Core Concepts, Challenges, and Future Directions in Blockchain: A Centralized Tutorial

JOHN KOLB, MOUSTAFA ABDELBAKY, RANDY H. KATZ, and DAVID E. CULLER,
University of California – Berkeley, USA

Blockchains are a topic of immense interest in academia and industry, but their true nature is often obscured by marketing and hype. In this tutorial, we explain the fundamental elements of blockchains. We discuss their ability to achieve availability, consistency, and data integrity as well as their inherent limitations. Using Ethereum as a case study, we describe the inner workings of blockchains in detail before comparing blockchains to traditional distributed systems. In the second part of our tutorial, we discuss the major challenges facing blockchains and summarize ongoing research and commercial offerings that seek to address these challenges.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Peer-to-peer architectures**; • **Security and privacy** → *Cryptography*; • **Information systems** → Data management systems;

Additional Key Words and Phrases: Blockchain, distributed ledger, smart contracts, cryptocurrency

ACM Reference format:
John Kolb, Moustafa AbdelBaky, Randy H. Katz, and David E. Culler. 2020. Core Concepts, Challenges, and Future Directions in Blockchain: A Centralized Tutorial. *ACM Comput. Surv.* 53, 1, Article 9 (February 2020), 39 pages.
https://doi.org/10.1145/3366370

1 INTRODUCTION

Blockchains have recently emerged as a dynamic area of research, investment, and product development. They are covered extensively in technology-focused [52, 88] and mainstream media [51, 105, 106], serve as the foundation for products and services that are valued at billions of dollars, and are the subject of numerous research publications put forth by both academia and industry. Given the intense hype around this technology [38, 82, 102, 112], it can be difficult to understand the true nature of blockchain systems, weigh their technical merits against their inherent limitations, and distinguish between legitimate claims versus optimistic marketing. Abstractly, a blockchain is a mechanism to reach agreement on the order of events among a set of entities. This problem traces its origins to classical ideas from distributed computing and cryptography. Although blockchains are designed with a slightly different set of goals in mind, they are best viewed as a continuation of prior research in these areas.

Authors' addresses: J. Kolb, M. AbdelBaky, R. H. Katz, and D. E. Culler, University of California – Berkeley, Computer Science Division, Berkeley, CA, 94720-1776; emails: {jkolb, moustafa, randy, culler}@cs.berkeley.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
0360-0300/2020/02-ART9 $15.00
https://doi.org/10.1145/3366370

Blockchain's distinguishing feature, and the source of the enthusiasm around it, is that it enables a fully *decentralized* system. No single entity, such as a user or institution, runs the blockchain or controls its operation. Instead, blockchains leverage cryptography and carefully designed incentive schemes to allow the creation of an immutable and reliable distributed ledger that serves as the final authority on the order of events. The only assumption is that a majority of the blockchain's participants behave honestly. As a result, two mutually distrusting entities can cooperate with one another by using the blockchain as a data repository and point of coordination rather than involving a trusted third party.

This article is intended to be a self-contained tutorial on blockchains that is accessible to a general computer science audience. We assume the reader has basic background knowledge of computer networks, simple applications of public key cryptography, and traditional database systems. We assume no prior experience with blockchains, cryptocurrencies, or general distributed computing. This tutorial provides an introduction to the inner workings of a blockchain, as well as their strengths and limitations, and then goes into more detail on Ethereum as a representative example and basis for comparison against other blockchains discussed later on in the article. We present the challenges currently facing blockchains, which also involves a discussion of system design choices, and summarize the ongoing research efforts to address these challenges. Finally, we provide an overview of several commercial blockchains. Given the large body of work and rapid pace of development in this area, it is impossible to present a fully comprehensive survey on all aspects of blockchain technology. We instead focus on the most prominent and illustrative concepts and examples, while providing ample references for readers to further pursue subtopics of interest.

Due to the intense interest in blockchain, a number of academic surveys and books covering the area have been written. Many of these focus on a specific platform such as Bitcoin [4, 15, 46, 69, 123] or Ethereum [5]. Other works focus on a particular application of blockchain, such as cryptocurrencies [124] and smart contracts [9], or on a specific aspect of their operation such as consensus [8, 23], smart contract security vulnerabilities [6], anonymity and privacy [31, 77], and networking [94]. Some surveys examine a wider array of blockchain topics but, unlike our tutorial, do not discuss the inner workings of blockchains in detail [36]. This can make it difficult to fully understand the tradeoffs in blockchain design and evaluate proposed alternatives. Additionally, while there are informal articles as well as longer books [70, 91] that attempt to explain blockchain, there is a shortage of academic tutorials that discuss blockchain and the related concepts from first principles and in the context of the many historical developments that make blockchains possible.

This article is divided roughly into two parts. The first part serves as an introduction and tutorial on blockchain technology. It begins by introducing the fundamental concepts of blockchain in Section 2, using Bitcoin as a running example. Section 3 goes into more detail on Ethereum, particularly its smart contracts. Section 4 then offers a comparison between blockchains and more traditional distributed systems. The second part of the article provides an overview of the research and development efforts that seek to address four key blockchain challenges: inefficient consensus mechanisms (Section 5), limited performance/scalability (Section 6), concerns about privacy (Section 7), and smart contract security (Section 8). We discuss some of today's most prominent commercial blockchain systems, examining their designs and the tradeoffs that result, in Section 9. Section 10 summarizes and concludes the tutorial.

2 BLOCKCHAIN OVERVIEW

In this section, we present an overview of blockchain technology, focusing on the basic design elements that have been adopted by typical blockchain systems. We explore these ideas using the Bitcoin blockchain as a running example. Bitcoin introduced the modern concept of a blockchain

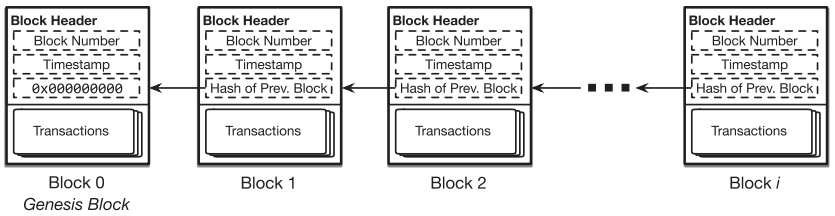

Fig. 1. A simple blockchain.

with a white paper, written under the pseudonym Satoshi Nakamoto, describing the system's design and operation [90]. Nakamoto envisioned a distributed software system for exchanging digital tokens, Bitcoins, that achieves many of the desirable guarantees offered by the traditional financial system—robust tracking of account balances, immutable and auditable transaction history, a global network for payment processing, and so on—without the centralization of control and trust in financial institutions that is inherent to modern banking. Instead, Bitcoin relies on a peer-to-peer network to collectively validate, process, and ensure the integrity of all transactions. By design, this network has no central authority with the ability to unilaterally approve invalid transactions, roll back previous transactions, or otherwise manipulate the state of the system through any means aside from normal submission of transactions for processing.

2.1 Blockchain as a Ledger

The fundamental capability offered by a blockchain is the maintenance of a *ledger*—an append-only log for storing data. A new item can only be added at the end of the ledger, and no previous items can be modified. Furthermore, a new item is not immediately added to the ledger. Instead, pending items are periodically batched together into a *block*. In Bitcoin, for example, a new block of transactions is appended to the ledger in roughly 10-minute intervals. Thus, the ledger is formed from a sequence of blocks, each containing an ordered list of ledger entries.

Bitcoin uses this ledger to track the ownership of virtual tokens, called Bitcoins. These tokens are distributed among accounts, each uniquely identified by a public key. Just as a bank might associate an account number with a balance of traditional currency, Bitcoin associates a public key with a balance in Bitcoins.[1] Each item added to Bitcoin's blockchain-backed ledger serves as a record of a *transaction*—a transfer of Bitcoins from one public key to another.

Cryptography plays an important role in Bitcoin and in blockchains more generally.[2] Each transfer of Bitcoins must be signed by the sender's private key, preventing forgery. Additionally, as shown in Figure 1, each block comprising the ledger includes a header with several fields, including its position within the sequence and a timestamp. A block's header also includes a cryptographic hash[3] of the contents of its predecessor within the sequence. A cryptographic hash of block i is included in block $i + 1$, a hash of the contents of block $i + 1$ (including its hash of block i) is included in block $i + 2$, and so on, meaning the contents of each block are reflected via a chain of hashes found within all following blocks in the sequence. This is what gives the blocks a definitive order and makes their contents immutable—any modification to a preceding block would break the

[1] In reality, Bitcoin associates a public key with possession of unconsumed digital tokens, known as unspent transaction outputs (UTXOs), but the abstraction presented to users is that of an account balance.

[2] We do not discuss the basics of public key cryptography in this article. Interested readers are encouraged to read a textbook or reference such as [71] or Chapter 8 of Reference [76] for an introduction.

[3] Blockchains fundamentally rely on several important properties of cryptographic hash functions. These functions are collision resistant, they are very difficult to invert, and their output values are uniformly distributed.

hash chain. Hence, the sequence forms a *blockchain*, inspired by prior research in cryptography on linked timestamping of documents [10, 55, 56].

Bitcoin's tokens are often referred to as a *cryptocurrency*, because they can be freely exchanged among any two entities possessing public/private key pairs, which are easily created and freely available, and because the integrity and immutability of transactions is achieved through cryptography. In the case of Bitcoin, the blockchain's ledger is used to record and safeguard all token exchanges, but entries in the ledger can be used to store general information. A blockchain can be viewed as a general class of distributed system, with cryptocurrency as just one of its applications.

2.2 Participating in and Building the Blockchain

A blockchain is maintained by a peer-to-peer network. However, one does not need to operate a member of the network to interact with its associated blockchain. Instead, when a user adds a new entry to a blockchain's ledger, she submits a transaction to an existing member using a Remote Procedure Call (RPC) protocol. The member broadcasts the transaction to the rest of the network for inclusion in a future block. Similarly, a user may submit a query to a network member about the contents of the blockchain's ledger. This means that the parties involved in a transaction, such as the sender and receiver of Bitcoins in a transaction on the Bitcoin blockchain, are not directly involved in the execution of that transaction. Instead, this task falls to the members of the network.

Some blockchain users may wish to participate directly in a blockchain's peer-to-peer network by running one or more members of the network. In *public blockchains* like Bitcoin, anyone may freely add new members to this network and later remove them at will. Each member stores its own full copy of the blockchain, containing every block and thus every ledger item, and keeps this copy synchronized with the latest updates to the blockchain by continuously monitoring the network for notification of new blocks. A user running a network member has to commit computation and storage resources to this purpose, but by retaining a copy of the blockchain she does not have to trust an intermediary to query the blockchain's state or submit transactions on her behalf.

A subset of the members within a blockchain's peer-to-peer network not only maintain copies of the blockchain, but also actively construct and propose new blocks to be added to the chain. This process is known as *mining*, and these members are therefore referred to as *miners*. Miners have to follow a certain protocol to ensure the property of *consensus*—all members of the blockchain's network reach agreement about each new block to add to the chain and also have an identical view of all previous blocks. This means that all blockchain copies are identical across the network. While there is an additional computational cost to assembling blocks and participating in a consensus protocol, users may choose to run miners, because they have a vested interest in the successful operation of the blockchain or because of more explicit incentives (discussed below).

In Bitcoin and several other blockchains, miners follow a *proof of work* algorithm (Figure 2) to determine which miner appends the next block in the chain. The main rationale for using this algorithm is to prevent miners from immediately appending any newly prepared batch of transactions as a new block on the chain. If this were permitted, then many miners could continuously and simultaneously grow the chain, making it difficult to determine a globally recognized ordering of blocks, which is required to form a unified view of the blockchain's state. Instead, each newly appended block must also contain a random quantity, a *nonce*, such that a cryptographic hash of the block's contents, including the nonce, falls below an upper threshold in its representation. Because a sound cryptographic hash function cannot be inverted, the only means of discovering a nonce satisfying this constraint is through brute-force search. This search process is the *work* while the satisfying nonce is the *proof* of this work. The first miner to find a proof appends the next block to the chain.

(1) Accumulate a batch of pending transactions that have been received from peers on the network but have not yet been included in any previous block and package these transactions as a payload p.
(2) Search for a nonce n that, when concatenated with p, produces a cryptographic hash that does not exceed a specified threshold. That is, $H(p \cdot n) \leq t$ for some bit string t.
(3) If some other valid block is received before n is found, append that block to the chain and return to Step 1.
(4) When the proof of work n is found, broadcast the new block, including n, to the network. Return to Step 1.

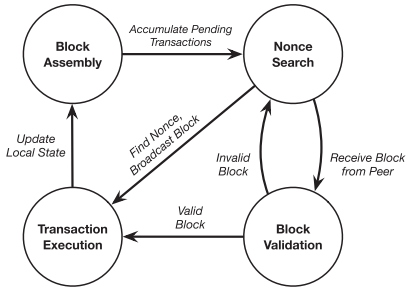

Fig. 2. The mining process in proof-of-work consensus.

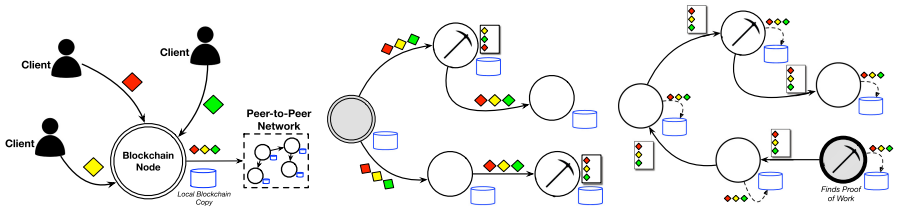

Fig. 3. Execution of transactions in a typical blockchain network.

Figure 3 summarizes the interaction among users and miners in a typical blockchain. In Figure 3(a), clients submit their transactions to a node within the blockchain's peer-to-peer network. This node forwards the transactions to the rest of the network. A subset of these nodes act as miners and incorporate the new transactions into blocks they are assembling as part of the proof-of-work consensus protocol, as shown in Figure 3(b). After one of the miners finds the proper nonce, its proof of work, it broadcasts the new block to the network. All other nodes learn of the new block, validate and execute its transactions, and commit the results to their local chain replicas, as in Figure 3(c).

In effect, proof-of-work consensus is a repeated lottery that determines which miner is allowed to dictate the next block in the chain. All other miners validate and accept the new block before moving on to the next round of the protocol and a new lottery. A particular miner's odds of winning a lottery round (and thus its degree of influence over the operation of the blockchain) are proportional to the rate at which it can test nonce values in search of a valid proof of work. Therefore, a miner's influence is tied to its computing power. To seize control of a blockchain's operation and carry out attacks such as rewriting prior blocks or adding invalid entries to the ledger, a malicious actor would need to control a majority of the computing power devoted to mining in the underlying peer-to-peer network. The assumption that this is extremely unlikely in large-scale public blockchain networks like Bitcoin's is fundamental to their security.

Tying influence to computing power also gives proof-of-work consensus resilience to *Sybil attacks*—a technique in which a single adversary masquerades as many synthetic users of a system to gain control of that system. In a simple consensus algorithm based on voting, for example, an attacker can submit votes on behalf of many artificial identities, engineering a majority and determining the outcome of the consensus protocol. Resistance to Sybil attacks is particularly important in public blockchains, where new user identities, represented as public keys, are freely available by design. If a single principal joins a proof-of-work blockchain by mining under many synthetic

identities, then she now has to decide how to apportion the computing resources under her control among these miners. Dividing these computing resources and acting as many miners is no more influential than acting as a single miner, as either configuration gives the principal the same odds of finding a valid proof of work and declaring the next block in the chain.

