BitFury

On Blockchain Auditability

White Paper

Facilitating auditability and accountability are major benefits of using blockchain technology. In this white paper, we examine the notions of auditability, accountability and non-repudiation in blockchains and focus on the aspects of blockchain technology that allow for these properties. We focus on blockchain receipts, blockchain anchoring, and proof of work as the means to provide non-repudiation and accountability in permissioned blockchains that utilize Byzantine consensus. A technical description to implement Bitcoin Blockchain anchoring for permissioned blockchains without reducing security properties is provided.

Version 1.0 (Nov 14, 2016)

Blockchain technology has been posited as a solution for various applications in the financial sphere, such as payment systems [1], securities trading [2], post-trade settlement [3], regulation [4], consulting and auditing [5, 6], etc. Furthermore, blockchains have been proposed for non-financial use cases including public registries [7], copyright [8] and provenance [9, 10]. Pragmatically, the reason of blockchain popularity may be simple: financial services and other applications are in a need of modernization, and blockchain technology seems to provide a solution [11]. Thus, the crucial question impacting the adoption of blockchain technology is whether the technology provides substantial benefits compared to alternate digital solutions.

Blockchains, as envisioned by Satoshi Nakamoto [12], combine the characteristics of three distinct technologies:

- Byzantine fault-tolerant systems
- Digital timestamping services
- Currency ledgers using cryptographic primitives

On its own, each of these technologies were studied well before Nakamoto's publication. Byzantine fault tolerance (i.e., tolerance to arbitrary malicious behavior, including attempts to thwart the system by an active adversary) was defined in 1980s [13, 14]; the practical Byzantine fault-tolerant (PBFT) algorithm [15], which is now frequently used as a building block for private blockchains, dates back to 1999. Digital timestamping – i.e., notarizing digital documents, associating these documents with reliable timestamps and establishing ordering among all notarized documents – was explored starting in 1990s [16, 17]. David Chaum's DigiCash [18] – an electronic ledger extensively relying on cryptography primitives – was founded in 1990 (and went bankrupt in 1998). It is the combination of timestamping, replicated log and cryptography that made Nakamoto's blockchain design innovative.

The digital timestamping feature of blockchain technology is sometimes overlooked in blockchain studies in favor of the other two aforementioned characteristics. In our opinion, this may lead to a distorted understanding of blockchain technology. If a blockchain is viewed only as a distributed fault-tolerant system with embedded business logic (such as a currency), it may seem that all blockchain users are equal; once we switch to the timestamping perspective, it becomes clear that this is not necessarily the case. Namely, there are entities that determine the state of the blockchain (e.g., a bank or a consortium of banks in a financial ledger; the registry agency in a public registry; insurance company or companies in an insurance network) and external users who do not participate in consensus, but would like to be sure that the blockchain is operated correctly (e.g., bank clients; immutable property owners; policyholders). An important category of external users are auditors and regulators.

One of the core features expected from timestamping services is *accountability* [19]. Accountability means that every user of a timestamping service can reliably verify that the service operates in the intended way (e.g., information provided by the service agrees with the information it provided to other users). If verification fails, the user has a proof of service's malicious behavior, which could be

used to hold the service accountable. Accountability may be lost if external parties are excluded from consideration when constructing the blockchain.

Feasibility of external audits is one of attractive characteristics of permissionless cryptocurrency blockchains such as Bitcoin. Bitcoin would not likely have gained as substantial popularity if it required each user to operate a full Bitcoin node to ensure the system operates correctly. In practice, the Bitcoin ecosystem allows each user to choose a fitting trust model to perform transactions: using a full node, a lightweight (SPV) node, a non-custodial multisignature wallet or a trusted third party. User trust in the automatically enforced properties of a blockchain, instead of in the identities of its processors (and by extension, trust in formally specified properties of services built on top of the blockchain instead of identities of the service providers) could be beneficial for both permissioned and non-permissioned blockchains. In other words, increased user trust could create an environment for even more third party service development and integration of blockchain technology.

