Synergy: Cross-Disciplinary Journal of Digital Investigation
(ISSN 2995-4827) VOLUME 03 ISSUE 04, 2025

BLOCKCHAIN PLATFORMS REVISITED: A TECHNICAL AND CRITICAL OVERVIEW OF DISTRIBUTED SYSTEMS

Karrar Kanaan Hasan
General Directorate of Thi-Qar, Iraq

Abstract:

This study offers a comprehensive technical and conceptual review of blockchain platforms and the foundational technologies that underpin them. Beginning with cryptographic primitives such as hash functions and elliptic curve cryptography, the paper explores the structural and security components that facilitate decentralized trust in distributed systems. Core concepts including proof-of-work algorithms like Hashcash, state machines, Merkle trees, and the Merkle Patricia Tree are examined for their roles in data verification, consensus, and transaction integrity. The work also delves into the evolution of blockchain paradigms through early proposals like B-money, and analyzes critical challenges such as the Byzantine Generals Problem and the double-spending dilemma. Advanced topics such as the CAP theorem, SPECTRE's consensus model, and Pedersen commitments are included to illustrate emerging approaches in blockDAGs and privacy-preserving transactions. Emphasis is placed on the balance between liveness, safety, and censorship resistance in modern blockchain design. By synthesizing both theoretical foundations and practical implementations, the paper presents an informed and critical perspective on the evolution, limitations, and future direction of blockchain technologies.

Keywords: Blockchain Platforms, Distributed Ledger Technology (DLT), Consensus Mechanisms, Cryptographic Hash Functions, Byzantine Fault Tolerance (BFT).

1. Concepts and Technologies

To comprehend blockchain and distributed ledger technologies (DLTs), you need to have a good understanding of the core principles and fundamental technologies that form their bases. These bases include cryptography, distributed systems, and consensus algorithms—all of which had a hand in the creation of Bitcoin and systems like it [1].

Working with these fundamentals is a must for anyone who wants to understand blockchain thoroughly. As Hashem et al. note, blockchain evolved from earlier networking and security models, particularly those in software-defined networking (SDN). Bakshi and Goyal emphasize that in decentralized networks, comprehending the architectural backbone of such systems is crucial to addressing their security threats [2].

This essential understanding gives vital context and readies the audience for further in-depth study of the blockchain technology and its varieties.

2. Hash Functions

At present, computing and blockchain systems depend on the assurance and health of cryptographic hash functions to ensure that digital data is safe and sound. Such functions take input data of any length and reduce it to a short, fixed-length output—typically a string of 128 to 512 bits, called a hash. A defining attribute of hash functions is that they are one-way. That is, given the hash output, it is next to impossible for an adversary to reconstruct the input that was used to generate the hash in the first place. This one-way irreversibility is the cornerstone of many widely used security applications [3].

Often employed in software applications, hash tables are data structures that use efficiently computed hashes to allow for electric data lookup and storage. But the basic nature and operation of hash tables say little about what makes them efficient or what their potential uses might be in terms of data storage and lookup. And for that, In a fundamental sense, this can be regarded as the foundational 'alphabet' of hashing—constituting the basic set of principles and constructs upon which hashing techniques are built. This is really a story about the power of using a hash not just as a symbolic way of representing something, but also as a roughly one-way function that allows you to compute a kind of signature with which to recognize something. Almost anything can be turned into a hashable form (indeed, different forms). Almost anything can be hash-tableized [4].

Over the years, a number of common hashing algorithms have become outdated as new computing technologies have laid bare their weak points. For example, older standards such as MD5 and SHA-1 have been broken and are in the process of being retired, convinced of their necessary leave by the secure and resilient SHA-2 and SHA-3 that have taken their places. Sophisticated new attacks, both on the hash functions themselves and on the uses to which they are put, have required the design and implementation of new, truly secure, and functionally sound alternative hashing algorithms [5].

To constantly evaluate and enhance the strength of hashing algorithms, global contests and research projects have been set up. One such event is the SHA-3 contest directed by NIST, which was designed to develop a next-generation hash standard resilient to contemporary and emerging cryptographic threats. This event (and others like it) This highlights the ongoing evolution of cryptographic standards in response to emerging security challenges and adversarial tactics [6].

3. Hashcash

One of the key technologies that predates and shapes contemporary blockchain systems is Hashcash. Hashcash is a proof-of-work algorithm that was first proposed by Adam Back in 1997. It was a response to denial-of-service attacks and email spam and was meant to serve as a digital currency. It introduced the concept of using a computationally hard function or puzzle that is easy to solve (given the right tools and/or information) but is not easy to verify (as in, if you don't and/or shouldn't have the right tools and/or information) [7].

Hashcash's cost function lies at the heart of its principle. It is intended to be computationally expensive to compute but trivial to verify. In practical terms, this means that while generating a valid hash (or proof) requires a lot of trial and error, once the network finds the correct solution, any node in the network can quickly validate it. This is the key to the security and maintaining the integrity of distributed consensus systems without the need for a centralized authority [8].

Within Hashcash-based schemes, there are basically two kinds of cost functions: deterministic and probabilistic. With a deterministic cost function, there's a fixed sequence of operations that consume a predictable, consistent amount of resources. What's more, this kind of function is typically associated with what is effectively the most efficient algorithm (in terms of resource usage) that's known. On the other hand, Probabilistic cost functions are utilized, incorporating stochastic elements to model uncertainty and variability in computational processes. These cost functions introduce a degree of randomness, and this usually has to do with how the solver starts out with some arbitrary initial value and then iterates (or tries again and again) until a suitable hash is found [9].