The quantity t in Figure 2 is a bit string representing an upper bound on the output of the cryptographic hash function used to produce a proof of work. This threshold is controlled by an adaptive and time-varying parameter known as the *difficulty* of the mining process. The precise determination of t from the current difficulty value varies across blockchains. A smaller t value reduces the number of nonces that can serve as a valid proof of work, while a larger t value increases the number of such nonces. Therefore, mining difficulty determines the expected number of nonces that must be tested before any of the network's miners succeeds in finding a proof of work. It is adjusted to keep the expected time delay between two successfully mined blocks constant even as the collective computing power of a blockchain's peer-to-peer network (and thus the rate at which nonce values can be tested) fluctuates over time as nodes join and leave the network.

2.3 Blockchain Incentive Structures

Miners in a public blockchain network perform the task of including a new transaction in a block, causing it to be processed by all nodes maintaining a copy of the blockchain, on behalf of the transaction's creator. There is no assumption that a miner has a direct interest in the processing of any particular transaction. Instead, miners are incentivized to process transactions as a service. A user augments their transaction with a *transaction fee*—a sum of the blockchain's native digital tokens, e.g., Bitcoins—that is paid to whichever miner ultimately appends a block containing that transaction to the chain. In other words, when a miner finds the necessary proof of work and creates a new block, they are compensated with the transaction fees associated with the block's transactions. In typical public blockchains, there is no standard amount for transaction fees. Instead, it is a market-determined value that changes with network-wide demand for transaction processing.

Most blockchains also allow miners to claim a fixed reward for each new block they append to the chain. This serves as a second incentive for miners to process transactions. Like transaction fees, a block reward is in the form of the blockchain's native token. These tokens are newly "minted" to expand the global token supply and are initially bound to the miner's public key for future use.

While cryptocurrency tokens have been the subject of intense financial trading and speculation, their innate value comes from one source: They are the only means of exchange to pay the transaction fees that go to miners. Tokens grant their bearer the ability to add transactions to the relevant blockchain, e.g., Bitcoins enable transactions on the Bitcoin blockchain. However, cryptocurrencies and blockchains are not the same thing. Digital tokens are used as a mechanism in most public blockchains to incentivize miners to process transactions, and in Bitcoin the entries in the blockchain-backed ledger track the exchange of these tokens. In other blockchain systems, like Ethereum, tokens remain as a source of incentive, but the ledger is more flexible and can record and protect token exchanges along with other general-purpose transactions. Still other blockchain systems discard the notion of tokens entirely, relying on alternative protocols to enforce consensus that do not rely on explicit miner incentivization.

2.4 Forks

Proof-of-work consensus is essentially leader election by lottery. The miner of each new block is chosen non-deterministically, which can lead to a split view of the blockchain's state among its participants, known as a *fork* in the chain and depicted in Figure 4. Imagine that two nodes *a*

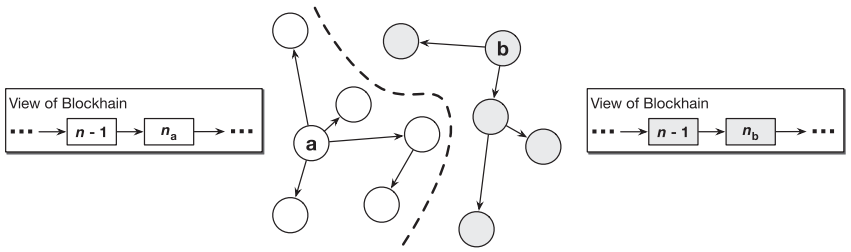

Fig. 4. A fork in the blockchain caused by two miners, a and b, each proposing a different block to add to the head of the chain. All nodes in the network agree on the state of chain through block $n-1$, but one subset of nodes has adopted n_a as the next block, while the other subset has adopted n_b.

and b each find valid proofs of work nearly simultaneously. As the new blocks they have mined propagate through the blockchain's peer-to-peer network, one group of nodes will append a's block to their local copy while another group of nodes will do the same for b's block. The network is more vulnerable during a fork, because the hashing power of the network is now split into two competing groups, which introduces security vulnerabilities [48].

This is resolved by adding a simple rule to the proof-of-work consensus protocol: When a node observes a fork in the blockchain, it should always follow the longer of the two chains, i.e., the chain that contains more blocks.[4] Every node should treat this longer fork as the canonical state of the chain, and every miner should add new blocks onto the head of this chain. One of the two forks will have its first block adopted by a subset of the network with more hashing power. This subset of the network will then be able to mine blocks at a faster rate than the nodes backing the second fork, so it will become longer with time, and this disparity will only grow as more nodes identify the longer fork and abandon the shorter fork.

The combination of the blockchain's structure, where each block contains a hash of its predecessor, and the possibility of forks has led to the notion of *confirmations*. When a transaction is included in what appears to be the newest block on the chain, it is not yet certain that this block will become part of the canonical chain, and therefore that the transaction will take effect throughout the network. The block in question could turn out to be part of an eventually abandoned fork. Moreover, the more successors a block has in the chain, the more resistant those block's transactions are to attack. This is because an attacker must have the hashing power necessary to force the network to roll back all of a block's successors before it can alter the contents of the block itself. Therefore, many blockchain applications will not consider a transaction immutable until a sufficient number of *confirmation* blocks have been appended as successors on the chain to the transaction's block. Each successor reduces the probability that the transaction is reverted by an attacker or discarded as part of an abandoned fork. Section 5 covers consensus mechanisms that attempt to minimize forks.

2.5 Scalability Issues

One of the fundamental limitations of a typical blockchain implementation—where proof of work is used for consensus and every transaction is validated and executed on every node—is its inability to handle high transaction load. In fact, public blockchains enforce upper limits on two factors that determine the rate at which they can process transactions—the rate at which new blocks are created and the size of each block—to maintain security. However, as we will discuss in Section 6,

[4]In some blockchains, such as Ethereum, the rule for choosing between two forks is actually a more complicated version of this principle. Interested readers can refer to the introduction of the GHOST protocol [117] for a more complete discussion.

the challenge of improving blockchain scalability has motivated significant research efforts, and we will see alternative system designs that seek to address this issue.

Block Creation Rate. Recall that mining difficulty in a proof-of-work blockchain is used to maintain a relatively steady time delay between the creation of subsequent blocks, even as the collective computational power of the blockchain's network varies over time. A constant delay between blocks is desirable, because it keeps the resource demands imposed by participating in a blockchain stable. All nodes in the blockchain's network, whether or not they are miners, must devote computational resources to the task of validating any new block upon arrival as well as executing the block's constituent transactions to update their respective local copies of the blockchain's state. As the block period shortens (meaning the block creation rate increases), more computational load is placed on the blockchain's nodes. If the period becomes too short, then only nodes backed by especially capable hardware will be able to keep up, which defeats the purpose of making public blockchains decentralized and open to anyone. Additionally, if the inter-block delay becomes shorter than the time required for knowledge of a new block to fully propagate throughout the blockchain's peer-to-peer network, some nodes will perpetually remain behind the head of the blockchain. Thus, the block period presents a tradeoff between scalability and centralization.

Furthermore, a shorter time period between blocks introduces security concerns, mainly because network propagation time is now a more significant fraction of the inter-block period. After a miner proposes a new valid block b, many other miners will still be working to find a new block on top of b's parent, or perhaps working as part of a fork, before they learn of b's existence. During this time, a potentially significant portion of the network's collective hashing power is not devoted to mining on top of—and thus protecting—the main chain, increasing the system's vulnerability to attack [21, 48]. Such a time interval exists anytime a new block is mined, but this window of vulnerability occupies a greater portion of the block period when this period becomes shorter. Block period therefore presents a second tradeoff between scalability and security.

Block Size. Similarly, the size of each block in the chain is capped to limit the data storage requirements placed on the blockchain's nodes. While the blockchain constantly grows over time, as does its representation stored locally on each node, limiting the block creation rate and block size limits the rate of this growth. Much like limiting the computational burden of participating in a blockchain, it is important to limit the analogous storage burden to avoid centralizing the operation of the blockchain to a relatively small group of more powerful nodes.

Transaction Throughput. The rate at which new transactions are added to a blockchain is equal to the product of the block creation rate and the number of transactions included per block. As explained above, however, both of these factors are intentionally limited to maintain security and to prevent excessive centralization. Hence, a typical public blockchain's throughput is fundamentally limited by its design. This situation arises because proof-of-work consensus is non-deterministic, forcing a blockchain's design to minimize but still accommodate the possibility of forks, and because the blockchain's guarantees of consistency, availability, and resilience are achieved by processing every transaction at every node within the network.

Transaction Latency. A client must wait a significant amount of time—on the order of minutes when a strong guarantee of immutability is required—before they know if their transaction has executed successfully. Not only must a client wait for their transaction to be batched into a newly mined block b, they must also wait for several confirmation blocks to be mined on top of b (as explained above in Section 2.4) to effectively eliminate the probability of an attacker rolling back the transaction or of block b ending up in a fork of the main chain. For example, the Bitcoin community recommends that users wait for six confirmation blocks, requiring 60 minutes of expected wait time, before assuming that a transaction is immutable.

This latency could be reduced by decreasing the time delay between subsequent blocks, but this raises the centralization and security concerns described above. Furthermore, if the inter-block delay is naively reduced by decreasing mining difficulty, then confirmation latency will become no shorter. Mining each individual block involves less computation than before, and the computation to revert a sequence of blocks similarly decreases. As a result, a blockchain user has to wait for more blocks to be appended to the blockchain to achieve the same level of confidence in the immutability of their transaction. In other words, more blocks are added during the confirmation period, but they require the same amount of total computation to be mined. The transaction in question is protected, because reverting these subsequent blocks would require a similar amount of computation.

2.6 Privacy Issues

A public blockchain's design also limits user privacy. Any transaction newly added to a blockchain's ledger is propagated for validation to all members of the network. This ensures a globally consistent view of the ledger's state and allows the full computational power of the blockchain's network to be used to secure the ledger. In addition, all transactions, along with their inputs and outputs, are propagated in unencrypted form. Transactions are validated by replaying them in the context of the latest version of the blockchain's ledger, and it is difficult to accurately replay a transaction without full access to the necessary data. For example, when two people make an exchange of Bitcoins, everyone on the network is able to see their public keys and the amount of tokens transferred.

Although a public key does not explicitly identify its owner, it is possible to deanonymize blockchain transactions and attribute keys to specific entities by constructing and analyzing graphs of transactions, keys, and their relationships [27, 108, 111]. In Section 7, we will see several approaches to improving blockchain privacy, without sacrificing desirable properties such as availability and consistency, that leverage tools such as zero-knowledge proofs and trusted execution environments (TEEs). However, there is currently no public blockchain system that enables fully private and general-purpose transactions without some expectation of trust from the user, such as trust in a hardware vendor's TEE implementation.

2.7 Permissioned Blockchains

A *permissioned blockchain*, also known as a *consortium blockchain* or *federated blockchain*, is set up to restrict who may join the underlying peer-to-peer network, either as a miner or as an observer. This is often motivated by the scalability and privacy concerns discussed above. A *private blockchain* is a specific type of permissioned blockchain in which just one entity has the ability to append new blocks to the chain, although multiple entities may be able to read the blockchain's contents. Because it still involves a chain of hashes, a private blockchain still protects the integrity of its data, but this arrangement arguably defeats the primary purpose of running a blockchain—to enable cooperation among entities who trust the cryptography underlying the blockchain itself rather than each other. In a private blockchain, control is centralized in its owner, who may unilaterally manipulate the blockchain's contents.

In a permissioned blockchain, a new node must present a proof of its right to join the blockchain's network and to begin receiving the necessary information to maintain a local replica of the blockchain. Typically, this proof is tied to the possession of a cryptographic key shared out of band or the possession of a private key corresponding to a whitelisted public key. The whitelist of public keys may be created out of band or maintained on the blockchain's ledger itself—entries in the ledger record modifications to the whitelist. If the blockchain wishes to enforce a distinction between the permission to join the network and the permission to mine blocks, then a miner must also present an appropriate proof whenever it proposes a new block to append to the chain.

Permissioned blockchains often set aside proof-of-work consensus because of its nondeterminism and the computational burden it imposes on miners. They are able to do this because they rely on a weaker set of assumptions than public blockchains. Membership in a permissioned blockchain's network is composed of a fixed set of principals, and new identities are not freely available as in a public blockchain, meaning Sybil attacks are no longer feasible. Permissioned blockchains can thus adopt weaker but more performant consensus mechanisms based on traditional protocols from distributed computing, such as Paxos [80], Raft [97], and Byzantine fault tolerant algorithms [25].

Permissioned blockchains have attracted considerable interest from the business community. In a typical use case, several companies who do business together form a permissioned blockchain to share, update, and agree upon transaction records without any requirement of mutual trust. As it is fundamentally distributed, the operation of the blockchain does not become dependent on the infrastructure of any single participant, and no company can unilaterally manipulate or repudiate transaction records. Furthermore, because the chain is permissioned, only authenticated principals acting on behalf of the companies involved, who have a vested financial interest in the operation of the system, are allowed to read or modify the blockchain's state.

2.8 Beyond Cryptocurrency

Although the modern concept of a blockchain first emerged in the context of Bitcoin and cryptocurrency, securing the exchange of digital tokens is just one application of a blockchain's ledger. The ledger can also be used more generally to record the creation of a body of data and subsequent transformations made to this data. Instead of simple token transfers as in Bitcoin, ledger entries correspond to transactions containing application-specific and fully programmable logic. Here, digital tokens become a means to a larger end, incentivizing miners to process transactions and maintain the ledger. These general-purpose blockchain platforms can support a rich array of applications by serving as a distributed, reliable, and immutable data repository. Because the blockchain is decentralized, no single entity can tamper with application data, whether they are an attacker, a user, or even the application's creator. Instead, the only way to effect a transformation is to add a new transaction for validation and addition to the blockchain's ledger.

Imagine that we wish to use a blockchain to record and secure the proceedings of a simple auction. Unlike a traditional auction, payment is made in the form of the blockchain's associated token. Proper payment by the buyer is enforced by the blockchain. If the asset for auction is in electronic form, then its transfer may also potentially be enforced by the blockchain, but the transfer of physical assets is handled out of band.

To begin, a seller initiates the auction by adding a new item to the ledger. This ledger entry specifies the rules of the auction, such as a minimum bid amount and the deadline for submitting bids. A potential buyer submits a bid by adding a ledger item containing her public key, the amount of her bid, and a timestamp. The tokens backing this bid are held in escrow when the ledger is updated. If the bid is later surpassed, then these tokens are returned to the bidder. When the auction's deadline is reached, the current highest bidder is declared the winner, and her tokens are released and credited to the seller.

Running an auction in this fashion has a number of benefits. The rules of the auction are declared at the outset on the ledger and cannot be manipulated afterwards. For example, the seller cannot extend the deadline in an effort to solicit higher bids, nor can bids be ignored to exclude certain parties from participating in the auction. Because each bid corresponds to an entry in the ledger, the full proceedings of the auction are preserved on the blockchain and therefore open for inspection.

