs by Wesolowski [Wes20] and Pietrzak [Pie19] use the following idea: if we have a group element g of some finite group G of **unknown** order (a concept we discuss more in the next section), then we do not know of a better way of computing g^{2^T} than doing T repeated squarings starting from g. Specifically:

1. To set up the VDF (with domain X), find an abelian group G of unknown order and a hash function $H : X \to G$. Then save the parameters (G, H, T).
2. To evaluate the VDF on $x \in X$, output $y = [H(x)]^{2^T}$ and a proof P.

To give a specific example on how P is generated, we show Wesolowski's [Wes20] method. Instead of showing P directly, first we outline how a prover and a verifier can engage in an *interactive proof* (we will discuss interactive proofs in more detail soon) protocol as follows:

1. First pick a T. As an example, assuming 10^9 operations a second, a 5-minute VDF might have T around 2^{40}.
2. The prover and verifier compute $g = H(x) \in G$. The prover then computes $y = g^{2^T}$ in $O(T)$ time.
3. The verifier gives the prover a random prime q (say, sampled uniformly around a certain size such as 2^{512}).
4. The prover computes $s, r \in \mathbf{Z}$, where $2^T = sq + r$ with $0 \le r < q$, and sends $P = g^s$ to the verifier. As s is approximately $2^T/q$, computing g^s takes $O(T)$ operations, a linear overhead to computing the VDF itself.
5. The verifier computes $2^T \pmod q$ (which should equal r) and checks that $P^q g^r = y$. Computing $2^T \pmod q$, P^q, g^r takes $O(\log(T))$, $O(\log(q))$, $O(\log(r))$ operations, respectively, all of which are fast.

Finally, we can remove the interactions required with the well-known *Fiat-Shamir heuristic* (using randomness to simulate verifier choices). Instead of asking the verifier for q, the prover and verifier can both generate q by using some (public) hash function that takes (G, x, y, T) as input and returns a random prime q. Then the prover can just give P to the verifier **as if she received q from the verifier in the interactive proof**, and the verifier performs an identical verification with the same q.

[9] *Yes, there are also terms such as* proof-of-space, proof-of-authority, *and even* proof-of-useful-work *floating around.*

[10] *As in high school physics, work is (hashing) power multiplied by time.*

[11] *In proof-of-work, if Alice has* 60% *of the resources in the world and Bob has the other* 40%*, then Alice publishes* 60% *of the blocks. However, if we naively replace the proof-of-work by computing a VDF, then Alice would publish near* 100% *of the blocks, because her VDF-machine is simply faster than Bob's. This promotes a monopoly, which is against the decentralizing goal of blockchain.*

Question 3. What are other ways to use VDFs creatively? For example, VDFs can be used to, e.g., create an unbiased "random beacon" that broadcasts randomness at set time intervals, which can be useful for situations beyond blockchain.

Cryptography and Number Theory

Most of the cryptography blockchain research is concentrated in the fairly specialized "program" of interactive proofs and SNARKs (which I cover in the next sections), but there are occasional ad-hoc problems that may be interesting to non-specialists, especially number theorists. I now give a couple of examples.

First, constructing groups of unknown order is useful for several applications; we already saw it in VDFs, and it also comes up in SNARKs (in particular DARKs). One natural idea is an *RSA group*, where a trusted authority multiplies two big primes to obtain $N = pq$ and then "throws away" p and q. Given N, we can do computations in the multiplicative group $(\mathbb{Z}/N\mathbb{Z})^*$, but we do not know the order of this group without the factorization of pq. In the decentralized context of blockchain, however, such constructions involving a central authority are usually undesirable.

The more popular constructions of groups of unknown order involve *ideal class groups* of imaginary quadratic fields, which I now present (omitting some technical details). In short, an *imaginary quadratic field* is an algebraic extension

$$\mathbb{Q}(\sqrt{d}) = \{x + y\sqrt{d} | x, y \in \mathbb{Q}\},$$

where d is a negative square-free integer. We define the *discriminant* Δ to be d if $d \equiv 1 \pmod 4$ and $4d$ otherwise. Each such field K creates a *ring of integers* $\mathcal{O}_K$. The *ideal class group* $h(\Delta)$ is the quotient group of the group of non-zero fractional ideals of $\mathcal{O}_K$ modded out by the principal fractional ideals. The order of $h(\Delta)$, also called the *class number* of K, is a number that is believed to be difficult to compute for sufficiently large Δ (assuming $\Delta \equiv 1 \pmod 4$), which means we can generate an ideal class group randomly and be fairly sure that the group order is hard to compute. For additional information, see [Wes20] for the proposal of using ideal class groups for VDFs, [BFS20] for the application to DARKs, or [DGS20] for a mathematical review of these constructions and a proposal of using Jacobians of hyperelliptic curves ([Lee20] gives counterpoints to the proposal).