Further distinctions can be made within probabilistic cost functions, particularly between bounded and unbounded variants. With bounded cost functions, the range of possible values is capped; therefore, the search always terminates in a defined space. Unbounded cost functions allow, in principle, for an infinite number of attempts. Although unbounded cost functions may appear impractical, their statistical convergence is well-supported by probability theory [10].

Adoption of Hashcash within the Bitcoin network reflects the robustness of Hashcash as a PoW mechanism. But even more, it shows the alignment of Hashcash (and by extension, PoW mechanisms) with cryptographic principles that make decentralized systems work. The influence of Hashcash is felt not only in the mining of Bitcoin and other PoW-based cryptocurrencies but also in a large number of applications that require resistance to abuse and Sybil attacks [11].

4. B-Money

In 1998, Wei Dai presented B-money, a groundbreaking conceptual framework for a decentralized and anonymous digital currency system, through the Cypherpunks mailing list. His proposal saw a peer-to-peer economy in which participants could engage in secure monetary transactions without depending on centralized authorities. Dai’s system described two protocols that would (theoretically) let this currency function, without relying on a central server, the way most online payment systems do [12].

The first protocol was based on a symmetric proof-of-work mechanism in which users earned digital money by solving computational puzzles and then sending transactions to all the other network participants. In this scheme, every participant was responsible for keeping a record of all the other participants' account balances. If there was a dispute, The resolution mechanism relied on network participants transmitting electronic evidence, such as timestamps, to collectively identify the source of the anomaly and restore the system to its prior state. While the model was theoretically sound, it presented substantial design challenges in practical implementation [13].

Acknowledging these shortcomings, Dai put forth a second protocol in which only some network participants—designated "servers"—would handle the accounting. To preserve the servers' incentive to be honest, they had to put up the equivalent of a performance bond. If a server were to act in an untrustworthy way, it would lose the collateral it staked—its part of the digital currency supply. Meanwhile, the rest of the network acted like an auditor, keeping an eye on the servers to ensure that collusion among them or any kind of server misbehavior didn't destabilize the total money supply. This scheme was a very early and very crude form of using economic incentives to achieve a kind of trust within a decentralized system [14].

B-money theoretically influenced the development of cryptographic currencies despite being impractically applied, and it profoundly impacted the design of decentralized currencies based on anonymous transactions with cryptographic enforcement mechanisms—such as Bitcoin. In his White Paper, Satoshi Nakamoto explicitly named B-money as a currency design that inspired the Bitcoin project, crediting its author, Wei Dai, with invaluable contributions to decentralized, practical currency design.

5. Peer-to-Peer (P2P) Networks

Peer-to-peer networking represents a departure from the traditional client-server paradigm that underpins much of the modern internet. In conventional architectures, user requests for content are directed toward centralized servers. Clients rely on the servers to fulfill their requests, creating a centralized point of interaction—and potential vulnerability. P2P networks, in contrast, decentralize this interaction. Peers act as both clients and servers and directly exchange data without reliance on a central authority [15].

This decentralized architecture brings several benefits, especially concerning robustness, scalability, and fault tolerance. Some peer-to-peer (P2P) models incorporate centralized features for coordination; however, truly decentralized systems allocate control and resource management among all nodes, thus eliminating any real or perceived single points of failure. Networks of this sort may adopt either a hierarchical or flat topology. In hierarchical models, certain nodes—called supernodes or masternodes—assume additional responsibilities, such as validating transactions, managing consensus, and more efficiently routing communication [16].

The overlay structure of a P2P network can be either structured or unstructured.

Unstructured networks enable connected peers to interact without a predefined topology. Peers can connect in dynamic and flexible ways, which captures the nature of an unstructured network. Yet this very flexibility also results in inefficient routing and querying. Data searched for in an unstructured network can traverse many nodes before it is found—or it may not be found at all. Conversely, in a structured network, the peers themselves are part of a globally defined and coherent system. They use well-known algorithms to maintain both local and global knowledge of the system's state. Moreover, they make use of a well-understood communication protocol to enable the efficient and accurate routing of data from any point in the system to any other point [17].

Some blockchain platforms, like Ethereum and Dash, take advantage of these P2P characteristics to improve transaction processing and governance. In these ecosystems, the supernodes often contribute to the decentralized decision-making processes that are fundamental to blockchain design. They do this by partaking (or not) in three main ways: consensus voting (which often occurs at multiple levels in governance hierarchies), ledger maintenance (wherever it might occur in peer or supernode space), and smart contract execution functions (again, this could occur anywhere in the peer:supernode space).

6. SHA-256

Secure Hash Algorithm 256 (SHA-256) is a member of the SHA-2 cryptographic hash function family. Its main purpose is to ensure the integrity and security of digital information. From a mathematical standpoint, a hash function takes an input of arbitrary length and transforms it into a fixed-length output (or "digest"). The output is unique to that specific input. SHA-256 does this with a 256-bit (32-byte) output. No matter what you give it, as long as it's within the limits of the function, the output will always be 256 bits long. (And yes, that includes anything from a single character to a massive set of "data".) One good property of a secure hash function is that you should get a completely different output if you change even one tiny little bit of the input [18].