Blockchains that serve only to back a cryptocurrency, like Bitcoin, are unable to support this use case, because the entries in the underlying ledger are restricted to simple token exchanges.[5] Instead, an auction requires a blockchain that supports application-specific transactions. The fundamental primitive in such blockchains is the *smart contract*—a body of data along with executable code that expresses the logic for transactions that transform this data. The only permissible means of modifying a smart contract's data is through the execution of one of its declared transactions. While the notion of a smart contract as a combination of code and cryptographic primitives to protect sensitive data is not new [121], they have become a topic of renewed interest in light of their deployment on blockchains.

3 ETHEREUM AND SMART CONTRACTS

Ethereum, introduced by Vitalik Buterin [20] and formally specified by Gavin Wood [126] in 2014, was the first blockchain to support smart contracts. It remains the best known, most widely used smart contract platform today. Ethereum's blockchain is public and operates very similarly to Bitcoin's. Identities are represented as public keys and authenticate their transactions with the corresponding private key. Each public key is associated with a balance of digital tokens, Ether, that are used to compensate miners for processing transactions in a proof-of-work consensus protocol. The key difference, however, is that Ethereum transactions can be customized to effect more specific transformations to the blockchain-backed ledger, such as recording a bid for an auction.

The auction described above can be implemented with a smart contract containing four fields:

- The public key of the seller;
- The amount of the highest bid seen so far;
- The public key of the sender of the highest bid;
- A timestamp for the auction's end deadline.

The contract also contains code specifying the procedure for processing a new bid, first checking that the auction's deadline has not passed and then comparing the new bid against the highest known bid. If the new bid exceeds the highest preceding bid, then the contract's internal data are updated to record its amount and sender. The transaction sender's tokens are transferred to the custody of the smart contract. Any tokens staked by the previous highest bidder are returned to them. A second contract transaction allows the seller to redeem the tokens making up the winning bid, but only after checking that the auction's deadline has passed.

3.1 Ethereum Internals

In Bitcoin, a public key represents an end-user account, which holds funds for a particular principal. In Ethereum, however, public keys represent both end-user accounts and smart contracts, which are not inherently bound to a specific entity but instead encapsulate data and operations to offer a particular abstraction. In fact, end-user accounts and smart contracts are treated identically during the execution of transactions. Each Ethereum transaction corresponds to a ledger entry, propagated throughout the network, that includes the following elements:

- A *destination*, specified as a public key;
- A *value* of Ether;
- A *payload*, as an arbitrary byte string;
- A *signature* generated by the private key of the sender.

[5] Bitcoin does offer a scripting interface, but it maintains a focus on tokens and cannot express general-purpose transactions.

A transaction's associated value is credited to the destination public key, increasing its balance. When the destination specifies the address of a smart contract, a miner or member of the blockchain's network executes the transaction by deciphering the message's payload, which must specify which of the contract's transformations to invoke and the parameters for this transformation. A contract may contain a *fallback function* that is invoked when a transaction's payload fails to match any of the contract's explicitly declared transformations. Conversely, when the destination public key identifies an end-user account, the transaction's value is absorbed, but no computation occurs.

Smart contracts and end-user accounts are thus two sides of the same coin. We can view an end-user account as a degenerate smart contract whose only function is a no-op fallback function. Similarly, we can view a simple transfer of Ether, analogous to a transaction on the Bitcoin blockchain, as a degenerate transaction that contains no payload, meaning it calls for no computation to take place. However, a smart contract is like an end-user account endowed with the ability to autonomously execute code as well as to send and receive funds. The invocation of a contract transformation is a special kind of Ether transfer augmented with payload data that specifies additional processing to take place.

3.1.1 Network Operation. Ethereum has a standardized peer-to-peer protocol that allows the members of its network to exchange messages to share knowledge of new smart contract transactions, broadcast and receive new blocks to append to the blockchain, or look up prior blocks. Each member, also known as a *node*, maintains a local copy of the Ethereum blockchain. At the owner's discretion, a member may also participate in Ethereum's proof-of-work consensus protocol by assembling new blocks and searching for a nonce as described in Section 2.2. While any computer running software that adheres to Ethereum's protocol may join the network, in practice the majority of Ethereum's community uses publicly available software to operate its nodes. The most popular and best supported options are Geth,[6] implemented in Go, and Parity,[7] implemented in Rust, although implementations in other languages, such as C++,[8] are also available.

Each member's local copy of the Ethereum blockchain is composed of two conceptual pieces. First, there is the sequence of blocks that were mined over time and make up the blockchain itself. Second, there is the current global *state* of the Ethereum blockchain. This consists of the balance of tokens assigned to each public key as well as the current values for the fields of every smart contract. Ethereum's distributed ledger is formed from the concatenation of the transaction sequences contained in all blocks mined thus far, and the blockchain's current state is the cumulative result of the sequential execution of the ledger's transactions. Ethereum represents its state in a key/value store backed by a trie-based data structure [41] that enables efficient queries, maintains a history of old versions, and uses cryptographic hashing to protect its integrity. Because maintaining a local copy of the blockchain can impose high storage costs, particularly as the blockchain is continuously growing and its state is continuously changing, members of Ethereum's network typically prune defunct state data.[9]

The logic of smart contract transformations is expressed in bytecode for a stack-based virtual machine specifically designed for Ethereum, the Ethereum Virtual Machine (EVM). EVM bytecode is Turing-complete and defines the low-level operations that are combined to specify a contract transaction. These include reading or writing the contract's fields, arithmetic operations and

[6] https://ethereum.github.io/go-ethereum.
[7] https://parity.io.
[8] https://github.com/ethereum/cpp-ethereum.
[9] In addition, there is an alternative mode of operation in which a member of the network acts as a *light client*, storing only block headers and downloading block contents and global state on demand to run transactions.

hashing, and jumps or conditional branches. The implementation of an Ethereum node includes a bytecode interpreter for the local execution of transactions. When a new smart contract transaction is added to the blockchain, it is executed on two occasions. First, any miner who assembles a block that includes the transaction will validate and execute the transaction's bytecode (whether or not the block is ultimately appended to the blockchain). Second, when a new block is mined and broadcast to the network, every node will execute the block's transactions to update their local copies of the blockchain's state. Therefore, a smart contract's code is not executed at one particular time and place; rather, it is executed many times and on every node. This strategy is computationally inefficient, but it is precisely what makes the blockchain decentralized—no single authority can stipulate the results of a transaction; instead, every member of the blockchain verifies the outcome independently to reach global consensus.

Any network member that stays synchronized with the Ethereum blockchain, whether or not it also acts as a miner, serves as an entrypoint for external software to interact with the blockchain. Each node in the blockchain network exposes a JSON-RPC interface that allows client software to issue requests. Bindings and libraries that abstract the necessary RPC operations are available in many languages, although JavaScript[10] has emerged as the best supported and most popular choice. If a client's request is to retrieve information that is stored on the blockchain, such as the value of a field within a specific smart contract, then a node can consult its local copy of the blockchain and respond immediately. If the client's request is to modify the state of the blockchain—say, by changing the value of a contract field or instantiating a new contract—then its entrypoint node must propagate the transaction to miners through Ethereum's peer-to-peer network so it is included in a future block in the chain. From the client's perspective, reading from the blockchain is synchronous, while writing to the blockchain is asynchronous and exhibits much higher latency.

3.1.2 Limiting Contract Computation with Gas. As explained above, smart contract logic is expressed as EVM bytecode that is executed whenever a transaction is added to Ethereum's ledger. Because this execution takes place at every single node in the blockchain's network, computation and storage become precious resources. Inefficient contract code will needlessly consume resources at a large scale. Worse still, a smart contract can contain non-terminating computations like an infinite loop. While this is a routine bug in the course of normal programming, it poses a serious problem in the context of a blockchain. Any miner that executes contract code containing an infinite loop will, in theory, remain stuck processing a single transaction indefinitely. A malicious or buggy contract could thus prevent any forward progress in the construction of the blockchain.

Ethereum has a mechanism built in to the execution of EVM bytecode to encourage efficient contract implementations and to avoid non-terminating computation. Each virtual machine instruction, such as computing a value or mutating a contract field, is associated with a cost, known as *gas*. More intensive operations incur higher gas costs, and the cost to run a transaction is simply the sum of the gas cost of all virtual machine operations performed in the course of its execution. Each transaction is associated with a finite amount of gas, and, if this supply runs out, the transaction is terminated and its changes to contract state are reverted.

The amount of gas available for transaction execution is determined by the transaction's sender, who specifies the price p they are willing to pay, in Ether, for each unit of gas consumed by the transaction. The sender must also include an amount of Ether E with her transaction (treated separately from any Ether being transferred to the transaction's target) to cover this gas cost. This effectively limits the transaction to E/p gas units' worth of execution. Any residual Ether

[10] https://github.com/ethereum/web3.js.

```
1  contract Auction {
2    address public highestBidder;
3    uint256 public highestBid;
4    ...
5    function bid() public payable { // Contains security vulnerabilities
6      require(msg.value > highestBid);
7      if (highestBid > 0) {
8        highestBidder.transfer(highestBid);
9      }
10     highestBid = msg.value;
11     highestBidder = msg.sender;
12   }
13   ...
14 }
```

Fig. 5. Part of a Naive auction contract.

is returned to the sender. In the event that all gas is exhausted before a transaction finishes, the sender is charged in full, but all computation is reverted. This discourages attacks in which the blockchain's members are forced to devote computational resources to the repeated execution of non-terminating transactions. Although the transactions would ultimately have no effect on the blockchain's state, the attacker must continuously supply gas fees to have them executed.

When a miner successfully adds a new block to the head of the chain, they are credited with all Ether expended to cover the gas consumption of the block's constituent transactions. Hence, gas functions both as protection against needlessly inefficient or non-terminating contract code and as a transaction fee that incentivizes mining. Moreover, the typical gas price p for new transactions fluctuates over time, dictated by the market forces of current demand on the blockchain and the availability of miners. A rational miner will always prefer to include transactions with high gas prices in their new blocks to maximize their earnings, but transaction senders want to minimize transaction fees. Gas price therefore represents a tradeoff—a sender sets the gas price of her transaction to some market-determined value, and raising this price generally reduces the delay until the transaction is mined and added to the blockchain.

3.2 Smart Contract Implementation

While one could write a contract directly in EVM bytecode, most Ethereum contracts are implemented in higher-level, contract-specific programming languages. Many of these languages represent contracts as an abstraction resembling a class in traditional object-oriented programming languages. Instance variables correspond to the contract's data, while methods correspond to the contract's transformations. A transaction against a contract instance on the Ethereum blockchain is thus analogous to a method invocation targeting a specific instance of a class in OOP.

Although numerous languages that compile to EVM bytecode have been proposed and implemented [39, 43, 60, 115], Solidity [42] is the most mature and best supported, which has made it the de facto standard implementation language for Ethereum's contracts. Solidity is similar to most imperative, statically typed programming languages but is augmented with blockchain-specific keywords and constructs. Solidity contract code is able to access blockchain metadata such as the number and timestamp of the current block as well as the public key of the principal who sent, and therefore signed, the current transaction. The language also features exceptions, which roll back any changes made to a contract's state within the current transaction when they are thrown.

Figure 5 shows Solidity source code to implement a part of the auction contract described above. Its fields `highestBid` and `highestBidder`, respectively, keep track of the highest bid seen thus far

and its sender. The `bid` function specifies the procedure that is executed upon submission of a new bid. Line 7 demonstrates two important Solidity features. The `require` keyword throws an exception if a certain condition is not satisfied, in this case rejecting the transaction if the new bid does not exceed the highest bid seen thus far in the auction. `msg.value` expresses the balance of Ether furnished by the bidder (represented by `msg.sender`) and associated with the transaction. On line 9, Solidity's `transfer` keyword is used to deduct Ether from the contract's internal balance and credit it to the former highest bidder. While the transfer operation appears simple, it involves a potential interaction with untrusted code defined by the recipient, a security concern that we will discuss next. Finally, the code updates the contract's internal state to reflect receipt of the new bid.

3.3 Contract Security Concerns

Solidity's syntax and basic execution semantics are, by design, very similar to those of traditional imperative programming languages, but writing a correct and secure smart contract can be challenging. This is because Ethereum's contract execution model and mining process introduce subtleties to Solidity's behavior, many of which have no analogues in other programming languages and platforms. Contract developers rely on their prior experiences and intuitions regarding the execution of imperative code, and Solidity's efforts to present familiar syntax can obscure the underlying blockchain's true execution semantics. Writing a correct and secure smart contract is particularly important, because smart contracts are immutable. When a new contract is instantiated, its code is stored on the blockchain's ledger and cannot be changed. In Section 8, we will summarize some of the ongoing efforts to help developers avoid introducing bugs in their smart contracts and to defend their contracts from potential attacks.

We focus this discussion on examples related to concurrency, one of the most prominent pitfalls in Solidity programming, but interested readers may wish to consult several broader treatments of contract security [6, 85, 114]. Each invocation of a function defined in Solidity corresponds to a transaction (and thus a new entry on the ledger) in Ethereum, with the function's parent contract as its target. Transactions comprising a block on the chain are ordered and executed sequentially. Hence, contract developers often assume that the body of a Solidity function cleanly executes from start to finish (i.e., it is atomic), aside from calls to other contract functions. However, transfers of control to external code can occur in subtle ways that are hard to restrict and can have catastrophic effects on a contract's integrity. As others have pointed out [58, 114], Solidity contracts are therefore better viewed as objects with mutable state that must be safeguarded against concurrent execution.

As an example, consider Line 9 of the auction contract in Figure 5, which uses Solidity's `transfer` keyword to transfer Ether to the address (public key) designated by the contract's `highestBidder` field. Recall that a transfer of Ether from one address to another is carried out as its own transaction, enacted as a message with a destination address and associated token value but no payload. When the destination address is a user account, that account is credited with the message's value, and control returns to the auction contract. However, the situation is considerably different when the destination address of the `transfer` is a smart contract. The transaction contains no payload to specify its target function and arguments, and the contract's fallback function is automatically invoked. The auction contract has now invoked arbitrary external code it does not control.

A malicious callee contract can then derail execution of the parent transaction by inducing a failure, such as simply throwing an exception, which reverts any progress made in the parent transaction. In this way, the target of a `transfer` operation can block the sender from making any forward progress, rendering it deadlocked. For example, in the auction contract of Figure 5, a

```
1  contract Auction {
2    address public highestBidder;
3    uint256 public highestBid;
4    mapping(address => uint256) pendingWithdrawals;
5    ...
6    function bid() public payable {
7      require(msg.value > highestBid);
8      if (highestBid > 0) {
9        pendingWithdrawals[highestBidder] += highestBid;
10     }
11     highestBid = msg.value;
12     highestBidder = msg.sender;
13   }
14   function withdraw() public { // Contains security vulnerabilities
15     if (pendingWithdrawals[msg.sender] > 0) {
16       msg.sender.call.value(pendingWithdrawals[msg.sender])();
17       pendingWithdrawals[msg.sender] = 0;
18     }
19   }
20   ...
21 }
```

Fig. 6. Part of an improved, but still vulnerable, auction contract.

malicious bidder can trigger an exception on Line 9, preventing any progress past this point in the code and effectively seizing control of the auction by preventing any new bidder from replacing her as the acknowledged highest bidder, an update that would normally occur on lines 12 and 13.