Question 4. How easy is it to get "partial information" from an ideal class group? For example (a specific question from Dan Boneh), given d (equivalently Δ), how hard is it to determine if $|h(\Delta)|$ is divisible by 3 or some other small prime?

Second, in blockchains (and more generally) it is often desired to have efficient and secure *multiparty computation (MPC)* where several users can collectively perform a computation requiring no trust in each other. For example, a blockchain may require a user to provide randomness. When several users want to act as a single user, they can use an MPC to jointly compute the randomness.

MPCs usually rely on some deterministic functions that simulate randomness (like hash functions, but with more requirements). One promising proposal uses the *Legendre symbol* $(\frac{x}{p})$, which equals 1 if $x \in (\mathbb{Z}/p\mathbb{Z})^*$ is a *quadratic residue* (equal to the square of some element) modulo the prime p and -1 otherwise; it obeys many nice properties and is easy to compute. Given a secret key $k \in \mathbb{Z}/p\mathbb{Z}$, define

$$L_{p,k}(x) = \left(\frac{x+k}{p}\right).$$

See `https://legendreprf.org/bounties` for references and bounties (for both theoretical and computational results) related to this construction.

Question 5. Given a small, say $O(\log(p))$, number of different evaluations of $L_{p,k}(x)$, is it possible to recover k with high probability?

Interactive Proofs; SNARKs

Interactive proofs is a subfield of cryptography that is evolving rapidly due to blockchain. Informally, an *interactive proof* is a protocol for a *prover* to convince a *verifier* (we can think of both as people armed with computers) of some fact by sending messages over potentially many rounds. A nice example of an interactive proof relevant to blockchain is the "sumcheck" protocol from [LFKN92]. In this protocol, the prover tries to convince the verifier that the prover performed the computation $\sum_{\alpha \in \{0,1\}^n} P(\alpha) = \beta$, where P is a known n-variate polynomial of low degree over some finite field F.

1. The protocol has n rounds. In round $i \in \{1, 2, \dots, n\}$, the prover sends the polynomial

$$p_i(X_i) = \sum_{\alpha \in \{0,1\}^{n-i}} P(c_1, \dots, c_{i-1}, X_i, \alpha),$$

where the c_i are field elements to be received from the verifier. At the beginning of round i, the prover knows c_1 through c_{i-1}. In particular, at the beginning of the

first round the prover does not have any c_i and the expression also involves no c_i.

2. After receiving the polynomial in round i, the verifier computes $S_i = \sum_{X_i \in \{0,1\}} p_i(X_i)$; then:
   (a) if $i = 1$, the verifier checks $S_1 = \beta$;
   (b) if $i > 1$, the verifier checks $S_i = p_{i-1}(c_{i-1})$.
   Afterwards, the verifier gives a random $c_i \in F$ to the prover.
3. After all n rounds, the verifier checks if $P(c_1, \dots, c_n) = p_n(c_1, \dots, c_n)$.

This algorithm is essentially a commitment to the computation which is narrowed down via the challenges c_i given by the verifier one variable at a time. Its correctness is based on the fact that it is extremely unlikely (assuming P is of low degree) that the prover can "keep up the act" if the prover did not in fact have such a computation. This idea is revisited frequently in deeper parts of blockchain research.