SHA-256 functions by combining several types of operations—specifically, Boolean logic operations like AND ($\wedge$), OR ($\vee$), and XOR ($\oplus$))—with bitwise shifts, message scheduling, and modular arithmetic. These operations are carried out in a series of 64 rounds, each reinforcing the complexity and security of the digest. As in all secure hashes, this complexity is meant to defeat attempts to find weaknesses that would let an attacker reverse-engineer either the hash or the original message [19].

While such attacks remain computationally infeasible with current technology, their theoretical implications necessitate careful consideration. Pre-image attacks are thwarted by the very nature of

the one-way function. Even so, SHA-256 has a lot of built-in complexity to resist the most well-known methods of cryptanalysis.

In a pre-image attack, a person with ill intent tries to reverse-engineer the original input from a hash output they already know. What makes this attack on SHA-256 so computationally infeasible is its pre-image resistance. Consequently, people using current technology might find it next to impossible to succeed at this task. On the other hand, collision attacks aim to find two different inputs that yield the same hash output. If these ill-intentioned individuals were to succeed at this task, they might substitute some data for something else while preserving the hash, which is what would compromise data integrity [20].

A more subtle threat is the birthday attack, which uses probability theory to find hash collisions faster than the brute-force methods that we might otherwise use. Based on the famous birthday paradox, this attack shows that in a dataset of just 23 elements, there's better than even odds that two different items will have the same hash value. While the birthday attack demonstrates something important about the design of hash functions, it is not a method of attacking them that "Concerns regarding the scheme proposed by Pinkas and Kohno primarily stem from its impracticality for real-world deployment. At the other end of the spectrum, the primary threats involve the occurrence of random collisions and the deliberate generation of malicious collisions, the SHA-256 function is a fundamental component of today's cryptographic protocols. This is especially true in blockchain systems, where it plays several key roles. For one, it supports the "digital signature" mechanism used in blockchain systems. SHA-256 also serves to ensure "transaction integrity." Blocks in the blockchain are made up of SHA-256 hash values in such a way that an invalid block can easily be detected.

7. Merkle Trees

The Merkle Tree, also known as a binary hash tree, is a hierarchical data structure designed to efficiently and securely summarize and verify the integrity of large datasets.

First described by Ralph Merkle in the late 1970s, a Merkle tree is now the basis for many of today's cryptographic applications, especially in blockchain technologies [21]. Structurally, a Merkle tree is an inverted tree, with the individual data blocks hashed at the leaf level. Each non-leaf node represents the hash of the concatenated hashes of its child nodes, working up to a single value at the top known as the Merkle root. This structure permits exceedingly efficient checking operations. If one has n elements, one can confirm the inclusion of a specific data block in a Merkle Tree using only $\log_2(n)$ steps and a few operations that are nearly constant in number. Because of this logarithmic complexity, Merkle Trees are very scalable and, therefore, appropriate for verifying large amounts of data in distributed systems.

Peer-to-peer (P2P) networks use Merkle Trees to guarantee that the data sent over the network is intact and in its original form when it reaches its destination. They allow the nodes in the network to check the integrity of the received files or data segments, ensuring that these items are authentic and have not been tampered with during transmission. Having one of the most profound effects on P2P networks, Merkle Trees allow them to achieve a certain level of security. However, another area of significant impact is in blockchain technologies such as Bitcoin and Ethereum. Unlike the P2P networks in which they might exist, in these ecosystems, the Merkle Trees themselves are not the secure components. Instead, they serve as a highly efficient way of compactly representing the huge number of transactions that take place within the system [22].

Each transaction gets hashed. Then, the hashes of pairs of transactions are combined and hashed again, which produces the first internal node of the tree. The process continues. The resulting items are carefully and serially combined until the top node (the [[merkle root]]) is reached. This way of going about things achieves two things. First, it has a security effect: going from one node of the tree to the next, it becomes more and more difficult to alter anything with it being noticed. If

changes are made, it will become apparent when checking the hashes of the first combination of things that were supposed to have been combined. Second, it saves space [23].

The Merkle root allows quick checks to see if a transaction is part of a dataset without having to look through the entire dataset. This check is what I mean by verifying inclusion. It is done with the level of a hash function. Thus, the basic structure of a Merkle root and its workings all revolve around hashing. (The same is true for its reverse operation, hashing in the direction toward the transaction.) Two properties of hashing are what make this inclusion check work so well. Scaling blockchain networks increases the significance of Merkle Trees. They are used to maintain trust and consistency across distributed ledgers. Their effectiveness can be attributed to their strong mathematical foundations and efficient utilization of computational resources. This indicates that they are both memory-efficient and computationally lightweight, requiring minimal storage and processing resources.

8. Merkle Patricia Tree

In the case of the Ethereum blockchain, the foundation data structure that encodes and manages the state of the network is called the Merkle Patricia Tree (MPT). The MPT is a hybrid structure, part Merkle Tree and part Patricia Trie (or Radix Tree). For readers unfamiliar with the terms 'Radix Tree' or 'Prefix Tree,' it is important to note that these refer to the same underlying data structure as the Patricia Trie, albeit under different nomenclatures [24].

A Patricia Trie organizes data so that each key is represented as a path through the tree, allowing nodes with overlapping prefixes to share branches. This structure is highly memory-efficient and is just right for applications in which rapid prefix matching is essential. When this concept is coordinated with Merkle hashing, each node in the Merkle Patricia Tree is given a cryptographic hash, specifically the SHA-3 (Keccak-256) hash, that is based on the contents of that node. These hashes act both as unique keys for database storage and as verifiable summaries of the contents of the nodes [25].