The Solidity development community has established an idiom to address this in which funds are only returned through a specific withdrawal function [40]. A contract user invokes this function to retrieve their funds, e.g., if they are no longer the winner of an auction. Now, if an attacker disrupts the execution of this transaction, they are only acting against their own self-interest, as they will be unable to reclaim their own Ether. Additionally, there is a second issue with the `transfer` keyword. It delegates only a very small amount of gas to the recipient contract, preventing the recipient contract from doing useful work in its fallback function in response to the incoming deposit. Solidity offers an alternative `call` primitive to forward the current transaction's gas to the recipient. A new version of the auction contract using this approach is shown in Figure 6. Note that the `bid` function no longer involves any transfer of funds, instead only updating contract state.

The improved contract of Figure 6 is still not secure. Solidity contracts are prone to reentrancy issues, where a function in progress is called again from external code. For example, the target contract of the `call` within the `withdraw` function of Figure 6 could again invoke, and thus reenter, the `withdraw` function from within its fallback function. In this example, a malicious contract is able to steal the entire balance of the auction contract through the following steps:

(1) The attacker submits a legitimate bid to the auction contract.
(2) When the attacker's bid is supplanted by a new and higher bid, the contract records its debt to the attacker in the `pendingWithdrawals` field.
(3) The attacker sends a transaction invoking the auction contract's `withdraw` function. Because of the attacker's previous bid, the conditional check on Line 15 of Figure 6 passes.
(4) The auction contract initiates a `call` targeting the attacker's contract, invoking its malicious fallback function.

```
1  function withdraw() public {
2    uint256 amount = pendingWithdrawals[msg.sender];
3    if (amount > 0) {
4      pendingWithdrawals[msg.sender] = 0;
5      msg.sender.call.value(amount)(): // Can also use transfer here
6    }
7  }
```

Fig. 7. A safe token refund implementation.

(5) The attack contract is credited with new Ether from the `call`. It then uses its fallback function to invoke, and reenter, the auction contract's `withdraw` function.
(6) In `withdraw`, `pendingWithdrawals` does not reflect the completion of the `call` from step 4. The conditional on Line 15 passes again, and another `call` to the attacker occurs.
(7) The attacker repeats this process, receiving more Ether each time a `call` occurs, and continuing the attack until the balance of the auction contract is exhausted.

There are two ways to prevent this attack. One is to again use `transfer` to send funds. This prevents the recipient contract from executing malicious code in its fallback function, but also means the fallback function cannot do any useful work in response to the deposit. In a second approach, widely used among contract developers [40], the `withdraw` function updates the user's balance in the `pendingWithdrawals` field *before* initiating the `call`, as shown in Figure 7.

This reentrancy bug is essentially the same as the vulnerability behind the now-infamous attack on `TheDAO`, a smart contract that accumulated approximately $150 million worth of Ether. The attacker was able to seize Ether valued at the time at nearly $60 million by exploiting reentrancy [35], although this caused the market value of Ether, and thus the value of the attacker's stolen tokens, to plummet. In response, a majority of the Ethereum blockchain's participants agreed to a controversial update to rewrite transaction history to revert the attack [59].

4 DISTRIBUTED COMPUTING AND BLOCKCHAINS

Blockchains represent a novel implementation of prior theoretical work that draws from multiple research areas such as distributed computing and cryptography. Interested readers may wish to consult References [91, 92] for a more detailed discussion on the history of theoretical work behind Bitcoin specifically. The goals of this section are to present a brief background on distributed computing concepts that are relevant to blockchains and to provide a high-level comparison between traditional databases and blockchains, in the process presenting the current challenges, dimensions, and tradeoffs in distributed computing that blockchains attempt to address.

4.1 Relevant Concepts

Public blockchains attempt to solve the problem of recording events in an ordered, consistent, immutable, and trust-free manner using public, anonymous, permissionless (i.e., where anyone can participate), distributed, and decentralized infrastructure. This is achieved by creating a replicated state machine [113] that maintains a distributed ledger. Entries in the ledger reflect transactions that can express simple operations (e.g., sending virtual tokens from one entity to another), as in the case of Bitcoin, or Turing-complete logic (e.g., smart contracts) as in the case of Ethereum.

A ledger is an append-only log of transactions, a concept that has been widely used in a variety of settings such as file systems and databases. There are, however, some drawbacks to maintaining a centralized, single copy of a ledger. First, it is a single point of failure. If the ledger is corrupted or the hardware behind it fails, then it becomes inoperable. Second, the ledger becomes a bottleneck

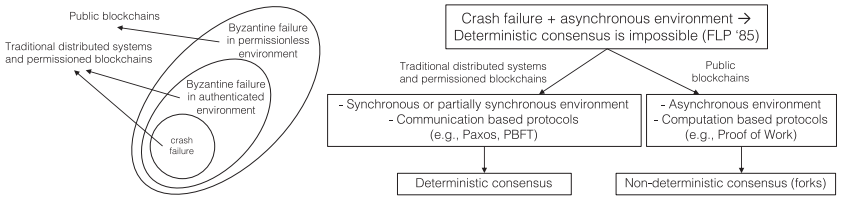

Fig. 8. Failure models and tradeoffs for reaching consensus. Public blockchains work in an asynchronous (i.e., unreliable communication among nodes) environment with a permissionless (i.e., where anyone can join) Byzantine failure model, which leads to non-deterministic consensus. Traditional distributed systems and permissioned blockchains work under a relaxed model (an authenticated environment with partially synchronous communication), which allows them to use deterministic consensus protocols.

that limits performance and scalability. Finally, if the entity maintaining the ledger is malicious, they may be able to tamper with the ledger.

A distributed ledger solves these issues by replicating the ledger among multiple entities. This allows the system to continue operation in the face of limited failures, can improve performance by spreading load across replicas, and prevents any malicious entity from unilaterally modifying the ledger's state. It also introduces a new set of challenges. For example, all participants must maintain an identical copy of the ledger, and each modification to the ledger must be agreed upon by at least a majority of participants (e.g., by voting). Honest participants may experience communication or processing failures, while dishonest participants may try to cheat the voting process.

All of these issues have been well studied in distributed computing. In a distributed system, a group of nodes works together to achieve a common goal (e.g., maintain a distributed ledger). Given that nodes in distributed systems operate concurrently (i.e., independently process transactions at the same time), there is a need for coordination among them. The task of getting all nodes to agree on the current state of the system is called distributed consensus. In particular, given an input value (e.g., a new entry in the ledger) that was proposed by a participating node, all nodes must agree upon the proposed value and append it to their local copy. A distributed system reaches consensus when it collectively and definitively determines the next modification to make to the system's state.

Usually, a distributed system reaches consensus via a consensus protocol, which specifies the procedure for the participants to follow as well as the messages they exchange. Different protocols offer different guarantees (as shown in Figure 8), such as the ability to tolerate node failures or communication failures. Distributed consensus is a fundamental problem that presents many challenges. First, given the distributed nature of these nodes, there is no single global clock to determine the ordering of events, which makes reaching consensus difficult [79]. Further, while consensus is easily achieved when all nodes are available and can communicate with one another, in a real-world setting where failures are inevitable, consensus protocols must be fault-tolerant.

Traditional distributed consensus protocols assume they are operating in an authenticated environment, where the set of participants in the protocol is explicitly known and each message can be attributed to a particular entity. Paxos [80] is arguably the best-known protocol of this type. Paxos is constructed around a replicated state machine, and its participants exchange messages to definitively determine what action to take next to modify the distributed system's state. Paxos, as well as similar protocols like Raft [97], operates under the *crash failure* model: A participant either operates honestly (i.e., follows protocol) or fails. It guarantees correct operation if a majority of the nodes work correctly. A second class of consensus protocols, e.g., PBFT [25], operates under the *Byzantine failure* [81] model, which makes a weaker set of assumptions. In this setting, nodes may

not only fail, but may also report arbitrary and incorrect results or send inconsistent messages that report different results at different times or to different entities. Protocols in this setting require honest participation from at least two-thirds of all nodes, rather than a simple majority.

Public blockchains must use even stronger consensus mechanisms, because they do not operate in an authenticated environment. Any entity may register an identity and participate in the system. In particular, a public blockchain must tolerate both Byzantine faults and Sybil attacks—a technique in which a malicious participant masquerades as many synthetic identities, allowing them to exert enough influence on a Byzantine Fault-Tolerant (BFT) consensus protocol to control its outcome, e.g., by producing an artificial majority of votes. For these reasons, both Bitcoin and Ethereum use a non-deterministic consensus protocol based on proof of work (covered in detail in Section 2.2), which is heavily influenced by previous research on limiting spam email messages [7, 37]. However, the probabilistic nature of proof-of-work protocols can lead to forks where different nodes have different views of the system (as discussed in Section 2.4). Permissioned blockchains relax the requirement of allowing any entity to join the consensus protocol and thus operate in an authenticated environment. They can use crash fault-tolerant or BFT consensus protocols such as the ones used by traditional distributed systems.

Aside from the class of failures and attacks they tolerate, there are two other important distinguishing features of consensus protocols. First, one way to view consensus protocols is as a spectrum [36] with one end representing communication-based protocols, such as those used in traditional systems, and computation-based protocols, such as blockchain's proof-of-work model. This distinction becomes less clear when we consider the alternative consensus mechanisms discussed in Section 5. Second, different protocols may make different assumptions regarding the communications among their participants. These communications may be either *synchronous*, meaning messages are always delivered within some upper time bound, or *asynchronous*, meaning messages can be indefinitely delayed, duplicated, or delivered out of order. In a famous theoretical result, reaching deterministic consensus in an asynchronous network with even one faulty node was proven to be impossible [45]. Similarly, according to the CAP Theorem [17], in an asynchronous environment one must choose between consistency and availability. Therefore, consensus protocols must relax some assumptions about the failure model, either assuming a synchronous or partially synchronous network or adopting a probabilistic approach to consensus as in proof of work.

4.2 Traditional Distributed Databases vs. Blockchains

Another way to understand blockchains is to compare them to traditional distributed databases, where in both cases a sequence of operations is applied against an existing body of state. In particular, blockchains are similar to distributed transaction management in distributed databases, in which participants maintain replicas of the state and agree on the order of transaction execution. We highlight the main differences between distributed databases, public blockchains, and permissioned blockchains in Table 1 and summarize them below. Interested readers may wish to consult Reference [36] for a more detailed discussion on the differences between blockchains and distributed databases.

First, blockchains differ from distributed databases in their inherent focus on decentralization. Blockchains, including permissioned blockchains, are typically composed of nodes running in multiple administrative domains. They must use cryptography to enable cooperation across these domains without any assumption of trust. Distributed databases, however, are primarily motivated by performance and storage considerations and are usually managed by a single entity. Public and permissioned blockchains rely on cryptographic primitives, like hashing and signatures, in fundamental aspects of their operation. Because they construct a hash chain of blocks, they are

Table 1. Comparison between Distributed Databases, Public Blockchains, and Permissioned Blockchains

Property	Distributed Database	Public Blockchain	Perm. Blockchain
Append-Only (Immutable)	No	Yes	Yes
Integrity via Cryptography	Varies	Yes	Yes
Turing-Complete Logic	Triggers	Smart Contracts	Smart Contracts
Privacy Easily Achieved	Yes	No	Yes, restricted to members
Easily Auditable	No	Yes	Yes
Requires Incentive or Penalties	No	Yes	No incentive, optional penalties for misbehaving
Requires Consensus	Yes	Yes	Yes
Consensus Protocol	Communication-based	Computation-based	Varies
Explicit Leader in Consensus	Yes	No	Yes
Permanent Consensus	Yes - Once reached	No	Yes—Once reached
Anyone Can Join	No	Yes	No
Must Tolerate Sybil Attacks	No	Yes	No
Must Trust Participant Nodes	Yes	No	Yes
Byzantine Fault Tolerance	Authenticated env.	Open env.	Authenticated env.
Unbounded Forks	No	Yes	No
Scalability and Throughput	Most	Least	Varies
Cost for Participation	No	Yes	No

append-only and immutable. A distributed database may use cryptography to protect its integrity and may expose transaction history, but neither of these features are innate to distributed databases in general. Finally, where blockchains rely on explicit incentives to reward members for processing transactions, distributed databases assume that all nodes voluntarily process transactions. Permissioned blockchains often do not require incentives, but they may use penalties if an authenticated participant behaves maliciously—another approach not seen in distributed databases.

All three systems require consensus for adding new transactions. Distributed databases and permissioned blockchains require authenticated participants and thus do not need to tolerate Sybil attacks. Therefore, they can use traditional consensus protocols such as Paxos and PBFT. Public blockchains consider a much more hostile environment and therefore use consensus mechanisms that can tolerate Sybil attacks. However, this comes at the cost of non-deterministic consensus, which can cause forks and lead to additional vulnerabilities [57]. Accounting for these vulnerabilities adds additional constraints on the performance, scalability, and throughput of public blockchains [33], making it the least scalable among all three systems (as discussed in Section 2.5).

5 ALTERNATIVE CONSENSUS PROTOCOLS

The proof-of-work consensus algorithm featured in the Bitcoin and Ethereum networks is attractive in environments where participants may arbitrarily join or leave the network and require mining rewards as an incentive to process transactions. However, it also has significant disadvantages. Searching for a valid nonce, as described above, involves computationally expensive hash calculations that serve no other useful purpose and incur tremendous energy costs. Recent studies have concluded that the energy consumption of Bitcoin's mining network exceeds that

of entire countries [34]. Additionally, because proof-of-work consensus is non-deterministic, the blockchain can temporarily fork and clients must wait for a confirmation delay before they can be confident that their transaction cannot be reversed. These drawbacks, together with a different set of assumptions made in permissioned blockchains but not in public blockchains, has motivated the development of many other blockchain consensus protocols.

This section is organized around alternative means of selecting the next block's creator: proof of stake, proof of elapsed time, and proof of authority. We conclude with a summary of approaches based on ideas from traditional distributed systems. Proof of stake and proof of elapsed time can be used in either public or permissioned settings, while proof of authority and the more traditional protocols typically require a permissioned blockchain. We only summarize the most prominent alternatives here; a more comprehensive survey of approaches to consensus is available in Reference [23].

5.1 Proof of Stake

Proof-of-stake consensus algorithms are intended to replace the proof-of-work mechanism in public blockchains. They still allow anyone to join or leave the network and offer protection against Sybil attacks, but do not involve the energy consumption of proof of work. In a proof-of-stake system, a node claims the ability to add new blocks to the chain, and thus receive transaction fees as a reward, by staking cryptocurrency as a security deposit, e.g., by submitting a special transaction or invoking a smart contract. This node then becomes a *validator*, as opposed to a miner in proof-of-work systems. Proof-of-stake implementations typically assume that two-thirds of all staked tokens are held by honest participants, arguably a stronger assumption than the requirement in proof of work that a simple majority of participants are honest.

To make forward progress on the chain, a new validator is periodically chosen to propose the next block and broadcast it to the network. This validator can be chosen at random, often by sampling from a distribution in which each validator is weighted by the size of its security deposit. These deposits must be made well in advance of a node's participation as a potential validator and must remain in effect for a minimum period of time. This prevents sudden churn in the set of eligible validators and prevents certain attacks against the proof-of-stake protocol. The next validator can be automatically chosen and enforced by a smart contract or via a cryptographic primitive.