Another useful feature provided by blockchain technology is *non-repudiation* [20], i.e., the ability to definitively verify authenticity of statements recorded on the blockchain. Non-repudiation could be accomplished with the help of digital signatures combined with public key infrastructure (PKI) [21] and reliable timestamping. The latter is important to prevent anyone (even the colluding blockchain maintainers) from backdating statements and to make retrospective authenticity verification not critically reliant on security of the utilized public key cryptosystem(s).

Many proposed blockchain applications (currency ledgers, public registries, insurance, copyright, supply chains, etc.) could benefit from built-in external auditability and non-repudiation, as they are legally required or at least expected by application users. External auditability of blockchains could make them a preferable means of computerization of such applications compared to alternatives. Furthermore, external auditability aligns with "Web 2.0," i.e., the general shift from service-centric to user-centric applications (cf. the notion of self-sovereign identity [22, 23]). The ability for any client to perform a partial or complete audit of the system could be viewed as a competitive advantage in the emerging user-centric world. Thus, auditability and accountability capabilities of blockchains constitute a promising topic for research.

Previous research. Accountability is one of core topics of the research on digital timestamping (see, e.g., [19]). The Bitcoin white paper mentions reliable timestamping as one of the design goals of the system (Sections 3, 4 of [12]). Interestingly, the reference list in the white paper contains 4 works on timestamping – more than on distributed computations and e-cash cryptography combined.

The topic of accountability was brought up in blockchain research with relation to the "nothing at stake" problem encountered by early proof of stake blockchains [24]; in fact, the problem could be viewed as a typical accountability failure. Later research on proof of stake codified the problems with accountability with the notion of weak subjectivity [25]. Security deposits were introduced into proof of stake to provide economic accountability of blockchain participants (see, e.g., Slasher [26], Tendermint [27] and Casper [28] consensus algorithms).

Private blockchain concepts, mostly based on the PBFT family of consensus algorithms, frequently declare auditability by external parties (i.e., accountability), often by introducing a special role of auditing nodes in the blockchain network (see, e.g., IBM Open Blockchain [29]). However, research analyzing and quantifying blockchain accountability is somewhat lacking (see [30] as one of examples). As we show in this paper, Byzantine fault-tolerant design and presence of auditing nodes in the system may not be enough to provide meaningful accountability for all applications.

Blockchain anchoring, which we consider in this paper as one of accountability measures, is similar to the notion of *attested append-only memory* (A2M) [31] in providing additional security for distributed consensus. Using the Bitcoin Blockchain as a means to provide accountability with the help of anchoring is explored by Proof of Existence [32], Factom [33], Tierion [34], Openchain [35], ChainDB [36], etc. Our description of anchoring a blockchain on the Bitcoin Blockchain in Section 3.3 resembles that used in Factom and Openchain with the exception of the structure and authorization of anchoring transactions.

We have briefly described anchoring and proof of work as means to achieve accountability in our white paper [37]. This work could be viewed as the extension and elaboration of that research.

Our contribution. We consider auditing capabilities, accountability and non-repudiation for state machine systems and define blockchains as a subtype of such systems that fulfill these properties. This conclusion is derived from the semantics of the word *blockchain*, as well as the design goals and linked timestamping roots of the Bitcoin Blockchain. Thus, our view of blockchains is narrower compared to defining a blockchain as an atomic broadcast ([38]; a similar or even looser definition is commonly used in non-specialized press). On the other hand, we do not limit accountability in blockchains to economic accountability provided by proof-of-work mining of native blockchain tokens.

As means to implement auditability, accountability and non-repudiation, we describe blockchain receipts (aka SPV proofs) and blockchain anchoring. We also examine proof of work in permissionless blockchains from the point of view of accountability. Both blockchain receipts and blockchain anchoring are well-known technologies; our contribution is in connecting them to accountability and, for blockchain anchoring, in reviewing implementations that would not compromise security assumptions on the system.