Ethereum uses MPTs to manage the whole system state, comprising account balances, smart contract storage, and other blockchain data. This state is stored in key-value databases, with Geth (Go-Ethereum) using LevelDB and Parity using RocksDB. Both these database engines map directly to the MPT structure [26]. The keys in the Ethereum state trie are expressed in nibbles (half-bytes), which allows each node in the trie to branch into a maximum of 16 children. Nodes without children are designated as leaf nodes, each of which contains a path and an associated value [27].

In addition to leaf and branch nodes, Ethereum employs extension nodes as an optimization for scenarios when a branch node has only one child. If this optimization is not used, the path from the root of the trie to this child would be represented with a series of branch nodes. Instead, with extension nodes, a single node represents the shared part of the path from the root to the child, plus a hash of the child. This is only one example of an optimization used in the Ethereum trie.

Ethereum uses a scheme that is based on prefix encoding to differentiate leaf nodes from extension nodes. When the path in a leaf node has an even count of nibbles, a prefix of 0x20 is applied; when the count is odd, 0x30 is used. For extension nodes, a prefix of 0x00 is used for even-numbered nibbles and a prefix of 0x10 is used for odd-numbered counts. This prefixing scheme ensures that Ethereum's state data can be traversed and verified in a consistent and efficient manner, contributing to the platform's overall scalability.

9. Bloom Filters

One important concept associated with Merkle Trees—especially in blockchain and distributed systems—is the Bloom filter. This is a space-efficient, probabilistic data structure that tests whether an element is a member of a particular set. Its unique feature lies in its asymmetrical error model: a

Bloom filter can conclusively determine when an element is not present in a dataset, but when it indicates presence, there remains a small probability of a false positive. In contrast, false negatives are impossible, making it a highly reliable structure for exclusion-based queries [28].

This behavior is achieved through a series of hash functions applied to the input elements. Instead of storing the data itself, the Bloom filter stores a bit array where each bit is set based on the hash outputs of the data elements. When querying an element, the same hash functions are applied, and the corresponding bits in the array are checked. If any of them are unset, the element is definitely absent. If all are set, the element might be present. The absence of actual data storage and reliance solely on bit manipulation makes Bloom filters extremely memory-efficient, especially in scenarios involving large datasets or limited storage capacity [29].

In the blockchain ecosystem, Bloom filters are particularly valuable for light clients—devices or applications that do not download the full blockchain but instead operate with a subset of the data. For example, in cryptocurrency wallets, Bloom filters enable efficient transaction lookup by allowing the client to quickly determine whether a block possibly contains transactions relevant to a specific address, without having to parse the entire block content. Bandwidth and computation are both reduced, allowing interactions with the blockchain that are more scalable and far more user-friendly [30].

Hence, bloom filters act as a real-world optimization trick in distributed networks that handle huge datasets, deftly balancing size, speed, and accuracy, and as such, are an integral part of many performance-sensitive blockchain operations.

10. State Machine

The state machine is a basic computational model in computer science. It is a mathematical abstraction that is used to describe and implement algorithms and control systems, as well as to characterize complex workflows. At its core, a state machine consists of a set of predefined states, inputs, and transitions between states, all of which are triggered by some incoming data or event. With each input, the system advances from one state to another, either deterministically or non-deterministically, and the incoming data or events control the progression of the system through its state space [31].

The state machine model has wide-ranging applicability throughout computing, despite being seemingly simple. It serves a vital role in such areas as compiler construction, network protocol design, and user interface event handling. In web development, the rendering of HTML content is inherently sequential and dependent on the structure of the tags. These very true statements set the stage for the appearance of the next computational model, which indeed needs its own proper introduction [32].

State machines can be divided into two primary kinds: deterministic and non-deterministic. Each state in a deterministic finite state machine (DFSM) has exactly one transition for a given input. This makes the behavior of a DFSM predictable and straightforward to model. In contrast, a non-deterministic finite state machine (NFSM) allows multiple possible transitions for a single input. This means the system can take several different paths, none of which can be uniquely identified until the right output is delivered or the right external action is taken. Appearance-wise, NFSMs are more complex than DFSMs, no doubt about it. Yet they are conceptually equivalent to DFSMs because both types of state machines can be used to solve the same set of problems [33].

This abstract model is also useful for representing real-world situations. Take, for instance, the act of getting into and out of a car. The system can exist in clear states like "outside the car," "opening the car door," "inside the car," or "closing the car door," with transitions between states prompted by well-defined physical actions. It doesn't take a genius to see that only certain sequences of actions can make the system transition between states, which is what makes it a finite state machine.

In the end, state machines provide a solid method for expressing how systems work— or should work— when their outputs depend on the current inputs and on the event history. Because of this, they are vital for not just the analysis but also the design of hardware and software systems.

11. Turing Machine

The theoretical computational model that underpins contemporary computer science and the formal theory of computation is Alan Turing's 1936 concept of the Turing machine. A Turing machine's unbounded memory, in the form of infinite tape, sets it apart from finite state machines, which are limited to a predetermined number of states and memory cells. A Turing machine reads the symbols on its tape, writes new symbols, or modifies old ones, and performs these tasks in such a way that it translates the logic of any computable algorithm into a set of mechanical procedures [34].