In some of the early proof-of-stake implementations, such as Peercoin [73], a proposed block is simply validated and added to the end of the chain, much like in a proof-of-work system. It is assumed that validators are incentivized to contribute to the successful operation of the blockchain to prevent their currency holdings (demonstrated by their stake) from decreasing in value. However, this situation leads to what is known as the "nothing at stake" problem—validators actually have no meaningful deterrent from engaging in certain kinds of malicious behavior. For example, if a validator sees two new blocks proposed on top of the same parent, which creates a fork in the chain, there is no reason not to propose new blocks as successors to both children, maximizing its prospects for block rewards. Without hashing power (which manifests itself as the rate of growth of each fork) to act as an arbiter, there is no reason to prefer one fork over the other.

Subsequent proof-of-stake consensus algorithms, such as the initial version of Tendermint [78] and Ethereum's Casper proposal [22], introduce two additional features. First, much like in traditional Byzantine fault-tolerant distributed systems, all validators must submit signed votes on whether or not to accept each new block. Each validator's vote is assigned a weight proportional to the size of its security deposit. A new block is only appended to the chain if the validators collectively holding at least two-thirds of all staked currency (as opposed to a simple two-thirds majority of validators in a traditional BFT setting) vote in favor. Second, when a validator has engaged in

malicious behavior, such as proposing or voting in favor of blocks on top of both sub-chains in a fork, it loses all or some portion of its security deposit as a penalty.

Several more recent proof-of-stake algorithms have made further improvements. Algorand [49] is a proof-of-stake algorithm that uses a cryptographic primitive, verifiable random functions, to periodically select a committee of participants who decide on the next block to add to the chain. The committee runs an efficient Byzantine agreement protocol among its members, meaning Algorand is able to achieve much higher transaction throughput than traditional blockchains based on proof of work. Forks are still possible but highly unlikely in Algorand. They only arise in a weakly synchronous network setting and can be easily reconciled.

Ouroboros [72] is a similar proof-of-stake algorithm, using secure multiparty computation to safely determine committee membership. Instead of electing a committee once per block, Ouroboros decides upon a committee once per *epoch*, a time period spanning several blocks. The committee for an epoch then assigns responsibility to a specific validator for appending each of the epoch's blocks to the chain. While a malicious validator could induce a fork by propagating different blocks to different portions of the blockchain's network, Ouroboros uses a longest chain rule to resolve forks, much like Ethereum. Because the probability of selection as a validator is proportional to the size of a participant's stake, and we assume no single party controls a majority of all staked cryptocurrency, no single malicious actor will serve enough turns as a validator to steadily grow the length of a fork, meaning the fork will be quickly discarded.

5.2 Proof of Elapsed Time

The concept of proof of elapsed time (PoET) [64] was introduced by Intel as a component in its Sawtooth Lake blockchain implementation [96], which can run as either a public or a permissioned blockchain. Where a traditional mining process effects a lottery at each block by forcing miners to perform expensive hashing computations in search of a nonce, PoET implements a lottery by leveraging Intel's secure enclave hardware (SGX).[11] Each node runs an instance of PoET in an enclave. When each new block is produced, each node receives a randomly chosen timeout duration from the software running in its local enclave. Once a node's assigned timeout has expired, it is allowed to propose the next block on the chain, assuming no other node has already done so in the interim. PoET allows a node to generate a cryptographic proof that it waited for the full duration of its assigned timeout, and only blocks accompanied with such a proof are accepted by the network.

Enclaves offer a number of advantages in PoET. An enclave can compute a cryptographic measurement of the code it hosts and it can shield sensitive data, such as private keys, from the host's operating system. This means that a node cannot tamper with the operation of the lottery, nor can it forge a proof of an elapsed timeout. Because enclaves, rather than computation, ensure the integrity of the lottery, PoET avoids the high energy costs associated with proof-of-work schemes. However, general concerns about SGX also apply to PoET. Suitable hardware is only available from a single vendor, Intel, and trusting an enclave means trusting Intel's hardware implementation as well as its management of the cryptographic keys that underpin every enclave's security.

5.3 Proof of Authority

In a Proof of Authority (PoA) protocol, a subset of nodes on the network are designated as fully trusted validators, i.e., authorities, and cooperate to maintain steady operation of the blockchain. Each validator takes a turn acting as the leader for a given time period. It aggregates the transactions that have newly appeared on the network and proposes the next block in the chain. The

[11] https://software.intel.com/en-us/sgx.

proposed block is accepted only if a majority of the validator set approves of the change. An initial list of validators, identified by their public keys, must be specified when the blockchain is initialized, and keys may be added or removed from the validator set with a majority vote.

PoA consensus is relatively simple, enables fast processing of transactions, and avoids the computational effort of proof of work. However, validators must be trusted not to engage in malicious behavior, as a proof-of-authority chain lacks the penalties of a proof-of-stake scheme and the incentive model of a proof-of-work scheme. This is why proof of authority is unsuitable in a public blockchain but very attractive in a permissioned model where validators are usually selected out of band and have a vested interest in the success of the blockchain. For example, each business participating in a shared chain may be assigned one keypair to use to run a validator. Prominent PoA implementations include Geth's Clique protocol,[12] Aura,[13] and the Coco framework [86].

5.4 Other Consensus Protocols

Many proposed blockchains draw more directly upon prior work in distributed computing, particularly Paxos [80] and Raft [97] as implementations of crash-tolerant consensus algorithms and PBFT [25] as a Byzantine fault-tolerant implementation. For example, newer versions of Tendermint[14] use a voting protocol heavily inspired by PBFT. Quorum [66], a permissioned blockchain platform developed by J. P. Morgan, offers consensus algorithms based on both PBFT and Raft. Several other new consensus protocols have been developed with blockchain as their primary motivation while using historical work in distributed systems as a foundation, such as HotStuff [127]. Finally, Thunderella [100] adopts a hybrid approach where under favorable conditions (three-fourths of network participants are honest, and an honest leader proposes transactions) transactions can be quickly confirmed with a simple committee signature scheme in two rounds of communication. In all other, hopefully uncommon, situations, Thunderella falls back to an underlying traditional blockchain that runs an algorithm-like proof of work or proof of stake.

5.5 Summary

In summary, there is a tradeoff between decentralization, anonymity, and scalability of both network peers and transaction throughput. This tradeoff is theoretically bound by technical limitations (e.g., an asynchronous environment with network partitions, the CAP Theorem [17], and the FLP impossibility result [45]), non-technical limitations such as malicious participants, and economic limitations as shown in References [18, 26]. Put simply, if a system requires every participant to maintain her own copy of state and to serve as her own authority, then every participant must agree upon the ordering and contents of every transaction. Eliminating this requirement would require trusting a subset of participants as a higher authority, leading back to centralization, or relaxing some other property such as strong consistency. All consensus protocols trade off among these design points, as shown in Table 2.

In public blockchains, the proof-of-work protocol imposes a cost on participation, computational power to search for a correct nonce, and rewards honest participation by allowing miners to collect transaction fees. Proof of stake is similar, but instead the cost is a security deposit of cryptocurrency that is confiscated if a participant behaves maliciously. Both protocols rely on principles from economics and game theory to ensure their viability and security. In theory, attacking a proof-of-stake system is more expensive than attacking a system built on proof of work, but an attack is still possible, and even a small disruption could destroy confidence in the blockchain, causing

[12] https://github.com/ethereum/EIPs/issues/225.
[13] https://wiki.parity.io/Aura.html.
[14] https://tendermint.com/.

Table 2. Comparison between Different Consensus Protocols

Property	Proof of Work	Proof of Stake	PoET	Proof of Authority	Paxos, Raft, PBFT Variants
Blockchain Type	Public	Both	Both	Permissioned	Permissioned
Peer Network Scalability	High	High	Medium	Medium	Low
Transaction Throughput	Low	High	Medium	High	High
Transaction Finality	Probabilistic	Probabilistic	Probabilistic	Absolute	Absolute
Cost for Participation	Yes	Yes	No	No	No
Adversary tolerance	≤50%	≤33%	≤33%	≤50%	≤50% (Paxos/Raft), ≤33% (PBFT)
Anonymous Participants	(Pseudo-) Anonymous	(Pseudo-) Anonymous	No	No	No
Trust Model	Untrusted	Untrusted	Semi-trusted	Semi-trusted	Semi-trusted
Limitations & Example Attacks	Forks, selfish mining, high energy consumption	Forks, nothing at stake	Forks, must use/trust special hardware	Must trust validators	Traditional distributed systems limitations

users to abandon it. Thus far, PoW has demonstrated a successful application of game theory in a distributed protocol, despite its low transaction rate. In permissioned settings, participants are authenticated, reducing the need for an explicit incentive mechanism. Permissioned protocols use out-of-band mechanisms to punish malicious behavior and therefore can use traditional distributed systems protocols to increase transaction throughput and improve scalability.

6 IMPROVING SCALABILITY

A typically designed public blockchain like Bitcoin or Ethereum offers strong guarantees but cannot process transactions with the same efficiency as other distributed systems. Throughput is limited by the size of blocks and the rate at which blocks are produced. These are system parameters that cannot be altered without compromising security. Transactions also suffer from high latency, because a user cannot be confident in the inclusion of their transaction within the main chain until additional confirmation blocks have been appended to the chain. Alternative consensus mechanisms can address the issue of latency, but transaction throughput cannot be improved without more significant changes in blockchain design. Numerous techniques have been proposed to improve blockchain throughput, including two broad classes. First, some systems have explored generalizing the structure of the relationships between blocks from a strict chain to a graph. Other systems seek to improve throughput by relaxing the constraint that every transaction is propagated to and processed by the entire network, known as *sharding* in the blockchain literature. In this section, we summarize work on both block graphs and sharding before briefly discussing some other approaches to improving scalability.

6.1 Block DAG Systems

In all of the systems described so far, each block declares itself as the successor to a parent block by including a hash of the parent's contents in its header. Moreover, these systems maintain the invariant that each parent block b_i has a unique child b_{i+1}, forming a chain. Under this approach, it is trivial to maintain a total ordering of all blocks and therefore a total ordering of all transactions in

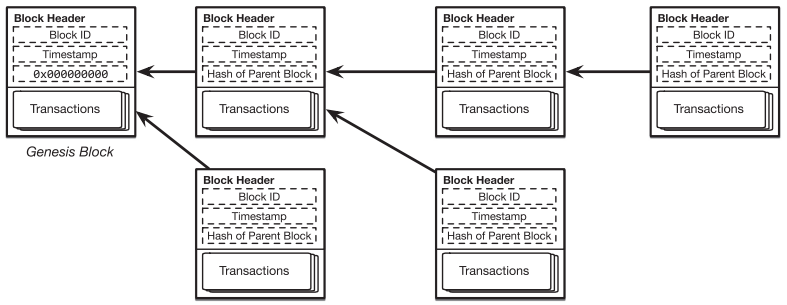

Fig. 9. A directed acyclic graph of blocks.

the distributed ledger. However, it is also impossible for miners[15] to process incoming transactions in parallel, because there can be only one canonical block appended to the chain for each round of consensus. All unsuccessful miners must discard their work and start anew with the updated version of the blockchain.

Some blockchain systems have explored an alternative design where a block may have multiple children, each produced by a different miner, as shown in Figure 9. Instead of a chain of blocks, miners now produce a more flexible directed acyclic graph (DAG) of blocks. This has several advantages, such as allowing miners to work in parallel to process incoming transactions. A block DAG also eliminates the need for a strictly deterministic consensus algorithm to avoid forks. Where forks are undesirable in a blockchain, they are part of routine operation in a block DAG system. The two concurrently mined blocks simply become two children of a common parent. Generalizing a blockchain into a block DAG comes at a price—there is no longer a clear total ordering of blocks and consequently no total ordering of transactions.

SPECTRE [116] was one of the first block DAG systems to be proposed, and it remains one of the most prominent examples of such a system. SPECTRE's most important feature is an algorithm to determine a non-transitive ordering of the blocks within the DAG. In many cases, this algorithm is not needed. Two blocks are often connected by a path in the graph, or their respective transaction sets do not conflict with one another. When conflicting transactions do emerge, SPECTRE selects one of them for execution (discarding the second transaction) based on the structure of the block DAG. Given two blocks a and b, every block in the DAG implicitly counts as a "vote" for either a or b based on a set of rules regarding its relative position to a and b. The transaction belonging to the block with votes from a majority of the DAG's blocks receives priority.

The SPECTRE developers later proposed a more powerful block DAG protocol, PHANTOM [118], that achieves a total ordering among all blocks. At a high level, PHANTOM distinguishes between two classes of blocks in the DAG, based on the intuitive notion that honest miners freely communicate and share knowledge of new blocks with one another. This implies that honestly mined blocks will have more connections to other blocks in the DAG and should be preferred in a total order.[16] Therefore, PHANTOM places all blocks believed to be honestly mined first in its total ordering by topologically sorting them. All remaining blocks are then also topologically sorted and appended to the end of the total order.

[15]We consistently use the term *miner* for the sake of simplicity, but this discussion of block DAGs also applies to a network of *validator* nodes running a consensus protocol other than proof of work.

[16]PHANTOM formalizes this distinction by defining the notion of a k-cluster in the block DAG and considering all blocks belonging to the largest such cluster to be honestly mined. The details are out of the scope of this brief summary.

Block DAGs have generated significant interest from blockchain designers and researchers, but they remain unproven as a complete scalability solution. These systems allow new blocks to be mined in parallel, but each member of the network must still maintain its own view of the complete block DAG. This raises the familiar concern of imposing excessive computation and storage demands on participants in the block DAG's network. As blocks can be created at a faster rate in a block DAG, the costs of participation are potentially even higher than in a normal blockchain. In addition, more frequent block creation means more traffic on the block DAG's overlay network and a higher communication burden for its members.

6.2 Sharding

A second approach to enabling parallel transaction processing, and therefore improving transaction throughput, is to partition the members of a blockchain's peer-to-peer network into disjoint subsets called *shards*. As new transactions are submitted to the network, they are each assigned to a shard for validation and addition to the distributed ledger. The shards operate independently, processing new transactions in parallel. Thus, a sharded blockchain system must have mechanisms to achieve consensus regarding which members belong to which shards and to determine which shard is responsible for processing which blockchain transaction.

The design of a sharded blockchain must carefully protect against the new types of attacks that are possible in this setting. For example, a malicious user may attempt to spend the same tokens twice by inducing two shards to independently and unknowingly process conflicting transactions. This is particularly challenging, because any coordination across shards to identify conflicting transactions is at odds with the parallelism that improves the blockchain's throughput. Moreover, because a shard by definition has fewer members than the full blockchain network, it is more vulnerable to compromise—it has less collective hashing power for a proof-of-work scheme and fewer nodes for a more traditional voting-based consensus protocol. If an attacker can directly control which shard she participates in, then she can target a particular shard and potentially hijack its consensus protocol.