Contents. The rest of the paper is organized as follows. We describe auditability and accountability for state machine systems in Section 1. In Section 2, we reason that replicated logs without a proper setup are not accountable and that blockchain design specifically addresses this issue; we also study blockchain receipts as basic accountability measures. Section 3 describes traditional anchoring, proof of work and blockchain anchoring as means to increase accountability of blockchains. Section 4 is dedicated to an overview of alternative approaches to accountability and auditability. An analysis of accountability of alternative consensus algorithms used in cryptocurrencies is given in Appendix A. Finally, Appendix B offers a framework for estimating attack costs on blockchain anchoring.

1 Basic Definitions

We start by giving basic definitions within the problem domain.

Definition 1. A *state machine system* [39] is a software product comprising:

- **Data storage** reflecting the system state in a certain domain (e.g., users' balances in a ledger)
- **Transactions,** which are the atomic, linearly ordered, sufficiently small changes to the system state (e.g., value issuance or transfer in a ledger)
- **Consistency rules** allowing to assess whether the system operates correctly, i.e., according to expectations placed on it by its users. Consistency rules determine whether a particular transaction is correct with respect to the current system state. Each transaction can be either correct or incorrect, but not both or something in between. Incorrect transactions must not be enacted; any correct transaction can be enacted without compromising the system

In essence, we assume that changes to a state machine system could be decomposed into small pieces (transactions) applied in a sequential order, and the problem of ensuring the correctness of the system could be reduced to the computational task of ensuring the correctness of transactions. We also assume that consistency rules are objective and deterministic, thus can be automated. As an example of consistency rules, there are usually rules regarding system security [40] (e.g., all unauthorized transactions are incorrect).

Definition 2. *Auditing* is a systematic and independent examination of a state machine system with the goal of determining whether its operation is correct (according to the consistency rules) and was continuously correct in the past.

We use the common definition of auditing [41] and confine it to state machine systems. Therefore, we assume that such computer systems in general can and should reflect objective reality; we reject a nihilistic point of view that computer systems cannot automate auditing process, as the evolution of computer technology proves the opposite. In this regard, we do not see substantial differences between state machine systems and other data sources subjectable to auditing.

The goal of auditing is to prevent or minimize *system corruption*, i.e., a failure of some enacted transactions to agree with consistency rules. This goal could be achieved in several ways:

- Make system corruption technically impossible (*tamper proofness*)
- Make system corruption prohibitively economically costly for anyone to try (*tamper resistance*). Costs may come from various sources:
  - System corruption may require bribing / corrupting many independent parties
  - System corruption may require solving a difficult computational problem requiring non-trivial resources at attacker's disposal

- Make system corruption easily and immediately detectable and the evidence of it independently verifiable and irrefutable (*tamper evidence*)
- Make system corruption irrefutably attributable to specific persons or organizations

Examples of state machine systems requiring auditing are financial ledgers (e.g., currency ledgers used in banking; securities ledgers operated by stock exchanges) and public state registries. There are three main user roles in such a system: *maintainers*, *auditors* and *clients* (Table 1). The first two roles are self-explanatory; we define clients as entities who initiate and authorize (whether directly or with the help of trusted intermediaries) changes to the system state. The user roles may overlap; e.g., in a permissionless blockchain any client can act as an auditor and/or as a maintainer.

Table 1: Core notions of auditable systems in various domains

Property	Domain		
	Financial ledger	Public registry	Provenance
Maintainer	Bank(s), exchange(s)	Government agency	Goods manufacturer(s) or specialized blockchain provider(s)
Auditors	Internal auditors, regulators, law enforcement	Government-appointed auditors; NGOs	Customers
Clients	Bank clients; securities owners	Citizens (e.g., immovable property owners, copyright owners)	Goods manufacturer(s)
System state	Balances of all clients; state of contracts between clients	State of all registered property	Registered goods
Transactions	Value issuance and transfer	Ownership change, leasing	Goods issuance and tracking
Examples of consistency rules	The same asset does not belong to two accounts (i.e., no double-spending)	A property cannot be sold if its owner already leases it	Goods with the same tag cannot be sold twice

1.1 Internal and External Audits

We assume that a state machine system could be subjected to two kinds of threats.