In blockchain, the main interactive proof systems people study are *(zk)-SNARKs*, or *(zero-knowledge) succinct non-interactive arguments of knowledge*. Informally, the important terms are:

- *Zero-knowledge*: the proof is done in a way that the verifier is convinced that the prover knows some information s, but the verifier does not gain any knowledge about s. For example, it is possible to create a proof that two graphs are isomorphic without giving away any information about the actual isomorphism. Note that a digital signature is exactly a zero-knowledge proof that the signer knows the secret key.
- *Succinct*: the proof generation time is fast (usually linear in computation length) and the verification time is fast (usually sublinear in computation length). Note that the latter requirement implies sublinear proof size (since it takes time for the verifier to read the proof).
- *Non-interactive*: the total communication is one message sent by the prover. The name is unfortunate as most interactive proofs we see in practice are technically "non-interactive interactive proofs" thanks to the Fiat-Shamir heuristic (recall our VDF presentation).

SNARKs were introduced into blockchain via [SCG+14] to introduce private transactions via the zero-knowledge property. However, the more important[12] goal of SNARKs is providing **succinct verified computation**. A blockchain fundamentally tries to save a shared state, which is the result of a sequence of state transition computations (such as blocks). Specifically, we may want to prove statements of the form "when the user x (the prover) performed some computation C from state S, using her own secret information w (say x's secret key), the new state is $S' = F((S, C), w)$," where F is a function encoding state transition. If we can provide easily verifiable (succinct) proofs of such computations, then a blockchain can just put the **proofs** of the computations "on-chain" (as transactions) and have much of the computation be done "off-chain" instead.

Given an arbitrary computation, a SNARK typically converts the computation from "circuit" form (composed of logic gates) into polynomial equations via a process called "arithmetization." Since we can combine many linear equations into a single polynomial equation, polynomials[13] are very important in scaling blockchains. SNARKs ignited a period of rapid cryptography research that resulted in many variations. I will just name a couple:

- STARKs [BSBHR19], which are *transparent*, meaning they do not require a trusted setup like the [SCG+14] SNARKs. As a bonus, STARKs are believed to be post-quantum secure. However, STARKs have much larger proofs than SNARKs due to the lack of preprocessing.
- DARKs [BFS20], which use groups of unknown order to achieve transparency with better asymptotic proof size and similar verification time compared to STARKs, but loses some "in-practice" efficiency and loses post-quantum security. As a theoretical contribution, the paper [BFS20] also gives a framework which can be used to express the other proposed SNARKs, including STARKs.

Question 6. Can we design other post-quantum SNARKs besides STARKs? In general, there is much interest in designing SNARKs which push any metrics (proof size, verification speed, post-quantum resistance, etc.) farther.

Recursive SNARKs and Pairs of Elliptic Curves

While it is old news by now, I have always found it pleasant that something as "pure" as elliptic curves have found their way into cryptography. In blockchain, a particularly cute extension of this story is the appearance of *amicable pairs of elliptic curves* in the study of recursive SNARKs.

In standard blockchains, a new user must download the entire history to be completely sure that what they have is valid. A *recursive SNARK* design uses SNARKs recursively to prove statements of the form

$$F(F(\dots F(F(S_0, w_0), w_1, \dots))) = S,$$

which says "the initial state S_0, which was correctly iteratively updated until it now equals S." Such guarantees

[12] *Case in point: one of the most important proposals for scaling Ethereum is called "zk-Rollup" because of the usage of SNARKs in combining transactions and saving space; what is important here is entirely the succinctness of the SNARK and the "zk" part is actually irrelevant.*

[13] *The choice of polynomials is what makes ideas such as the aforementioned sumcheck algorithm important.*

would greatly reduce the total amount of data that users would need to hold. The key step of a recursive SNARK is to **create a different SNARK that verifies the computation of verifying the first SNARK.**

The construction of SNARKs in [SCG+14] is associated with a triple (E, q, r), where E is an *elliptic curve* over the finite field $\mathbb{F}_q$, r is the prime order of some subgroup in E, and (E, q, r) is *ordinary* and *pairing-friendly*. Sketching the essential details:

1. An *elliptic curve E over $\mathbb{F}$* is a group of solutions to a particular family of polynomial equations over some field $\mathbb{F}$. For us, we always have $\mathbb{F} = \mathbb{F}_q$ for some prime q. For cryptographic operations, typically we focus our attention on some subgroup of prime order r within E, so we always specify (E, q, r) in conjunction.
2. The *embedding degree of* (E, q, r) is the minimal natural number k such that $r|(q^k - 1)$. It only depends on r and q, though those are always in context of E.
3. We call (E, q, r) *pairing friendly* if $r > \sqrt{q}$ and the embedding degree of (E, q, r) is not too big.[14]
4. We call (E, q, r) *ordinary* if $\gcd(q + 1 - |E|, q) = 1$, a technical condition (again, this depends only on E and q, but we always consider the triple in our applications) which guarantees that the embedding degree of (E, q, r) is not too small.[15]