Zilliqa [129] was one of the earliest proposals for a sharded blockchain. It supports smart contracts and shares many high-level similarities with Ethereum. Zilliqa's design centers around a subset of its members designated as a directory service (DS) committee. These members run a consensus protocol based on PBFT [25] among themselves to assign all members of the network to a shard and to assign transactions to shards. Each shard uses PBFT internally to approve transactions and forward them to the DS committee for serialization into a globally shared ledger. A member of the Zilliqa network earns the opportunity to serve on the DS committee by mining a special "DS block" in a typical proof-of-work scheme. Thus, proof of work in Zilliqa serves as the basis for a lottery that determines DS committee membership.

Several other sharded blockchain systems have emerged that make improvements to Zilliqa's design. Omniledger [74] only supports token exchanges rather than more general smart contracts but eliminates the need for a centralized committee to determine shard membership and to coordinate the assignment of transactions among shards. Instead, random shards are constructed using a secure, distributed algorithm [120]. Unlike Zilliqa, Omniledger partitions blockchain state across shards. Since a single transaction may now involve multiple shards, Omniledger introduces a protocol for atomic cross-shard transactions. Chainspace [1] adopts a similar strategy, but supports fully general smart contracts and features a simpler protocol for cross-shard transactions that does not require the participation of the transaction's sender. Finally, RapidChain [128] is a more recent system that occupies a middle ground. Blockchain state is partitioned across shards as in Omniledger and Chainspace, but shard membership is determined by a central committee, as

in Zilliqa. RapidChain's consensus protocols are carefully designed to minimize the number and size of messages exchanged within and across shards, reducing communication overhead.

Like block DAGs, sharding is a promising technique, but it faces ongoing challenges [65] and has not yet proven itself in a large-scale system. Shards may have to communicate with each other to validate transactions if the blockchain's state is partitioned, and this process must be efficient even though neither shard has access to the other's state. If inter-shard communication is too frequent or too expensive, then the scalability of the blockchain is limited. There are also concerns about system availability in sharded blockchains. A traditional blockchain is highly available, because every node has a complete view of the blockchain and can process any transaction. Under a sharding scheme, each transaction can only be processed by a particular shard, a much smaller collection of nodes that is more prone to outages, and if a shard is lost, a portion of the blockchain's state is permanently lost with it. Thus, strategies to replicate shards and to create backup archives of shard state are being developed.

6.3 Other Approaches

Bitcoin-NG [44] retains many elements of a blockchain: Blocks are organized into a strict sequence that is agreed upon by all members of the network, and new blocks are added to the chain sequentially, each by a specific member. The blockchain's operation is organized into a sequence of time periods known as *epochs*. A new epoch starts whenever a member of the network appends a new *key block* to the blockchain through proof-of-work. However, this block contains no transactions. Instead, it serves as a proof that its miner has been elected as the leader for the new epoch. The leader is allowed to unilaterally create a sequence of *microblocks* that contain actual transactions submitted by end-users. The epoch ends when a new key block has been mined, and the new block's miner becomes the leader for the following epoch. This has the advantage that several blocks are efficiently appended to the chain in sequence by one partcipant. However, this design also requires a carefully designed incentive scheme to discourage an epoch's leader from exploiting its position, e.g., by withholding microblocks to gain a head start in mining the next key block.

There are also several efforts to improve transaction throughput not by redesigning the blockchain itself, but instead by making more judicious use of it. *Payment channels,* such as Bitcoin's Lightning Network [103] and Ethereum's Raiden Network [107], are arguably the foremost example of this approach. Two parties who wish to frequently exchange tokens with one another initialize a payment channel by adding a new ledger entry to the blockchain. This ledger entry records an initial stake of tokens for each party. The tokens underlying each participant's stake are thus held in escrow. Then, one participant in the channel can freely write a "check" to the other, containing a cryptographic signature, declaring a payment of tokens. This declares a reallocation of a portion of the tokens held in escrow from the payer to the payee, but does not involve a blockchain transaction. After an arbitrary number of checks have been exchanged, either participant may close the payment channel by adding a new entry to the blockchain's ledger, declaring the final balance of funds owed to each party and substantiating this with a cyrptographic proof constructed from the sequence of checks previously exchanged out of band. When the second party corroborates this (or a timeout expires), the channel's funds are properly dispensed. The blockchain is only involved in creating the payment channel and settling any debts when the channel is terminated.

7 IMPROVING PRIVACY

In a traditional blockchain, every transaction and its associated data is propagated to and stored on every node in a blockchain's peer-to-peer network. Nodes must be able to validate and execute each transaction, so transactions cannot be encrypted to maintain confidentiality. As discussed in

Section 2.6, it is therefore possible to attribute blockchain transactions and their associated public keys to specific users through transaction graph analysis. Here, we examine work to enable private transactions that prevent such an analysis, even on public blockchains, while still preserving the consistency, availability, and other properties that make blockchains attractive. We first discuss Monero, a blockchain system that employs a combination of cryptographic primitives to conceal transaction details, then examine technologies that leverage recent advances in zero-knowledge cryptography, and finally discuss systems that use secure hardware enclaves to achieve privacy.

7.1 Monero

Monero [89] is a proof-of-work blockchain and associated cryptocurrency—originally based on the CryptoNote [125] platform—that offers fully private transactions by default. Its implementation incorporates a number of cryptographic primitives to hide the sender, receiver, and payment amount of every transaction. The destination of a transaction is a one-time use stealth address [32] derived from the receiving entity's true address. It is constructed so only the intended recipient can redeem the transaction's associated currency, but no external observer can trace the stealth address back to its owner. Similarly, the sender signs a transaction with a special type of ring signature [109] that prevents double spending of tokens. An external validator can attribute the ring signature to some member of a group of public/private key pairs but not to a specific member within that group. Thus, observers can trace a transaction back to a group of entities, but they cannot identify the true sender within this group.

Monero added Ring Confidential Transactions [53, 95] in 2017. These transactions conceal their payment amounts but can be still be validated for correctness by all members of the blockchain. This is possible through the use of Pedersen committments [101] and cryptographic range proofs, which prevent the unauthorized creation of new tokens. The ability to make transactions both private and verifiable comes at the expense of storage space, as the representation of a transaction now involves several cryptographic elements, such as signatures and proofs, that must be placed on the ledger. The Monero community is planning to make transactions more space-efficient by replacing one of the most verbose elements, range proofs, with Bulletproofs [19] in the near future.

7.2 Zero-knowledge Cryptography

Much of the work in implementing inherently private blockchain systems relies on zero-knowledge proofs, particularly zk-SNARKs [12, 99], which are a relatively modern cryptographic primitive that enables compact and efficient zero-knowledge proofs.[17] Zerocash [11] was the first implementation of a zk-SNARK-based cyrptocurrency supporting fully featured and completely anonymous transactions. Private transactions are still maintained on a public blockchain and benefit from the associated guarantees, but they do not consist of explicitly stated sender and receiver addresses or a payload amount. Instead, a transaction contains a zero-knowledge proof demonstrating that the sender is adhering to the proper protocol—all coins in the transaction are well formed, the sender is the true owner of these coins and has sufficient funds to back the transaction—without revealing sensitive transaction metadata. The research behind Zerocash has since been used as the basis for ZCash,[18] a popular cryptocurrency that offers fully private transactions.

Hawk [75] extends the application of zk-SNARKs to smart contracts. Hawk contracts are written much like normal contracts (although they cannot express fully general logic like a Solidity contract) and then compiled into a program in which contract participants run computations locally, using the blockchain only for the storage of encrypted data and a rendezvous point for

[17] For background information on zero-knowledge proofs, see Chapter 5 of Reference [83].
[18] https://z.cash.

zero-knowledge proofs to enforce the correctness of the contract's execution. However, Hawk's construction requires a partially trusted party to manage certain aspects of the contract's operation, unlike a more traditional smart contract with no trusted parties. Similarly, zkLedger [93] is an implementation of a distributed financial ledger where all transactions are private, but third-party auditors can query the network to obtain quantitative measurements of the transactions in aggregate, such as a sum or average, to identify suspicious activity. The information presented to auditors is backed by zero-knowledge proofs, although they use an alternative construction to zk-SNARKs.

7.3 Trusted Execution Environments

A third line of work in blockchain privacy centers on the use of Trusted Execution Environments (TEEs), such as the enclaves offered by Intel's SGX technology and Arm's TrustZone.[19] The application of enclaves to blockchain has been widely discussed [75, 86]. One major enclave-based blockchain system is Ekiden [28]. In Ekiden, a contract's transactions are executed by one or more *compute nodes*, each running the contract's code within a TEE. These nodes are distinct from the *consensus nodes* that maintain the blockchain. A TEE ensures the confidentiality and integrity of sensitive internal contract data, and a compute node need only store cryptographic attestations and checkpoints of correct contract execution on the blockchain. Ekiden contracts thus offer confidentiality and scalability, because their execution proceeds independently of the blockchain (unlike in Ethereum, where every contract transaction is executed by every node participating in the network). Because cryptographic proofs of proper contract execution are stored on the chain, contracts achieve consistency—there is a canonical sequence of transactions—and availability—contract state is reflected on the chain and does not depend on any specific compute node.

8 TOWARD ROBUST SMART CONTRACTS

Because smart contracts are intended to handle management of data and enforcement of rules in high-stakes situations, and because they are both difficult to implement correctly and impossible to modify once deployed, there is significant interest in applying techniques from formal methods and programming languages research to the domain of smart contracts. The general goal of these efforts is to allow developers to reason about and establish guarantees for the behavior of contracts before they are deployed. This interest further intensified after the theft of funds from `TheDAO`, a famous contract on the main Ethereum blockchain. Here, we summarize approaches belonging to three categories: formal analysis of existing contract code, translation of contract code into alternative languages that facilitate formal analysis, and alternative languages to express contract logic. We close with a brief discussion of a slightly different approach to contract defense involving bug bounties.

8.1 Analyzing Contract Code

A number of research efforts have attempted to analyze contracts developed within existing languages, most often Ethereum's Solidity, to establish formal properties and guarantees of behavior. One of the first efforts was Oyente [85], a symbolic execution tool for EVM bytecode that analyzes a contract's compiled form to identify potential security holes, such as improper error checking or susceptibility to corruption of internal state in the face of reentrancy. Oyente's authors ran their tool on approximately the first 20K contracts deployed on Ethereum, with approximately 8K contracts flagged as vulnerable. The authors observed a false positive rate of 6.4% when they compared the tool's output to manual analysis for a subset of these contracts.

[19] https://developer.arm.com/technologies/trustzone.

Grossman et al. [54] developed an algorithm to analyze execution traces of Solidity contracts to determine if a contract is *Effectively Callback Free* (ECF). Here, the authors define a callback as the reentrant invocation of code within a contract making an external call by the callee. The authors define a contract to be ECF if any execution trace involving callbacks can be replaced by an equivalent trace without callbacks. Two execution traces are equivalent if the contract's starting and ending states (as reflected by its internal data) are identical for both traces. This property has two important applications. First, if a contract is not ECF, then it is vulnerable to reentrancy attacks like those targeting `TheDAO`. Second, if a contract is ECF, then its behavior does not depend on any other contract, and we need study only the original contract's code.

Finally, Securify [122] is a tool for analyzing Ethereum contracts and proving properties about their behavior. It parses EVM bytecode to construct control flow and data flow graphs for a smart contract. A Securify user does not prove contract security properties directly. Instead, she expresses properties of the control flow and dataflow graphs in a domain-specific language and invokes the tool to determine if they hold. Securify's developers argue that these graph properties are simpler to verify and can effectively approximate many useful security properties of the source contract. For example, Securify can identify the bug that affected `TheDAO` contract by searching for an execution trace where a contract field is updated immediately after a call to an external contract.

8.2 Translating Contract Code

Other work has focused on translating existing contracts into languages and frameworks that are more amenable to formal analysis. Solidity* [13] is a prototype tool that converts a subset of Solidity code into code expressed in F* [119], a dependently typed programming language in which certain desirable properties can be verified directly by the type system, such as proper control flow around error handling. However, verifying any contract-specific properties, such as guarantees about the integrity of internal state, cannot be automated. In this case, the developer must write a contract specification and associated proof by hand in F*.

Similarly, Scilla [115] is a proposed intermediate language for smart contracts. The authors envision that Solidity code will be translated to Scilla code, which is then analyzed for correctness, before ultimately being translated to EVM bytecode. In Scilla, contracts are written explicitly as state machines. A state transition prompts updates to the contract's internal state. Contracts may interact by emitting messages at the end of a state transition and receiving a message at the beginning of a state transition. The intent is to isolate the contract's computation from its interaction with external contracts, eliminating reentrancy issues. A Scilla contract is analyzed through translation into Coq,[20] a computational theorem prover. The contract's properties and associated proofs are then manually written by the contract developer. Scilla's authors are working towards automatic translation between Solidity and Scilla and between Scilla and Coq, but both are currently manual.

8.3 Alternative Contract Languages

Several blockchain platforms expect developers to work directly in alternative programming languages to write their smart contracts. These languages are designed to protect programmers from security vulnerabilities as well as other bugs and to lend themselves to straightforward formal analysis. Some of these languages [14, 47] are domain-specific and declarative. Developers express contract logic as a set of rules and operations rather than in code. Other languages support general-purpose programming and borrow heavily from the domain of functional programming,

[20] https://coq.inria.fr.

e.g., favoring permanent variable bindings over mutable state and higher-order functions over loops. Two examples are Kadena's Pact [104] language and Cardano's Plutus [24] language, which adopt syntax and features from Lisp and Haskell, respectively. This makes the behavior of code in these languages easier to reason about than code written in a more traditional imperative language. The Tezos blockchain adopts an even more restrictive but easily analyzed language called Michelson [50]. Michelson is intended to serve as a "higher-level bytecode"—it has a full-fledged type system but is stack-based, like EVM or JVM bytecode. Michelson is simple enough for contract code to be analyzed with tools like Coq, and while it can be written by hand, the Tezos developers anticipate it will also serve as a target for compilers.

Obsidian [29, 30] is a contract language under development that seeks a middle ground between the approaches described above. Obsidian contracts resemble imperative code but must adhere to a state machine structure. The developer explicitly describes the different states that a contract may assume during its operation. Transitions between states are expressed as blocks of code that manipulate the internal state of the contract, much like methods in a traditional language. Obsidian applies techniques from programming languages research to detect errors that would normally emerge at run time, after the contract has been deployed, at compile time. For example, the type system enforces programmer-specified constraints on when each transition may be invoked, triggering execution of the associated code.

8.4 Bug Bounties

Not all ongoing efforts to develop secure smart contracts rely on traditional formal methods. Breindenbach et al. [16] have proposed a methodology for identifying bugs in contracts and rewarding the users who find them. They employ a strategy called *N-of-N-version Programming*: develop N independent implementations of the same smart contract and wrap them in a parent contract. The parent is accessible by end-users and mediates all access to the N implementations. When a user invokes an operation on the parent contract, all N underlying implementations perform the operation in parallel. If the outputs of all N versions match, then the operation proceeds normally. However, if any of the versions disagree in their output, the situation is treated as a bug. The operation is safely aborted and the end-user is compensated with a bug bounty. Breindenbach et al. have implemented a prototype, *Hydra*, that runs on the Ethereum blockchain and thus is able to automatically disburse bounties. Of course, this strategy relies on the assumption that the N contract versions will not exhibit correlated failures. When using *Hydra*, the authors implemented one contract in three programming languages (Solidity, Serpent, and Vyper) to avoid this.