Definition 3. *Local threat* is a threat corrupting a minor part of the system, so that operation of the major part remains correct. *System-wide threat* is a threat corrupting the system in whole.

An example of a local threat would be system components being corrupted by an external hacker or a rogue employee. An example of a system-wide threat is corruption of the bank ledger by the bank's executives in an attempt to hide its insolvency. Note that threats may be caused by operational

errors committed by persons interacting with the system, and not necessarily by internal or external malicious activity.

By discerning local and system-wide threats, we assume that the system can provide for a certain degree of redundancy so that each transaction is checked by multiple independent verifiers; local threats can corrupt only a certain minority of these verifiers. This assumption is not necessarily the case: in some centralized systems, any local threat would realistically escalate to a system-wide threat.

Assumption 1. Certain state machine systems necessitate continuous auditing of their operation both by internal and external parties:

- The goal of **internal audits** is to ensure protection against local threats
- The goal of **external audits** is to ensure that the system operates according to expectations of its clients and to the public interests, i.e., provide protection against system-wide threats

The notion of *accountability* is closely related to the feasibility of external audits. Accountability essentially means that external users (auditors and, ideally, clients) have the means to timely detect system corruption attempted by maintainers, attribute such activity and provide unambiguous proofs of it that could be used publicly, e.g., in a court of law. Correspondingly, we will use terms *external auditability* and accountability interchangeably.

By confining ourselves to Assumption 1, we assume that:

Assumption 2. The state machine system is *anti-discretionary* [42], i.e., there are objective consistency rules governing its operation which cannot be changed at will by the system maintainers.

Assumption 2 is true for regulated systems (public registries, banks, exchanges, etc.), in which case there are generally parties interested in the system auditability who do not maintain the system. For example, citizens may be interested in having an auditable immovable property registry without needing to participate in the registry operation (which may be unwieldy for an ordinary citizen); the same is true for bank or exchange clients. Another application area of Assumption 2 is peer-to-peer financial services (e.g., P2P lending and insurance). Naturally, cryptocurrencies are anti-discretionary as well; indeed, the goal of cryptocurrencies is to completely abstract from the identities of system maintainers (so called *trustless-ness* property).

At the same time, Assumption 2 does not hold for many ledger-like or registry-like systems (e.g., an in-game currency ledger), in which external audits are superfluous or illogical because of trusted parties being inherent to the system design. In some other cases (e.g., a supply chain ledger or an inter-bank settlement system), the initial requirements on the system may not satisfy Assumption 2 as these systems may not require external auditing from the start. However, these systems may eventually fall into the scope of Assumption 2 in the process of their evolution:

- It would be quite natural to use supply chain ledgers for determining taxation. In this case, ledger maintainers might collude to evade taxation

- Inter-bank settlement system could be fine-grained and include client accounts instead of bank accounts, thereby attracting clients' interest in the system auditability
- Inter-bank settlement system could be used by the regulator to assess systemic risks for the participating banks

An approach to automate internal audits by making a system fault-tolerant may be ineffective in the case of external auditing. Here is why:

Assumption 3. Maintainers of a state machine system may have a common interest to mislead an external auditor.