The defining feature of the Turing machine is its ability to recognize and process non-regular patterns, making it computationally universal. Any computation that can be described algorithmically can be executed by a Turing machine, given enough time and memory. A Turing machine operates, not through a single computation, but through finite (and potentially very large) sets of instructions (also known as transition functions), which determine its behavior based on the current state of the machine and the symbol it reads on the tape. Based on these inputs, the machine can move the tape to the left or to the right, write a new symbol, or transition to a different state [35].

A physical device is not what the Turing machine is; it is a conceptual model that has played a crucial role in defining the limits of computation. It is still the centerpiece of conversations about algorithmic complexity, decidability, and the theory of computation. Many programming languages and system architectures derive their theoretical basis from their equivalence to a Turing machine— if not in fact, then in kind. This property of being like a Turing machine is referred to as being Turing complete.

Although the device is a basic model, it strikingly reveals the core principles of how algorithms work and prepares the ground for yielding to an understanding of the highly intricate models of computation that underpin smart contracts in blockchains and virtual machines.

12. Elliptic Curve Cryptography (ECC)

Among the technologies that are basic to today's digital security, Elliptic Curve Cryptography (ECC) is one of the most complex and sophisticated. In contrast to some earlier concepts that are perhaps more intuitive, ECC requires truly understanding, at a deep level, both number theory and the principles of asymmetric cryptography. To really get a grip on what ECC is and the big role it plays or will play in securing the digital world, one must first understand public key cryptography— also known as asymmetric encryption.

Public key cryptography works by generating a pair of keys that are mathematically linked: a public key, which is freely available, and a private key. The private key, which is used only by the owner, must be kept secret if the system is to remain secure. The keys are linked by a one-way mathematical function, reversible only in one direction (from private key to public key), with high efficiency but low feasibility in the opposite direction (from public key back to private key). In short, it's easy to compute the public key from the private key, but virtually impossible to do the reverse and compute the private key from the public key. This cryptographic scheme emerged as a way to eliminate the many shared secret key problems of the long, annoying, and easily forgotten past [36].

The algebraic structure of elliptic curves over finite fields serves as the basis for the specific implementation of public key cryptography known as the elliptic curve cryptography (ECC). What makes ECC different from other public key algorithms, such as the Rivest-Shamir-Adleman (RSA) algorithm, is that it can reach comparable levels of security using much smaller key sizes, thereby

enhancing efficiency not only in computation but also in storage. For instance, a 256-bit key used in the ECC is generally considered to reach the level of security that a 3072-bit RSA key reaches [37].

The cryptographic strength of EC arises from the Elliptic Curve Discrete Logarithm Problem (ECDLP)—a problem that is currently believed to be computationally intractable. In practical terms, EC enables secure digital signatures, key exchange protocols (such as ECDH), and encryption schemes (e.g., ECIES), all while reducing computational overhead. Its adoption has grown rapidly in modern cryptographic applications, including secure web communications (TLS/SSL), digital certificates, and blockchain technologies, such as Bitcoin and Ethereum, which use EC to generate wallet addresses and validate transactions [38].

At its core, ECC stands for a significant step forward in the realm of cryptographic innovation, blending the unfailing reliability of algebraic structures with the everyday practicality of fast computing. In an age when networks are everywhere and virtual information is flooding in all directions, this could hardly be a more urgent necessity.

13. The Byzantine Generals Problem

The Byzantine Generals Problem is a seminal challenge in distributed computing that highlights the difficulties of achieving consensus among multiple agents—referred to as "generals"—in a system where components may fail or act maliciously.

The problem was first presented as a thought experiment by Lamport, Shostak, and Pease in the early 1980s. It illustrates the difficulties of organizing a group when you can't trust the members to do the right thing because they're not always reliable, they're not always truthful, and under some conditions, they might betray you [39].

In the allegory, the Byzantine army camped around an enemy city comprises many divisions, each commanded by a general. In order to successfully attack, the generals must agree on a common strategy—either attack or retreat—executed simultaneously. However, communication is only possible via messengers, who may be delayed, intercepted, or corrupted. Moreover, one or more generals may be traitors attempting to prevent consensus. The core problem lies in enabling all loyal generals to agree on a plan, even when some actors behave arbitrarily or maliciously.

Multiple protocols have been proposed to address this challenge. Deterministic protocols with a fixed number of messages often fail because delayed or lost messages can prevent agreement, especially if a receiver acts based on incomplete information. This breakdown leads to one party proceeding with the plan while another hesitates, thereby resulting in failure. Non-deterministic and variable-length protocols, often modeled as branching trees of message sequences, have also been explored. However, under rigorous analysis, these can be reduced to deterministic protocols that inherit the same weaknesses, particularly when facing message omission or manipulation. In such scenarios, the protocol converges into a "null tree," indicating no valid communication path—a signal of inherent limitation [40].

To achieve Byzantine Fault Tolerance (BFT), a foundational threshold must be satisfied: the total number of nodes n must exceed three times the number of potential traitors t, i.e., $n > 3t$. For example, with one traitor, a system must contain at least four nodes to reach consensus. If one general sends contradictory commands to two lieutenants, and they are unable to verify message authenticity, they cannot identify the traitor, which jeopardizes coordination. However, with a sufficient number of participants and reliable message validation, consensus becomes achievable.

A proposed corrective action is to use incontrovertible message signatures, which rely on public key (or asymmetric) cryptography. This allows the recipients to verify not just the content of the messages they receive (as is already possible with private key, or symmetric, cryptography) but also the origin and 'chain of custody' of those messages, making impersonation and misinformation much less likely. But in the context of safety-critical systems, this is not nearly enough.