With these assumptions, we can create a SNARK which proves that the prover performed some computation in $\mathbb{F}_r$ arithmetic, while the SNARK itself involves $\mathbb{F}_q$ arithmetic. Thus, if we have an elliptic curve E over $\mathbb{F}_q$ encoding computations in $\mathbb{F}_r$, to perform the key recursive SNARK step, we want to verify computations done in $\mathbb{F}_{q'}$, meaning we want a different elliptic curve E' over some $\mathbb{F}_{q'}$ which has an order q subgroup.

Thus, to perform this operation ad infinitum, we want an infinite walk in the *pairing friendly graph of elliptic curves*:[16]

- Each vertex of the graph is a pairing friendly and ordinary (E, q, r).
- We have an edge $(E, q, r) \to (E', q', r')$ if $q = r'$.

The easiest way to create an infinite walk is via a self-loop/1-cycle, but this is impossible since we cannot have $q|(q^k - 1)$. It is then natural to ask the following question.

Question 7. Can we find/classify pairs (or longer cycles) of ordinary pairing-friendly elliptic curves E_1 over $\mathbb{F}_q$ and E_2 over $\mathbb{F}_r$ such that $|E_1| = r$ and $|E_2| = q$, ideally with larger embedding degrees than 6?

As of now, the only known family for 2-cycles is a family of pairs of *MNT cycles* of (lower than desired for us) embedding degrees $(6, 4)$. [CCW19] proves that cycles of MNT curves must be of lengths 2 or 4, with embedding degrees $(6, 4)$ or $(6, 4, 6, 4)$, so this direction is a dead end. They also show that 2-cycles for certain pairs of small embedding degrees are not possible, though slightly bigger degrees remain open.

Decentralized Finance; CFMMs

Broadly speaking, finance happens when there are *(financial) assets*. Assets include money, gold, stocks, or more complicated instruments, such as *derivatives*, contracts containing logic depending on other assets. Assets generate *markets*, mechanisms/protocols where they can be traded. The amount of assets available on a market to trade is called *liquidity*. Classical mathematical finance studies these financial objects. *Decentralized finance (DeFi)* does the same, except with financial objects that exist on a blockchain.

Many of these concepts are fairly close to traditional financial concepts, and can be studied similarly. For example, many *tokens* (currencies built on top of a blockchain, usually Ethereum) are analogous to pachinko balls (assets bought with money that can be used to perform specific tasks) or stocks (assets that can pay money later or be used as governance). Also, *stablecoins* are tokens that are designed to stay at a fixed value "in the real world," like how Hong Kong pegs its currency to a constant multiple of the USD. However, some concepts are truly unique to DeFi. I introduce one now.

Traditionally, a *market maker* is a person working for a stock exchange who adjusts (based on market conditions) assets available for market participants. The market maker "balances" the exchange (by creating positions where other participants can both buy and sell) and makes money "from the spread" (by buying the same assets at slightly lower prices and selling at slightly higher prices). Blockchain allows *automated market makers*, which are smart contracts that provide this type of service with no (or optional) human oversight.

Uniswap (`https://uniswap.org/`) is such an automated market maker. Uniswap is a smart contract that holds an amount X of one asset and Y of another, which allows people to trade between the two assets as long as they keep XY to be some "constant" (not accounting for a

[14] *Depending on source, I have seen this to mean bounded above by some constant times* $\log(r)$ *or a straight constant such as* 50. *The main idea here is that if it is too small, we cannot compute* pairings *efficiently on them, which is needed for the underlying cryptography.*

[15] *Small embedding degree leads the discrete logarithm problem to be more easily broken, so it is also not desirable. A non-ordinary or* supersingular *curve can only have embedding degree at most* 2 *over a prime field* $\mathbb{F}_q$.

[16] *As defined in* `https://coinlist.co/build/coda/pages/theory`.

trading fee). Uniswap is extremely successful: it provides more than 2 billion dollars of liquidity, completely automated.