This assumption is quite sound if a system is operated by a single entity even if the maintained system is distributed. For example, a distributed bank ledger with separate nodes corresponding to regional departments could be secure against a rogue employee or a hacker (as it is reasonable to assume that the attacker cannot corrupt more than a couple of nodes); however, it could not be secure against the bank's board of directors conspiring to mislead the regulator (as in this case, *all* bank nodes may be rigged). Consortium systems may be prone to this kind of attacks too, as evidenced by practice [43, 44, 45, 46] and the following weak argument: if maintainers have reached the agreement to create and maintain a shared system, they may as well reach an agreement to rig it.

The theoretical goal of external auditability would be to make the system completely auditable by any interested party (as there is always a possibility of institutional auditors colluding with system maintainers). The biggest reason why public auditability is not implemented for existing systems is that it contradicts clients' confidentiality and/or similar concerns (e.g., non-disclosure of trade secrets). Hence, auditing on behalf of the clients or public is performed by the designated auditors who have a non-disclosure agreement with the system maintainers. Therefore, we will keep in mind that:

- There should be means to perform a complete audit of the system
- A system should ideally provide public auditability insofar it does not contradict the declared security properties (e.g., clients' confidentiality)

In particular, we would like to implement the following simplest audit capabilities for the clients:

- A client may want to know all concerning transactions (e.g., incoming and outgoing value transfers in a ledger) and receive timely updates on these transactions
- A client may want to receive *electronic receipts* for concerning transactions and for the part of the state of the system concerning the client (e.g., client's balance in a ledger). The authenticity of an electronic receipt should be independently verifiable; it should be difficult to create a fake receipt, both for a client and for system maintainers

Another property we would like to provide is *non-repudiation*, i.e., correspondence between the syntax of transactions recorded in the system, and real-world semantics of these transactions. For example, we want to ensure that the timestamp of any transaction is accurate, and that it has been

indeed authorized by the entities purported by its syntax. Basic non-repudiation is a well-studied problem usually addressed with public key cryptography and secure key management (e.g., using public key infrastructure with hardware tokens). A bigger challenge is *retrospective* non-repudiation, i.e., ensuring transactions remain non-repudiated if some keys (or even the whole underlying public key cryptosystem) are compromised, and/or if the system maintainers may act maliciously. This kind of non-repudiation is especially important for public state machine systems, such as public registries.

1.2 Auditable Logs

We are now ready to define the core notion of our study: auditable systems.

Definition 4. *Auditable state machine system* is a state machine system satisfying Assumptions 1–3 and providing internal and external auditors with sufficient means to conduct online, periodic and/or by-request audits of the system in order to minimize the risk of system corruption.

Auditability could be beneficial for system auditors since it provides them with data as to the state of the system, which could be used to quickly identify or prevent system failures (e.g., bank insolvency; damages incurred by a rogue employee). As audits and regulation in financial services are often performed on behalf of the system clients, they could indirectly benefit from the system auditability as well.

For system maintainers, profits from auditability are indirect and correspond to the induced security properties (e.g., counterfeit resistance, flexible regulation compliance framework, the off-the-shelf secondary market, and streamlined third-party application development capabilities in the case of a digital asset ledger). The cost of implementing auditability could be substantially reduced by using an existing blockchain as a cloud platform (PaaS) [47]; one may argue that external auditability is *the* enabling feature for the cloud deployment of auditable state machine systems.

In order to build an auditable state machine system, we postulate that the entire history of transactions is stored in a linear *audit log*. It is quite obvious that audits of a state machine system could be reduced to the audits of its log, as system states could be sequentially restored from the log.

The approach with a linear log is commonly used in distributed computing in *replicated logs* (see, e.g., [48]). Blockchains are a subset of replicated logs. Replicated logs provide sufficient and well-studied tools for internal audits; they enjoy correct operation even if a certain percentage of nodes in the system behaves incorrectly. Thus, we will assume that audited systems are built using a replicated log as a starting point.