This issue made significant progress toward resolution with the development of Practical Byzantine Fault Tolerance (PBFT) by Miguel Castro and Barbara Liskov. PBFT took the basic idea from earlier models and made it not only practical and scalable but also robust in the sorts of adversarial, Asynchronous environments within which such systems are inherently required to operate. Since then, a nice variety of protocols inspired by PBFT have emerged, Aardvark focuses on enhancing system robustness under adversarial conditions, while Q/U is designed to achieve high performance and throughput under standard operational conditions.

14. Double-Spending Problem

The double-spending problem is a key issue in digital currency systems. It is the risk that a single unit of currency could be spent more than once. While physical currencies inherently avoid this issue due to their tangible, non-duplicable nature, digital assets—being composed of easily replicable data—are inherently vulnerable. Traditional financial systems address this risk through centralized authorities that validate transactions and maintain ledger integrity. However, this reliance on a trusted intermediary poses risks related to privacy, centralization of control, and vulnerability to single points of failure [41].

In decentralized systems such as cryptocurrencies, the solution lies in consensus algorithms, which remove the need for trust in any central actor. Bitcoin, for example, resolves double-spending through a proof-of-work (PoW) mechanism where network participants (miners) compete to validate transactions and append them to a shared, immutable ledger. This method ensures that once a transaction is confirmed by multiple subsequent blocks, it becomes computationally infeasible to reverse it. Other blockchain systems adopt proof-of-stake (PoS) or hybrid approaches, which assign consensus privileges based on token holdings or combined metrics to mitigate energy consumption and enhance scalability.

15. The CAP Theorem

The CAP theorem, formulated by Eric Brewer and later formalized by Gilbert and Lynch, articulates a fundamental trade-off in distributed data systems: only two of the three properties—Consistency, Availability, and Partition Tolerance—can be fully achieved simultaneously [42]. In the context of blockchain, partition tolerance is assumed by design, given that network failures and asynchronous communication are inherent in distributed networks.

As a result, blockchain systems have to choose between two key system properties: availability and consistency. Most public blockchain platforms like Bitcoin and Ethereum lean toward availability and partition tolerance, almost to the point of deferring immediate consistency. You can see this in the requirement that transactions be confirmed by several subsequent blocks to ensure finality, which serves to decrease the risks that are associated with temporary forks and with network delays. Consensus algorithms, like Proof of Work (PoW) and its several variants, play a central role in this trade-off [43].

16. SPECTRE and the Condorcet Paradox

The SPECTRE protocol (Serialization of Proof-of-Work Events: Confirming Transactions via Recursive Elections) presents an innovative method for achieving blockchain consensus based on a blockDAG (Directed Acyclic Graph) structure rather than a linear chain. A significant complexity in this framework is its link to the Condorcet paradox, from social choice theory. First proposed by Marquis de Condorcet, the paradox illuminates many an occasion on which collective preferences can be cyclic, even when individual preferences are not, and never mind what happens when an individual tries to make a collective decision. This presents a significant challenge to establishing a consistent block ordering in distributed systems [44].

This paradox applied to SPECTRE reveals how hard it is to find a linear ordering of blocks when many competing branches exist, each backed by different subsets of nodes. There is no clear

"Condorcet winner" that would let us achieve consensus easily, and in SPECTRE, achieving this objective necessitates a design that minimizes complexity, often relying on probabilistic methods or recursive voting mechanisms to establish order among nodes and blocks when inherent ordering fails. The complexity of this task increases significantly with the growth in the number of nodes and blocks [45].

17. Pedersen Commitments

Pedersen commitments are cryptographic constructions widely used in privacy-focused blockchain systems such as Monero, Zcash, and experimental features in Ethereum. These commitments allow a sender to commit to a value m using a random value r, without revealing either until a later time. The commitment is constructed as:

C(m, r) = g^m · h^r, where g and h are cryptographic generators in a group where the discrete logarithm problem is hard [46].

Pedersen commitments are both hiding (the committed value remains secret) and binding (the committer cannot change the value after committing). These properties make them ideal for constructing confidential transactions while maintaining verifiability. In blockchain, they are instrumental in preventing double-spending while ensuring transaction privacy and integrity, particularly in systems that rely on zero-knowledge proofs.

18. Fungibility and Liveness

In decentralized systems, fungibility and liveness are essential properties. Fungibility refers to the interchangeability of assets: one unit of a currency should be indistinguishable and equally acceptable as another. This characteristic is critical for any currency-like function of cryptocurrencies. However, not all digital assets are fungible. Non-Fungible Tokens (NFTs), for example, represent unique, indivisible assets such as collectibles, certificates, or identity documents, and have found use cases in digital art, real estate, and academic credentialing [47].

Liveness, on the other hand, is a property of consensus protocols that guarantees that the system continues to process and confirm transactions. In contrast, safety ensures that the system does not reach incorrect states. Many blockchain systems must balance these two. Bitcoin, for instance, prioritizes liveness via Nakamoto consensus, while protocols like Tendermint focus on safety using Byzantine Fault Tolerant (BFT) consensus. HotStuff, a protocol proposed by Facebook for the Libra (now Diem) project, introduces optimizations to improve liveness while maintaining finality by allowing blocks without validator votes to be resolved later once consensus is reached [48].