Formally, [AC20] generalizes Uniswap (and other similar protococols) to *constant function market makers (CFMMs)*. These are defined by a *trading function* $\phi : \mathbb{R}_+^n \times \mathbb{R}_+^n \times \mathbb{R}_+^n \to \mathbb{R}$ and a set of *reserves* $R \in \mathbb{R}_+^n$. R encodes the amounts of n types of coins a smart contract holds. Besides R, the trading function takes as input $\Delta \in \mathbb{R}_+^n$ (a trader's offer to the contract) and $\Gamma \in \mathbb{R}_+^n$ (what the trader wants in return). The CFMM then accepts the trade (Δ, Γ) if and only if $\phi(R, \Delta, \Gamma) = \phi(R, 0, 0)$; alternatively, it accepts a trade if and only if the trading function is kept **constant**, hence the name. Afterwards, the coins are exchanged, and R now equals $R + \Delta - \Gamma$.

In this framework, Uniswap is an $n = 2$ CFMM with

$$\phi(R, \Delta, \Gamma) = (R_1 + \gamma\Delta_1 - \Gamma_1)(R_2 + \gamma\Delta_2 - \Gamma_2),$$

where $(1 - \gamma)$ is the percentage fee of a trade. Balancer (`https://balancer.finance/`) and Cycle (`https://www.curve.fi/`) are two more automated market makers which can be expressed as CFMMs. CFMMs are a great starting point for DeFi research due to their balance between mathematical clarity, simplicity, and actual usage.

The work [TW] is an example of traditional mathematical finance applied to (a slight variation of) Uniswap. The work takes the perspective of the *liquidity provider (LP)*, which we can think of as the[17] person who owns the two assets held by Uniswap. If X_t and Y_t are the amounts of the two assets held by the smart contract at time t, then the *wealth* of the LP is defined by $X_t + Y_t S_t$, where S_t is the *external price*, or how much a unit of the second asset is worth in terms of the first. For example, the LP could be holding X_t USD and Y_t Ether in Uniswap, while S_t is the (current) price of Ether in USD. Then the wealth is a measure of how much the LP's assets are valued in USD. The authors make a few assumptions:

1. S_t follows a discrete simple random walk with (multiplicative) steps $e^{\pm\delta}$ for some constant δ;
2. the "outside world" is modeled by a single trader (whom we call OT for "outside trader") with unlimited assets who trades with Uniswap if and only if the trade makes them money; e.g., if they can trade x of the first asset for y of the second asset such that $yS_t > x$, they do.

The authors' goal is to find

$$\lim_{t\to\infty} \frac{\mathbb{E}[\log(X_t + Y_t S_t)]}{t},$$

the asymptotic geometric return of the LP's wealth. Their main idea is defining $S_t^* = X_t/Y_t$, the *implicit price*; one way to motivate this quantity is to consider the feeless $\gamma = 1$ limit, where the OT's strategy essentially becomes keeping the ratio of X_t/Y_t equal to S_t. Then, the authors note that $M_t = \log(S_t/S_t^*)$ follows a random walk where:

1. the state space has $(2k+1)$ states ($k \in \mathbb{N}$ being a derived quantity from γ and δ) labeled $\{s_i | -k \le i \le k\}$;
2. at each step we go from s_j "up" to s_{j+1} with probability p or "down" to s_{j-1} with probability $(1 - p)$ for some constant p;
3. if we attempt to go "up" at s_k, then a trade happens: the LP changes X_t to $X_t(e^{\frac{\delta}{\gamma+1}})$ and Y_t to $Y_t(e^{\frac{-\gamma\delta}{\gamma+1}})$ and M_t **stays at** s_k. A similar result happens if we go "down" at s_{-k}.

We "hit the boundary" exactly when the OT can make money by trading with the LP; the changes to X_t and Y_t are obtained by solving for the optimal trade amount from the OT's perspective. The random walk has the very nice property that after resolving each trade, the resulting S_t and S_t^* make it so that M_t is unchanged, which has no obvious reason of being true! Using this and other properties of the random walk, [TW] derives a closed formula for the desired asymptotic return.

Question 8. Can we generalize this type of analysis (wealth growth, liquidity growth, expected trade volume, etc.) to other CFMMs? Do we always get a random walk with nice properties, or did we just get "lucky" with a particularly nice M_t?