Definition 5. A replicated log is *auditable* if it satisfies the following requirements:

1. **Log consistency:** All changes in the system audit log are valid according to the consistency rules. (Note that it follows from this property that the system state is also valid.)
2. **State consistency:** The state of the system corresponds to the state inferred by “replaying” all transactions in the audit log

3. **Transaction finality:** The audit log is append-only, i.e., transactions are never removed from the log, modified in the log or added to the log retroactively. The only possible operation is appending transactions to the end of the log
4. **Reliable timestamping:** transactions in the log are reliably timestamped with a degree of accuracy sufficient for a problem domain
5. **Log uniqueness:** The audit log is unique in the sense that all system users (including auditors and clients) never receive conflicting logs or system states
6. **Blame ascription:** It is possible to reliably identify parties failing to uphold to the properties above (e.g., in the case several conflicting system states are presented to different users)
7. **Retrospective auditability:** It is possible to check that properties 1–6 hold for any previous moment of time

Compared to the requirements on auditable systems outlined in Section 1.1, we add two additional requirements deserving explanation: transaction finality and reliable timestamping.

1.2.1 Transaction Finality

Transaction finality (also commonly referred to as *log immutability*) is a vital requirement for non-repudiation. Indeed, compromised finality makes statements about transactions challengeable; e.g., it becomes impossible to definitively state that a transaction has indeed occurred at the time implied by its timestamp and ordering rather than was retroactively added to the log later.

Transaction finality may seem to contradict transaction reversibility, which may be required by the problem domain. For example, a bank may want to reverse transactions during a chargeback procedure; a public registry may want to remove documents from the registry deemed falsified or otherwise void by a court decision. However, reversibility could be implemented without removing or modifying existing transactions in the audit log:

- In a financial ledger, a transaction may be logically reversed by creating a new transaction with the equal value flow in the opposite direction. Special authorization may be introduced for reversing transactions (e.g., by encoding the corresponding logic into the smart contracts) for compliance. For example, in the case of a centrally issued electronic currency the reversing transaction may be authorized by the currency issuer instead of the sender
- More generally, a special type of transactions could mark a previous log transaction as void

Similarly, log revisions that erase transactions entirely from the log, while more attractive to some parties, could lead to abuse of the system. A careful system design (e.g., anonymization of sensitive data with the help of cryptographic commitments, public-key encryption and zero knowledge proofs of data integrity) could take better care of the issue.

If there exists an ability for any party to revise the audit log, then the log no longer reflects *all* changes to the system state, i.e., it ceases to be a single source of truth about the system. Thus, a mutable log harms auditability, non-repudiation and increases risks of system corruption.

1.2.2 Timestamping

As with transaction finality, timestamping is beneficial for non-repudiation. Timestamping could be useful in financial contracts, establishing precedence for copyright, auction bids, etc. Transactions are timestamped by the system maintainers since clients' clocks are generally unreliable; it is assumed that the accuracy of timestamping could be later verified by auditors.

1.2.3 External Auditability Properties

Properties 5–7 from Definition 5 are fundamental for external auditability:

- It is usually not enough for an auditor to check that a system seems to be correct *now*; he needs to be sure that it operated correctly in the past, prior to the audit
- Similarly, it is instrumental that the audit log presented to an auditor correspond to the audit log presented to other users of the system. That is, system maintainers should not be able to craft a seemingly correct audit log specifically for the purpose of being audited
- If there are violations of the rules, they need to be attributable to specific entities, both for the purpose of investigation and to discourage system participants from transgressions

As we will see, these properties are substantially more difficult to implement than properties from Definition 5 corresponding to internal auditability.

2 Auditability in Blockchains

2.1 Replicated Log Structure

A generic replicated log consists of several *servers*, each of which maintains a local *replica* of the global log. Changes to the log (i.e., transactions) are requests sent by *clients*. Servers and clients communicate among themselves over an unreliable network, which may delay, lose, reorder or duplicate packets of data. Both servers and clients are identified within a certain public key cryptosystem; correspondingly, every message in the system is authenticated. Every replicated log implements a certain *consensus protocol* in order to reach common understanding among servers as to the system state; the key part of consensus is establishing the complete ordering of all the transactions. Consensus usually implies dynamic server roles (such as a Leader, Candidates and Followers in Raft), which change according to the network state to maintain system liveness.