9 COMMERCIAL BLOCKCHAIN EFFORTS

Today, there are a myriad of commercial blockchain platforms offered by a large variety of companies including startups, the traditional large technology companies, and financial firms. In this section, we discuss some of the most prominent blockchain products as representative examples.

9.1 Blockchain as a Cloud Service

Amazon [2], IBM [63], Google [110], Microsoft [87], and Oracle [98] all offer blockchain as a service on their respective cloud platforms. These services allow a customer to quickly instantiate a blockchain network in the cloud. All of the resources and services underpinning the network's operation are seamlessly provisioned using the relevant vendor's cloud platform. For example, a new Ethereum network's miners are provisioned as cloud-based virtual machines. An end-user then interacts with the cloud-based blockchain network just as it would with a decentralized, peer-to-peer network, typically by submitting transactions and reading blockchain state using an RPC protocol. Of course, this setup—where every blockchain node is running within the infrastructure

of a single vendor—arguably defeats blockchain's primary purpose of achieving decentralization and eliminating the need for trust.

Many vendors also take this opportunity to closely integrate their blockchain products with their other cloud services. Azure's blockchain platform is a good example of this. Microsoft allows entities to identify themselves to the blockchain using Azure's Active Directory service and stores private keys in Azure's Key Vault service. As transactions modify the blockchain's state over time, these changes can be translated into event streams that are consumed by external Azure services. Finally, the blockchain is optionally synchronized with an Azure-hosted database to support more traditional query-based and analytics workloads.

9.2 Microsoft Coco

Coco [86] is a blockchain framework developed by Microsoft that is distinguished by its extensive use of Trusted Execution Environments (TEEs), such as Intel SGX. While it was announced with a white paper in 2017, its code has yet to be released as of the writing of this tutorial. Coco's target use case is permissioned blockchains in which the core participants are enterprise businesses with some degree of trust in one another, eliminating the need for Sybil-resistant consensus protocols like proof of work. Each participant in the network runs a validator node (VN) that executes both the consensus protocol and smart contract transactions within a TEE. This makes Coco different from systems like Ekiden [28], where TEEs are used only for the latter. Because a TEE can provide a cryptographic attestation proving that a validator node is executing the proper code, all VNs in the network are fully trusted. Consensus in a Coco blockchain can therefore avoid expensive Byzantine fault-tolerant algorithms in favor of crash fault-tolerant algorithms such as Paxos and Raft.

Coco's central component is a persistent, globally consistent data store that is replicated across the TEEs running within all validator nodes. This data store is used to maintain two logical ledgers. The first stores contract state and the results of all transactions, much like in a traditional blockchain. The second ledger stores all of the network's configuration parameters such as the identities of the authorized validator nodes (to enforce the chain's membership permissions) and a voting policy for updating the network's configuration. The latter allows the network's members to propose changes—such as adding a new member or changing the consensus protocol—and seamlessly enact them without interrupting the operation and forward progress of the blockchain. Furthermore, Coco's design protects the confidentiality and integrity of both ledgers, because the contents of the data store are only decrypted within the network's TEEs. The data are only persisted in encrypted form, and Coco leverages a threshold encryption scheme to ensure that decryption requires the cooperation of multiple validator nodes; a single node cannot leak ledger data unilaterally.

9.3 Quorum

J. P. Morgan has forked and modified the codebase of Ethereum's Go implementation to produce Quorum [67], a permissioned blockchain that is general-purpose but specifically marketed to the financial sector. Quorum originally removed Ethereum's proof-of-work consensus mechanism and instead used a custom algorithm, QuorumChain, in which a set of authorized nodes proposed new blocks to append to the chain and a set of voter nodes had to approve each of these changes. While it avoided the computational costs and low transaction processing rate of proof-of-work consensus, a number of security and performance problems with QuorumChain were identified by both the Quorum developers and independent researchers [23]. As a result, Quorum has shifted to an implementation of the Raft consensus protocol in later versions of the product.

As financial transactions may involve sensitive information like pricing agreements, Quorum also has a focus on privacy. While the developers are exploring a future integration with zk-SNARK

technology from ZCash [68], Quorum currently implements two classes of transaction: public and private. Public transactions on the Quorum blockchain are identical to standard transactions on an Ethereum chain. When a node initiates a private transaction, however, it encrypts the transaction's payload such that only a designated group of entities—identified by their respective public keys—can decrypt it. The payload itself is then distributed to these parties using Constellation,[21] a system for confidential point-to-point communications. Each authorized node locally executes the associated smart contract code. Meanwhile, a hash of these contents is propagated throughout the complete Quorum network and goes on the blockchain, but does not induce any transaction processing among the ordinary nodes. The members of the blockchain's network no longer share an identical body of state. Public state remains globally consistent, but each member maintains its own body of private state based on which private transactions it was a party to.

9.4 Hyperledger Fabric

Hyperledger [61, 62, 84] is an industrial blockchain development consortium organized by the Linux Foundation. Rather than concentrating on a single implementation, Hyperledger functions as an umbrella project for several blockchain permissioned, business-oriented blockchain systems. Its most well-known and actively developed project is Fabric [3], an open-source blockchain platform created and managed by IBM. Fabric serves as a platform for smart contracts and does not stipulate any particular choice of consensus algorithm. Instead, a Fabric deployment requires what its developers have termed an *ordering service*—a software component that collects transactions submitted by clients and propagates them in a specific, well-formed, and globally consistent order to all members of the network. Fabric currently features a built-in ordering service that uses Apache Kafka,[22] a system for scalable message streaming, but the developers plan to add support for Raft.

The Hyperledger developers argue that the *order-execute* approach of traditional blockchain systems—in which every transaction is put into an order established through consensus and then executed on every node in the network—is responsible for many of their limitations, such as a lack of transaction privacy and limited scalability. Additionally, this arrangement requires smart contract transactions to be fully deterministic. If their execution produced different results on different nodes, then the nodes' state would diverge and consistency would be lost.[23] Fabric addresses these limitations by introducing an *execute-order-validate* approach to transaction processing. In this regime, a blockchain application consists of logic expressed in a smart contract, known as a *chaincode* in Fabric's terminology, and an *endorsement policy* specifying which nodes within the network are allowed to execute the application's transactions and must endorse these transactions before they are permitted to take effect. Chaincodes can be written in a variety of general-purpose programming languages, such as Go or Java, and are provided with the abstraction of a persistent key/value store, which is maintained on a distributed ledger.

This alternative design has a number of consequences. First, it is possible for inter-transaction conflicts to occur. Two transactions against the same chaincode and batched into the same block may each try to read and update the same key/value pair. In this case, the first transaction within the block will take effect, while the second will be thrown out in the validation phase. Second, because validation occurs after blocks have been assembled by the ordering service, a peer will retain both valid and invalid transactions in its local copy of the blockchain. This is unlike a traditional

[21] https://github.com/jpmorganchase/constellation.
[22] https://kafka.apache.org.
[23] This is why Ethereum contracts, for example, are executed in a specially designed virtual machine and written in a domain-specific language like Solidity.

blockchain, where invalid transactions are discarded through the consensus protocol. Finally, while transaction execution is restricted to an authorized set of nodes, the information it reads and writes is still visible to the network at large. Fabric has introduced the concept of a *channel* to remedy this. Fabric's channels call for the ordering service to deliver a chaincode's transactions—already executed but not validated—only to an authorized group of peers. Like Quorum's private transactions, this means that nodes now diverge with respect to their local state.

10 CONCLUSION

This article began with a an overview of the key blockchain concepts, motivated by Bitcoin as an example system, before diving into a more detailed study of Ethereum and its smart contracts. We then compared the capabilities and limitations of blockchains to traditional distributed systems. Next, we transitioned into a high-level survey of four key blockchain challenges: improving the efficiency of consensus, making blockchains more scalable, enabling privacy in transactions, and ensuring the security of smart contracts without sacrificing the blockchain's desirable properties. We presented ongoing research and development efforts to address each of these challenges. Finally, we discussed prominent commercial blockchain offerings, many of which address the challenges we have described.

While blockchains are built on classical ideas from distributed computing and cryptography, they represent an application of these ideas to create systems with a new and powerful set of capabilities. In a public blockchain, anyone can submit a transaction to be added to the ledger, anyone can maintain their own copy of this ledger, and anyone can process transactions to update the ledger. No participant needs to trust any other participant, and no single participant can exert unilateral control over the ledger. In private blockchains and traditional distributed databases, a participant must be authenticated to make use of the system, but a mechanism to achieve consensus on the order of events is still required.

As we have discussed, the power of public blockchains comes at a price. With no central authority and no trust among the participants, a public blockchain is hard to govern and manage. Changes to the blockchain's operation require a majority of the network's miners to follow the newly proposed protocol, and it can be difficult to achieve a sufficient degree of agreement within the community. Additionally, public blockchains require an explicit set of incentives to process transactions (e.g., Ethereum's gas model) and/or punishments for bad behavior (e.g., confiscating a validator's funds in a proof-of-stake consensus model). These incentives and punishments complicate the system's design and need to be carefully designed to be effective.

The proof-of-work consensus model comes with its own set of tradeoffs. It effectively eliminates Sybil attacks by tying influence in the consensus process to computational power, which means an adversary derives no benefit from acting under multiple simulated identities. However, the collective computational power of the blockchain network cannot be devoted to any other useful purpose and consumes significant energy. Proof of work is also probabilistic—there is no explicitly determined leader for each round of consensus, and this gives rise to forks in the blockchain. This situation has significant consequences. It means we must limit the transaction processing rate to make forks less likely and maintain the security of the main chain, and it means that a client must wait for confirmation blocks to appear before she can be confident in the durability of a transaction. Hence, a traditional proof-of-work blockchain has limited transaction throughput and high transaction latency.

Consensus is not the only reason for the limited performance of public blockchains; they also impose the constraint that every transaction is processed by and recorded on every node. The two factors that determine the rate of transaction processing—block size and block creation rate—must both be limited to accommodate this constraint. The inter-block delay needs to be large enough

to allow each block to propagate throughout the peer-to-peer network underlying the blockchain, and blocks must be small enough to be processed by and stored on reasonable hardware platforms.

Thus, public blockchains exhibit limited performance but offer stronger security guarantees and an ability to operate without trust. In contrast, distributed databases provide strong performance but have weaker security and require trust. Permissioned blockchains are a middle ground. They improve performance at the expense of security, as they do not tolerate Sybil attacks and depend upon a degree of trust among the network's members. However, both public and permissioned blockchains are vibrant areas of research and development. We have already seen new designs for blockchains that have the potential to improve performance without sacrificing security. Blockchains may yet be able to accommodate real-world applications deployed at large scale.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous reviewers as well as Michael Andersen, Gabe Fierro, Hyung-Sin Kim, Sam Kumar, and K. Shankari for providing feedback on drafts of this tutorial.

REFERENCES

[1] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis. 2018. Chainspace: A sharded smart contracts platform. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'18)*. Internet Society, 15.

[2] Amazon. 2018. Blockchain on AWS. Retrieved from: https://aws.amazon.com/blockchain/.

[3] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. Cocco, and J. Yellick. 2018. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the 13th EuroSys Conference (EuroSys'18)*. ACM, 30:1–30:15.

[4] Andreas M. Antonopoulos. 2017. *Mastering Bitcoin: Programming the Open Blockchain*. O'Reilly.

[5] Andreas M. Antonopoulos and Gavin Wood. 2018. *Mastering Ethereum: Building Smart Contracts and Dapps*. O'Reilly.

[6] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A survey of attacks on Ethereum smart contracts. In *Principles of Security and Trust*. Lecture Notes in Computer Science, Vol. 10204. Springer, 164–186.

[7] Adam Back et al. 2002. Hashcash—A denial of service counter-measure. http://www.hashcash.org/papers/hashcash.pdf.

[8] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. 2017. Consensus in the age of blockchains. *ArXiv* (Nov. 2017). Retrieved from: https://arxiv.org/abs/1711.03936.

[9] Massimo Bartoletti and Livio Pompianu. 2017. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In *Financial Cryptography and Data Security*. Springer, 494–509.

[10] Dave Bayer, Stuart Haber, and W. Scott Stornetta. 1993. Improving the efficiency and reliability of digital time-stamping. In *Sequences II*. Springer, 329–334.

[11] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'14)*. IEEE, 459–474.

[12] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2013. Succinct non-interactive arguments for a Von Neumann architecture. *IACR Cryptology ePrint Archive* (2013), 879.

[13] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin. 2016. Formal verification of smart contracts: Short paper. In *Proceedings of the ACM Workshop on Programming Languages and Analysis for Security (PLAS'16)*. ACM, 91–96.

[14] Alex Biryukov, Dmitry Khovratovich, and Sergei Tikhomirov. 2017. Findel: Secure derivative contracts for Ethereum. In *Financial Cryptography and Data Security*. Springer, 453–467.

[15] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. 2015. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Proceedings of the IEEE Symposium on Security and Privacy*. 104–121.

[16] Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. 2018. Enter the Hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *Proceedings of the 27th USENIX Security Symposium*. USENIX, 1335–1352.

[17] Eric A. Brewer. 2000. Towards robust distributed systems. In *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'00)*, Vol. 7.