ACKNOWLEDGMENTS. I thank Vitalik Buterin, Jing Chan, Justin Drake, Dankrad Feist, Petr Kuznetov, Steven Sam, Yi Sun, Andrei Tonkikh, and Barry Whitehat for valuable conversation. I thank two anonymous referees for useful feedback. I disclose that I have worked as a research consultant for the Ethereum Foundation.

[17] *In reality, there would be potentially many LPs all contributing different amounts of liquidity.*

References

[AC20] Guillermo Angeris and Tarun Chitra, *Improved price oracles: Constant function market makers*, arXiv:2003.10001, 2020.

[Arr50] Kenneth J. Arrow, *A difficulty in the concept of social welfare*, J. Political Economy **58** (1950), no. 4, 328–346.

[BCNPW19] Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S. Matthew Weinberg, *Formal barriers to longest-chain proof-of-stake protocols*, Proceedings of the 2019 ACM Conference on Economics and Computation, 2019, pp. 459–473.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec, *Transparent SNARKs from DARK compilers*, Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2020, pp. 677–706.

[BSBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev, *Scalable zero knowledge with no trusted setup*, Annual International Cryptology Conference, 2019, pp. 701–732.
[CCW19] Alessandro Chiesa, Lynn Chua, and Matthew Weidner, *On cycles of pairing-friendly elliptic curves*, SIAM J. Appl. Algebra Geom. **3** (2019), no. 2, 175–192, DOI 10.1137/18M1173708. MR3934100
[CL99] Miguel Castro and Barbara Liskov, *Practical byzantine fault tolerance*, Proceedings of the Third Symposium on Operating Systems Design and Implementation, 1999, pp. 173–186.
[DGS20] Samuel Dobson, Steven D. Galbraith, and Benjamin Smith, *Trustless groups of unknown order with hyperelliptic curves*, IACR Cryptol. ePrint Arch. **2020** (2020), 196.
[ES14] Ittay Eyal and Emin Gün Sirer, *Majority is not enough: Bitcoin mining is vulnerable*, International Conference on Financial Cryptography and Data Security, 2014, pp. 436–454.
[Lee20] Jonathan Lee, *The security of groups of unknown order based on jacobians of hyperelliptic curves*, IACR Cryptol. ePrint Arch. **2020** (2020), 289.
[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan, *Algebraic methods for interactive proof systems*, J. Assoc. Comput. Mach. **39** (1992), no. 4, 859–868, DOI 10.1145/146585.146605. MR1187215
[Nak19] Satoshi Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, Manubot, 2019.
[Pie19] Krzysztof Pietrzak, *Simple verifiable delay functions*, 10th Innovations in Theoretical Computer Science, LIPIcs. Leibniz Int. Proc. Inform., vol. 124, Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2019, pp. Art. No. 60, 15. MR3899854
[Rou20] Tim Roughgarden, *Transaction fee mechanism design for the ethereum blockchain: An economic analysis of eip-1559*, arXiv:2012.00854, 2020.
[SCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza, *Zerocash: Decentralized anonymous payments from bitcoin*, 2014 IEEE Symposium on Security and Privacy, 2014, pp. 459–474.
[SZ15] Yonatan Sompolinsky and Aviv Zohar, *Secure high-rate transaction processing in Bitcoin*, Financial Cryptography and Data Security, Lecture Notes in Comput. Sci., vol. 8975, Springer, Heidelberg, 2015, pp. 507–527, DOI 10.1007/978-3-662-47854-7_32. MR3395041
[TW] Martin Tassy and David White, *Growth rate of a liquidity provider's wealth in $xy = c$ automated market makers*.
[Wes20] Benjamin Wesolowski, *Efficient verifiable delay functions*, J. Cryptology **33** (2020), no. 4, 2113–2147, DOI 10.1007/s00145-020-09364-x. MR4163449
[W+14] Gavin Wood et al., *Ethereum: A secure decentralised generalised transaction ledger*, Ethereum project yellow paper **151** (2014), no. 2014, 1–32.
[YMR+19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham, *HotStuff: BFT consensus with linearity and responsiveness*, Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, 2019, pp. 347–356.

Yan X Zhang

Credits

Opening image is courtesy of Suebsiri via Getty.
Figures 1 and 2 and photo of Yan X Zhang are courtesy of Yan X Zhang.