Replicated logs are used for fault tolerance (i.e., there is a requirement for a system to remain operational under the assumption that some hardware or software components may fail). There are two main types of fault tolerance explored in replicated logs:

- **Ordinary (non-Byzantine) faults** include arbitrary network faults (duplicated and reordered messages, delays in message delivery, packet loss, etc.) and replicas being non-responsive, e.g., because of hardware faults. Ordinary fault-tolerant consensus algorithms include Paxos [49] and Raft [48], which are implemented in many distributed databases
- **Byzantine faults** [13, 14] correspond to inconsistent behavior of replicas, whether caused by benign faults or by an active adversary. Byzantine fault-tolerant (BFT) consensus algorithms include PBFT [15], QU [50], Zyzzyva [51], RBFT [52], Aardvark [53], etc.

Fault-tolerant systems have well-defined limits of tolerating faults. For example, it is impossible to reach agreement with even a single faulty server if the system is deterministic and asynchronous, i.e., when message relay and processing can be arbitrarily slow [54]. Thus, the network is usually considered to be partially synchronous [55]: bounds on message relay and processing exist, but are not known a priori[1]. If there can be f simultaneously faulty servers in the system, the minimum number of servers required for a functional ordinary fault-tolerant system is $2f + 1$; for a Byzantine fault-tolerant system, the minimum number is $3f + 1$ [58].

Because of the auditability requirement, the structure of an auditable log network differs from the client – server model utilized for replicated logs. Instead, it includes 3 types of replicating nodes corresponding to three user roles described in Section 1:

- **Consensus/maintainer nodes** are nodes that participate in consensus. Each consensus node maintains the up to date system state and possibly (but not necessarily), the complete audit log. Consensus nodes may be identifiable if it is required by the consensus algorithm
- **Verifying/auditing nodes** are nodes operated by external auditors. These nodes do not take part in consensus, however, they maintain the complete blockchain. An auditing node can verify the correctness of all transactions or an arbitrary subset of transactions not known to the system maintainers; it may perform online verification of incoming blocks of transactions or verify transactions on demand
- **Lightweight nodes** are nodes operated by clients in order to satisfy their needs in simplest audits (Section 1.1). Lightweight nodes do not participate in consensus and do not store neither the full system state nor the full blockchain. Instead, a lightweight node replicates a chain of block headers and transactions concerning the client. Lightweight nodes could be identifiable in order to limit transactions the node has the access to. Semi-anonymous lightweight nodes in a public system could be implemented using Bloom filters [59], as in Bitcoin[2]

It follows from the properties of BFT systems that a BFT replicated log with n consensus nodes may provide protection against local attacks in which the adversary controls at most $(n - 1)/3$ consensus

[1] An alternative approach relies on a secure source of randomness [56, 38], which allows building a Byzantine fault-tolerant consensus in a completely asynchronous setting. Another approach is to use *failure detectors* [57] that can tell with non-zero accuracy whether a certain replica works properly.

[2] Note that Bloom filters as implemented in Bitcoin do not provide a sufficient level of privacy [60].

nodes, making these attacks impossible to succeed under standard cryptographic assumptions. In order to provide full protection from local attacks, all components of the system need to be Byzantine-fault tolerant with the expected degree of redundancy; if there are centralized components, such as an authorization verification module, they are required to be incorruptible.