[18] Eric Budish. 2018. *The Economic Limits of Bitcoin and the Blockchain*. Technical Report. National Bureau of Economic Research.
[19] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'18)*. IEEE, 319–338.
[20] Vitalik Buterin. 2014. A Next-Generation Smart Contract and Decentralized Application Platform. Retrieved from: https://github.com/ethereum/wiki/wiki/White-Paper.
[21] Vitalik Buterin. 2014. Toward a 12-second Block Time. Retrieved from: https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time.
[22] V. Buterin and V. Griffith. 2017. Casper the friendly finality gadget. *ArXiv* (Oct. 2017). Retrieved from: https://arxiv.org/abs/1710.09437.
[23] Christian Cachin and Marko Vukolic. 2017. Blockchain consensus protocols in the wild. *ArXiv* (7 2017). https://arxiv.org/abs/1707.01873.
[24] Cardano Foundation. 2018. Plutus Introduction. Retrieved from: https://cardanodocs.com/technical/plutus/introduction.
[25] Miguel Castro and Barbara Liskov. 1999. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI'99)*. USENIX, 173–186.
[26] Christian Catalini and Joshua S. Gans. 2016. *Some Simple Economics of the Blockchain*. Technical Report. National Bureau of Economic Research.
[27] T. Chen, Y. Zhu, Z. Li, J. Chen, X. Li, X. Luo, X. Lin, and X. Zhange. 2018. Understanding Ethereum via graph analysis. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'18)*. IEEE, 1484–1492.
[28] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. 2018. Ekiden: A platform for confidentialityPreserving, trustworthy, and performant smart contract execution. *ArXiv* (7 2018). https://arxiv.org/abs/1804.05141
[29] Michael Coblenz. 2017. Obsidian: A safer blockchain programming language. In *Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C'17)*. ACM, 97–99.
[30] Michael J. Coblenz, Jonathan Aldrich, Joshua Sunshine, and Brad A. Myers. 2018. User-centered design of permissions, typestate, and ownership in the Obsidian blockchain language. In *HCI for Blockchain: Studying, Designing, Critiquing and Envisioning Distributed Ledger Technologies Workshop at CHI 2018*. ACM.
[31] M. Conti, E. Sandeep Kumar, C. Lal, and S. Ruj. 2018. A survey on security and privacy issues of bitcoin. *IEEE Commun. Surv. Tutor.* 20, 4 (2018), 3416–3452.
[32] Nicolas T. Courtois and Rebekah Mercer. 2017. Stealth address and key management techniques in blockchain systems. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP'17)*. 559–566.
[33] K. Croman, C. Decker, I. Eyal, A. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Sirer et al. 2016. On scaling decentralized blockchains. In *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 106–125.
[34] Alex de Vries. 2018. Bitcoin's growing energy problem. *Joule* 2, 5 (2018), 801–805.
[35] Michael del Castillo. 2016. The DAO Attacked: Code Issue Leads to $60 Million Ether Theft. Retrieved from: https://www.coindesk.com/dao-attacked-code-issue-leads-60-million-ether-theft.
[36] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang. 2018. Untangling blockchain: A data processing view of blockchain systems. *IEEE Trans. Knowl. Data Eng.* 30, 7 (7 2018), 1366–1385.
[37] Cynthia Dwork and Moni Naor. 1992. Pricing via processing or combatting junk mail. In *Proceedings of the International Cryptology Conference*. Springer, 139–147.
[38] A. Elliot. 2018. The blockchain technology revolution is about to remake the stock market. *Investor's Business Daily* (24 Dec. 2018). Retrieved from: https://www.investors.com/news/technology/blockchain-technology-blockchain-stock-market-revolution/.
[39] Ethereum Foundation. 2017. Serpent. Retrieved from: https://github.com/ethereum/serpent.
[40] Ethereum Foundation. 2018. Common Patterns. Retrieved from: http://solidity.readthedocs.io/en/v0.4.24/common-patterns.html.
[41] Ethereum Foundation. 2018. Patricia Tree. Retrieved from: https://github.com/ethereum/wiki/wiki/Patricia-Tree.
[42] Ethereum Foundation. 2018. Solidity. Retrieved from: http://solidity.readthedocs.io.
[43] Ethereum Foundation. 2018. Vyper. Retrieved from: https://github.com/ethereum/vyper.
[44] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert Van Renesse. 2016. Bitcoin-NG: A scalable blockchain protocol. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*. USENIX, 45–59.
[45] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1982. *Impossibility of Distributed Consensus with One Faulty Process*. Technical Report. Massachusetts Institute of Technology Cambridge Lab for Computer Science.

[46] Pedro Franco. 2014. *Understanding Bitcoin*. Wiley.
[47] C. K. Frantz and M. Nowostawski. 2016. From institutions to code: Towards automated generation of smart contracts. In *Proceedings of the IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W'16)*. IEEE, 210–215.
[48] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the security and performance of proof of work blockchains. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. ACM.
[49] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. 2017. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17)*. ACM, 51–68.
[50] L. M. Goodman. 2014. Tezos—A self-amending crypto-ledger. (9 2014). Retrieved from: https://tezos.com/static/papers/white_paper.pdf.
[51] Timothy Green. 2018. A safe way to bet on blockchain amid bitcoin, cryptocurrency craze. *CBS News* (4 Jan. 2018). Retrieved from: https://www.cbsnews.com/news/a-safe-way-to-bet-on-blockchain-amid-bitcoin-cryptocurrency-craze.
[52] Andy Greenberg and Gwern Branwen. 2015. Bitcoin's creator Satoshi Nakamoto is probably this unknown Australian genius. *Wired* (12 Dec. 2015). Retrieved from: https://www.wired.com/2015/12/bitcoins-creator-satoshi-nakamoto-is-probably-this-unknown-australian-genius/.
[53] Gregory Maxwell. 2015. Confidential Transactions. Retrieved from: https://people.xiph.org/ greg/confidential_values.txt.
[54] S. Grossman, I. Abraham, G. Golan-Gueta, Y. Michalevsky, N. Rinetzky, M. Sagiv, and Y. Zohar. 2017. Online detection of effectively callback free objects with applications to smart contracts. *Proc. ACM Prog. Lang.* 2, POPL, Article 48 (Dec. 2017), 28 pages.
[55] Stuart Haber and W. Scott Stornetta. 1990. How to time-stamp a digital document. In *Proceedings of the Conference on the Theory and Application of Cryptography*. Springer, 437–455.
[56] Stuart Haber and W. Scott Stornetta. 1997. Secure names for bit-strings. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*. ACM, 28–35.
[57] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse attacks on bitcoin's peer-to-peer network. In *Proceedings of the USENIX Security Symposium*. 129–144.
[58] Maurice Herlihy. 2018. Blockchains from a Distributed Computing Perspective. Retrieved from: https://cs.brown.edu/courses/csci2952-a/papers/perspective.pdf.
[59] Alyssa Hertig. 2016. Ethereum's Two Ethereums Explained. Retrieved from: https://www.coindesk.com/ethereum-classic-explained-blockchain.
[60] Yoichi Hirai. 2018. Bamboo: A Morphing Smart Contract Language. Retrieved from: https://github.com/pirapira/bamboo.
[61] Hyperledger Architecture Working Group. 2017. Hyperledger Architecture, Volume 1. Retrieved from: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf.
[62] Hyperledger Architecture Working Group. 2018. Hyperledger Architecture, Volume 2. Retrieved from: https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf.
[63] IBM. 2019. IBM Blockchain Platform. Retrieved from: https://www.ibm.com/blockchain/platform.
[64] Intel. 2018. PoET 1.0 Specification. Retrieved from: https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html.
[65] Yaoqi Jia. 2018. Op Ed: The many faces of sharding for blockchain scalability. *Bitcoin Mag.* (20 Mar. 2018). Retrieved from: https://bitcoinmagazine.com/articles/op-ed-many-faces-sharding-blockchain-scalability.
[66] J. P. Morgan Chase. 2018. Quorum. Retrieved from: https://www.jpmorgan.com/global/Quorum.
[67] J. P. Morgan Chase. 2018. Quorum Whitepaper. Retrieved from: https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf.
[68] J. P. Morgan Chase. 2018. Quorum-ZSL Integration: Proof of Concept. Retrieved from: https://github.com/jpmorganchase/zsl-q/blob/master/docs/ZSL-Quorum-POC_TDD_v1.3pub.pdf.
[69] Ghassan Karame. 2016. On the security and scalability of bitcoin's blockchain. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. 1861–1862.
[70] Ghassan O. Karame and Elli Androulaki. 2016. *Bitcoin and Blockchain Security*. Artech House.
[71] J. Katz, A. Menezes, P. Van Oorschot, and S. Vanstone. 1997. *Handbook of Applied Cryptography*. CRC Press.
[72] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Proceedings of the International Cryptology Conference*. 357–388.
[73] Sunny King and Scott Nadal. 2012. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. Retrieved from: https://peercoin.net/assets/paper/peercoin-paper.pdf.

[74] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. 2018. OmniLedger: A secure, scale-out, decentralized ledger via sharding. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'18)*. 583–598.
[75] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'16)*. IEEE, 839–858.
[76] James F. Kurose and Keith W. Ross. 2013. *Computer Networking: A Top-Down Approach* (6th ed.). Pearson.
[77] M. C. Kus Khalilov and A. Levi. 2018. A survey on anonymity and privacy in bitcoin-like digital cash systems. *IEEE Commun. Surv. Tutor.* 20, 3 (2018), 2543–2585.
[78] Jae Kwon. 2013. Tendermint: Consensus without Mining. Retrieved from: https://tendermint.com/static/docs/tendermint.pdf.
[79] L. Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM* 21, 7 (1978), 558–565.
[80] L. Lamport. 1998. The part-time parliament. *ACM Trans. Comput. Syst.* 16, 2 (May 1998), 133–169.
[81] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *ACM Trans. Prog. Lang. Syst.* 4, 3 (July 1982), 382–401.
[82] Timothy B. Lee. 2017. Iced tea company rebrands as long blockchain and stock price triples. *Ars Technica* (21 Dec. 2017). Retrieved from: https://arstechnica.com/tech-policy/2017/12/iced-tea-company-stock-triples-after-adding-blockchain-to-name.
[83] Yehuda Lindell. 2017. *How to Simulate It—A Tutorial on the Simulation Proof Technique*. Springer, 277–346.
[84] Linux Foundation. 2019. Home—Hyperledger. Retrieved from: https://www.hyperledger.org.
[85] Loi Luu, Duc Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. ACM, 254–269.
[86] Microsoft. 2017. The COCO Framework. Retrieved from: https://github.com/Azure/coco-framework/blob/master/docs/Coco%20Framework%20whitepaper.pdf.
[87] Microsoft. 2018. Blockchain Technology and Applications. Retrieved from: https://azure.microsoft.com/en-us/solutions/blockchain/.
[88] Ron Miller. 2018. IBM teams with Maersk on new blockchain shipping solution. *TechCrunch* (Aug. 2018). Retrieved from: https://techcrunch.com/2018/08/09/ibm-teams-with-maersk-on-new-blockchain-shipping-solution/.
[89] Monero. 2018. Monero—Secure, private, untraceable. Retrieved from: https://getmonero.org/.
[90] Satoshi Nakamoto. 2009. Bitcoin: A Peer–to–Peer Electronic Cash System. Retrieved from: https://bitcoin.org/bitcoin.pdf.
[91] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. 2016. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press.
[92] Arvind Narayanan and Jeremy Clark. 2017. Bitcoin's academic pedigree. *Commun. ACM* 60, 12 (2017), 36–45.
[93] Neha Narula, Willy Vasquez, and Madars Virza. 2018. zkLedger: Privacy-preserving auditing for distributed ledgers. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*. USENIX, 65–80.
[94] T. Neudecker and H. Hartenstein. 2019. Network layer aspects of permissionless blockchains. *IEEE Commun. Surv. Tutor.* 21, 1 (2019), 838–857.
[95] Shen Noether and Adam Mackenzie. 2016. Ring confidential transactions. *Ledger* 1, 0 (2016).
[96] Kelly Olson, Mic Bowman, James Mitchell, Dan Middleton, and Montgomery Cian. 2018. Sawtooth: An Introduction. Retrieved from: https://www.hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_WhitePaper.pdf.
[97] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'14)*. USENIX, 305–319.
[98] Oracle. 2019. Autonomous Blockchain Service. Retrieved from: https://www.oracle.com/cloud/blockchain.
[99] B. Parno, J. Howell, C. Gentry, and M. Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'13)*. 238–252.
[100] Rafael Pass and Elaine Shi. 2018. Thunderella: Blockchains with optimistic instant confirmation. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'18)*.
[101] Torben Pryds Pedersen. 1992. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the Conference on Advances in Cryptology (CRYPTO'91)*, Joan Feigenbaum (Ed.). Springer Berlin, 129–140.
[102] Lynne Peeples. 2018. Ocean plastic is a huge problem. Bitcoin could be part of the solution. *NBC News* (16 Oct. 2018). Retrieved from: https://www.nytimes.com/2018/06/27/business/dealbook/blockchains-guide-information.html.
[103] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning nework: Scalable off-chain instant payments. (Jan. 2016). Retrieved from: https://lightning.network/lightning-network-paper.pdf.
[104] S. Popejoy. 2017. The Pact smart contract language. (12 2017). Retrieved from: https://kadena.io/docs/Kadena-PactWhitepaper.pdf.

[105] Ben Popken. 2017. Why did bitcoin fork today and what is bitcoin cash? *NBC News* (1 Aug. 2017). Retrieved from: https://www.nbcnews.com/business/consumer/why-bitcoin-forking-today-what-bitcoin-cash-n788581.
[106] Nathaniel Popper. 2018. What is the blockchain? Explaining the tech behind cryptocurrencies. *The New York Times* (27 June 2018). Retrieved from: https://www.nytimes.com/2018/06/27/business/dealbook/blockchains-guide-information.html.
[107] Raiden. 2019. The Raiden Network. Retrieved from: https://raiden.network/.
[108] Fergal Reid and Martin Harrigan. 2013. An analysis of anonymity in the bitcoin system. In *Security and Privacy in Social Networks*. Springer, 197–223.
[109] R. Rivest, A. Shamir, and Y. Tauman. 2001. How to leak a secret. In *Advances in Cryptology*. Springer, 552–565.
[110] Jeff John Roberts. 2018. Google expands blockchain push with digital asset tie-up. *Fortune* (July 2018). Retrieved from: http://fortune.com/2018/07/23/google-cloud-digital-asset/.
[111] Dorit Ron and Adi Shamir. 2013. Quantitative analysis of the full bitcoin transaction graph. In *Financial Cryptography and Data Security*. Lecture Notes in Computer Science, Vol. 7859. Springer, 6–24.
[112] Kevin Roose. 2018. Kodak's dubious cryptocurrency gamble. *The New York Times* (30 Jan. 2018). Retrieved from: https://www.nytimes.com/2018/01/30/technology/kodak-blockchain-bitcoin.html.
[113] Fred B. Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.* 22, 4 (Dec. 1990), 299–319. DOI: https://doi.org/10.1145/98163.98167
[114] Ilya Sergey and Aquinas Hobor. 2017. A concurrent perspective on smart contracts. In *Financial Cryptography and Data Security. Lecture Notes in Computer Science*, Vol. 10323. Springer, 478–493.
[115] I. Sergey, A. Kumar, and A. Hobor. 2018. Scilla: A smart contract intermediate-level language. *ArXiv* (1 2018). https://arxiv.org/abs/1801.00687.
[116] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive* (2016). Retrieved from: https://eprint.iacr.org/2016/1159.pdf.
[117] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security*, Rainer Böhme and Tatsuaki Okamoto (Eds.). Springer, 507–527.
[118] Yonatan Sompolinsky and Aviv Zohar. 2018. PHANTOM, GHOSTDAG: Two scalable BlockDAG protocols. *IACR Cryptology ePrint Archive* (2018). Retrieved from: https://eprint.iacr.org/2018/104.pdf.
[119] N. Swamy, J. Chen, C. Fournet, P. Strub, K. Bhargavan, and J. Yang. 2011. Secure distributed programming with value-dependent types. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*. ACM, 266–278.
[120] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford. 2017. Scalable bias-resistant distributed randomness. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'17)*. 444–460.
[121] Nick Szabo. 1997. Formalizing and securing relationships on public networks. *First Mon.* 2, 9 (1997). Retrieved from: http://ojphi.org/ojs/index.php/fm/article/view/548.
[122] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*.
[123] F. Tschorsch and B. Scheuermann. 2016. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Commun. Surv. Tutor.* 18, 3 (2016), 2084–2123.
[124] Florian Tschorsch and Björn Scheuermann. 2016. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Commun. Surv. Tutor.* 18, 3 (2016), 2084–2123.
[125] Nicolas van Saberhagen. 2013. CryptoNote v 2.0. Retrieved from: https://cryptonote.org/whitepaper.pdf.
[126] Gavin Wood. 2014. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Retrieved from: http://gavwood.com/paper.pdf.
[127] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus in the lens of blockchain. *ArXiv* (July 2019). Retrieved from: https://arxiv.org/abs/1803.05069.
[128] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling blockchain via full sharding. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. 931–948.
[129] Zilliqa. 2017. The Zilliqa Technical Whitepaper. (Aug. 2017). Retrieved from: https://docs.zilliqa.com/whitepaper.pdf.

Received February 2019; revised October 2019; accepted October 2019