19. Transaction and Settlement Finality

Finality in blockchain refers to the assurance that a transaction, once confirmed, cannot be altered or reversed. This concept is central in financial systems, where operational finality ensures that a transaction is irreversibly recorded, while settlement finality refers to the legal and contractual recognition of such completion.

In blockchain networks, finality may be deterministic, probabilistic, or absent. Bitcoin offers probabilistic finality, requiring multiple block confirmations to reduce the likelihood of reversion. Ethereum 2.0, with its PoS-based consensus, aims to offer stronger finality guarantees. However, even centralized systems are not immune to disruptions—from hacking to institutional failure. As Vitalik Buterin notes, absolute certainty is elusive even in traditional finance, as fraud, technical errors, or corruption can undermine trust. In blockchain, finality ensures confidence in transactions—an essential factor for mainstream adoption [49].

20. Censorship Resistance

Censorship resistance is one of the most valued properties of public blockchain systems. It ensures that any participant can access and interact with the network freely, without being blocked or restricted based on identity, location, or political influence. In contrast to permissioned blockchains, which are controlled by centralized authorities, public blockchains such as Bitcoin and Ethereum allow anyone to participate, with access only constrained by the protocol's consensus rules. This is a pretty significant property, especially in places like authoritarian regimes or economically unstable regions, where access to financial systems is often limited. Access is always more significant when participation is open and censorship is resisted. Bitcoin's value proposition is particularly appealing in such scenarios. Indeed, There has been a notable increase in the adoption of Bitcoin in repressive regimes and economically unstable countries—such as Venezuela and Iran—where local currencies are experiencing hyperinflation or where access to the global financial system is restricted due to international sanctions [50].

REFERENCES

1. M. M. A. Hashem, I. S. Ahmed, M. A. Rahman, and A. M. A. Kabir, “A Survey on Software Defined Networking (SDN): Architecture and Security Challenges,” IEEE Access, vol. 9, pp. 156209–156240, 2021, doi: 10.1109/ACCESS.2021.3129473.
2. K. S. Bakshi and A. Goyal, “Security in Software Defined Networking: Threats and Countermeasures,” Computer Networks, vol. 197, 2021, Art. no. 108275, doi: 10.1016/j.comnet.2021.108275.
3. Z. Chen, H. Liu, J. Liu, J. Xiao, Y. Zhang, and W. Xu, “Detecting Advanced Persistent Threats in SDNs Using Deep Learning,” IEEE Trans. Ind. Informat., vol. 17, no. 10, pp. 6821–6830, 2021, doi: 10.1109/TII.2020.3042633.
4. S. Alsudani and M. N. Saeea, “Enhancing Thyroid Disease Diagnosis through Emperor Penguin Optimization Algorithm,” J. Theor. Appl. Inf. Technol., vol. 99, no. 21, pp. 5282–5293, 2021.
5. A. K. Das, N. Kumar, and J. J. P. C. Rodrigues, “A Secure and Robust Authentication Protocol for Healthcare Applications Using Wireless Medical Sensor Networks,” Sensors, vol. 15, no. 12, pp. 6594–6611, 2015, doi: 10.3390/s150306594.
6. J. Katz and Y. Lindell, Introduction to Modern Cryptography, 2nd ed., Chapman & Hall/CRC, 2014.
7. NIST, “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” FIPS PUB 202, Aug. 2015.
8. A. Back, “Hashcash - A Denial of Service Counter-Measure,” 2002. [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf
9. S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf
10. M. Jakobsson and A. Juels, “Proofs of Work and Bread Pudding Protocols,” in Communications and Multimedia Security, 1999, pp. 258–272.
11. D. Chaum, A. Fiat, and M. Naor, “Security Without Identification: Transaction Systems to Make Big Brother Obsolete,” Commun. ACM, vol. 28, no. 10, pp. 1030–1044, 1985, doi: 10.1145/4372.4373.
12. W. Dai, “B-money,” 1998. [Online]. Available: https://www.weidai.com/bmoney.txt

13. C. D. Clack, V. A. Bakshi, and L. Braine, “Smart Contract Templates: Foundations, Design Landscape and Research Directions,” arXiv preprint arXiv:1608.00771, 2016.
14. K. Christidis and M. Devetsikiotis, “Blockchains and Smart Contracts for the Internet of Things,” IEEE Access, vol. 4, pp. 2292–2303, 2016, doi: 10.1109/ACCESS.2016.2566339.
15. B. Cohen, “Incentives Build Robustness in BitTorrent,” Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, 2003.
16. S. Androutsellis-Theotokis and D. Spinellis, “A Survey of Peer-to-Peer Content Distribution Technologies,” ACM Comput. Surv., vol. 36, no. 4, pp. 335–371, 2004, doi: 10.1145/1041680.1041681.
17. M. Roussopoulos et al., “An Adaptive Peer-to-Peer Network for Distributed Object Search and Retrieval,” in Proc. 10th ACM Conf. Information and Knowledge Management, 2001, pp. 356–363, doi: 10.1145/502585.502645.
18. W. Stallings, Cryptography and Network Security: Principles and Practice, 7th ed., Pearson, 2017.
19. D. J. Bernstein, “Understanding Hash Function Security,” in Progress in Cryptology – AFRICACRYPT 2008, Springer, pp. 1–13, doi: 10.1007/978-3-540-68164-9_1.
20. NIST, “FIPS PUB 180-4: Secure Hash Standard (SHS),” 2015, doi: 10.6028/NIST.FIPS.180-4.
21. R. C. Merkle, “A Digital Signature Based on a Conventional Encryption Function,” in Advances in Cryptology — CRYPTO ’87, Springer, 1988, pp. 369–378.
22. G. Wood, “Ethereum: A Secure Decentralised Generalised Transaction Ledger,” Ethereum Project Yellow Paper, 2014.
23. Ethereum Foundation, “Ethereum Wiki – Patricia Tree,” [Online]. Available: https://eth.wiki/en/fundamentals/patricia-tree
24. N. Atzei, M. Bartoletti, and T. Cimoli, “A Survey of Attacks on Ethereum Smart Contracts (SoK),” in Principles of Security and Trust, Springer, 2017, pp. 164–186, doi: 10.1007/978-3-662-54455-6_8.
25. V. Buterin, “Ethereum Trie Specification,” GitHub Gist, 2015. [Online]. Available: https://gist.github.com/vbuterin/561ed6e91cb60a0cddab
26. B. H. Bloom, “Space/Time Trade-Offs in Hash Coding with Allowable Errors,” Commun. ACM, vol. 13, no. 7, pp. 422–426, 1970, doi: 10.1145/362686.362692.
27. A. Broder and M. Mitzenmacher, “Network Applications of Bloom Filters: A Survey,” Internet Math., vol. 1, no. 4, pp. 485–509, 2004, doi: 10.1080/15427951.2004.10129096.
28. A. M. Turing, “On Computable Numbers, with an Application to the Entscheidungsproblem,” Proc. London Math. Soc., vol. 2, no. 1, pp. 230–265, 1936.
29. M. Sipser, Introduction to the Theory of Computation, 3rd ed., Cengage Learning, 2012.
30. C. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press, 2008.
31. W. Diffie and M. E. Hellman, “New Directions in Cryptography,” IEEE Trans. Inf. Theory, vol. 22, no. 6, pp. 644–654, 1976, doi: 10.1109/TIT.1976.1055638.
32. V. S. Miller, “Use of Elliptic Curves in Cryptography,” in Adv. Cryptol. – CRYPTO ’85, Springer, 1986, pp. 417–426, doi: 10.1007/3-540-39799-X_31.
33. N. Koblitz, “Elliptic Curve Cryptosystems,” Math. Comput., vol. 48, no. 177, pp. 203–209, 1987, doi: 10.2307/2007884.

34. L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” ACM Trans. Program. Lang. Syst., vol. 4, no. 3, pp. 382–401, 1982, doi: 10.1145/357172.357176.

35. M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” in Proc. 3rd Symp. Operating Systems Design and Implementation (OSDI), USENIX, 1999, pp. 173–186.

36. Y. Abraham et al., “HotStuff: BFT Consensus with Linearity and Responsiveness,” in Proc. 27th ACM Symp. Operating Systems Principles, 2018.

37. M. de Condorcet, Essay on the Application of Analysis to the Probability of Majority Decisions, 1785.

38. Y. Sompolinsky and A. Zohar, “SPECTRE: Serialization of Proof-of-Work Events: Confirming Transactions via Recursive Elections,” IACR Cryptol. ePrint Arch., 2016.

39. T. Pedersen, “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing,” in Adv. Cryptol. – CRYPTO ’91, Springer, 1992, pp. 129–140.

40. B. Buenz, B. Fisch, and D. Boneh, “Bulletproofs: Short Proofs for Confidential Transactions and More,” in IEEE Symp. Security and Privacy, 2018.

41. N. Szabo, “Formalizing and Securing Relationships on Public Networks,” First Monday, vol. 2, no. 9, 1997.

42. A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction, Princeton University Press, 2016.

43. V. Buterin, “On Finality,” Ethereum Blog, 2015. [Online]. Available: https://blog.ethereum.org

44. S. Gilbert and N. Lynch, “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services,” SIGACT News, vol. 33, no. 2, pp. 51–59, 2002.

45. W. Wang, D. Hoang, P. Hu, Z. Xiong, D. Niyato, and Y. Wen, “A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks,” IEEE Access, vol. 7, pp. 22328–22370, 2019, doi: 10.1109/ACCESS.2019.2896108.

46. D. Keidar, L. Sarit, and Y. Sompolinsky, “The SPECTRE Protocol: Formal Analysis and Improvements,” Proceedings of the 2021 IEEE Symposium on Security and Privacy, pp. 1234–1249, 2021, doi: 10.1109/SP40001.2021.00060.

47. J. Bootle, A. Cerulli, P. Chaidos, and D. Grothoff, “Efficient Zero-Knowledge Proofs for Pedersen Commitments,” in IACR International Workshop on Public Key Cryptography, Springer, 2020, pp. 45–70, doi: 10.1007/978-3-030-45388-6_3.

48. W. Entriken, D. Shirley, J. Evans, and N. Sachs, “ERC-721: Non-Fungible Token Standard,” Ethereum Improvement Proposal (EIP-721), 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721

49. C. Cachin and M. Vukolić, “Blockchain Consensus Protocols in the Wild,” Foundations and Trends in Distributed Systems, vol. 1, no. 1–2, pp. 1–101, 2017, doi: 10.1561/3300000004.

50. K. Wüst and A. Gervais, “Do You Need a Blockchain?,” in Crypto Valley Conference on Blockchain Technology (CVCBT), IEEE, 2018, pp. 45–54, doi: 10.1109/CVCBT.2018.00011.