2.2 Blockchains

Given the description above, it may seem that a BFT system with appropriate business logic could be effectively used to implement an auditable log as per Definition 5. The main obstacle is that BFT systems are usually not adapted for external audits and do not mitigate system-wide attacks for the following reasons:

- Replicated logs usually contain log compaction logic, which is detrimental in audits
- All consensus logic is internal to the servers, whereas external parties (including auditors) have a limited view of the system operation. For this reason, we would want for the audit log to record voting results for each consensus decision; it could help both in ascribing blame and verifying the correctness of the audit log retrospectively
- To complete any request, a client needs to receive a response from at least 1/3 of consensus nodes (e.g., at least 34 consensus nodes in a system with 100 such nodes). This drastically differs from a common practice when a client communicates with a system using a single entry point, such as a website
- Log uniqueness is only partially addressed by BFT consensus. It is impossible to create an alternative log if the system corresponds to the BFT threat model (i.e., there is no more than 1/3 of faulty servers). However, the threat model itself may be inappropriate. If there is a possibility of maintainer collusion (Assumption 3), then the maintainers can overwrite the log partially or completely at any time, present different versions of the log to various clients and auditors, etc.

2.2.1 Transaction Blocks

Blockchain architecture solves problems with blame ascription, retrospective audits and interaction with clients by grouping transactions into *blocks* (cf. checkpoints in PBFT). Consensus decisions are made on the block level instead of the transaction level. Each block consists of two parts: relatively small block header, the size of which does not depend on the number of transactions in the block, and transactions. Transactions are committed to the block header, usually as a root of a Merkle tree [61] (*Merkle root*) since Merkle trees provide the optimal $\mathcal{O}(\log N)$ length of a proof of existence for a transaction included into a block with N transactions. A block header can also include:

- Reference to the previous block (a cryptographic hash of its header)
- Block timestamp
- Commitment to the state of the system (e.g., using Merkle Patricia trees [62, Appendix D])

- Data allowing to independently verify the consensus. In a consensus with known validators (e.g., Tendermint [27]), consensus-related data in the block header could be a set of digital signatures of the previous block belonging to more than 2/3 of validators. In a proof of work consensus, consensus-related data is a nonce and the network difficulty; a block is valid if its hash is less than a threshold value determined by the network difficulty

Grouping transactions into blocks is beneficial in several ways:

- Batching transactions decreases the number of messages sent over the network. Batching may increase delays in transaction processing, but the delay is acceptable for most applications
- Blocks boost the speed of verifying consensus decisions. The boost is particularly important if these decisions are encoded as digital signatures, as operations with digital signatures are computationally costly compared to other parts of block header verification
- Blocks introduce a possibility of efficient simplest audits and lightweight nodes (see Section 2.4)

Because of the block structure, a synchronizing node can reliably download blocks from any node regardless of whether it participates in consensus. Thus, the network of auditing nodes could provide a blockchain content delivery network (CDN) [63], which would distribute the load more evenly among nodes replicating the blockchain in whole. Bitcoin demonstrates a well-developed auditing node network with over 5,000 auditing nodes [64]. The number is particularly impressive because auditing nodes in Bitcoin perform online validation of all incoming blocks.

2.2.2 Autonomy

Another characteristic feature of blockchains is embedding authorization data into all transactions; i.e., for any transaction it is possible to determine who authorized the transaction. Authorization data is commonly represented as digital signatures, possibly of more than one party (e.g., multisignature wallets [65]; cf. multi-factor authentication).

More generally, blockchains are *autonomous* (i.e., minimizing reliance on external components as sources of truth), which makes blockchains good for external audits and strong non-repudiation policies. A scheme in which some transactions on a blockchain do not include *any* authorization data would not be autonomous, as it would rely on unprovable and non-auditable (within the blockchain) assumption that authorization logic was properly executed when the transaction was made. If a PKI-based scheme is used for authorization, all PKI operations (adding a new certificate, revoking a certificate, etc.) should ideally be recorded into the blockchain – otherwise the system becomes critically dependent on non-verifiable external assumptions.

Consensus verification data in block headers could be viewed as another example of blockchain autonomy. Similar to the example above, all changes to consensus nodes (adding a new node, removing a node from consensus, etc.) should be recorded in the blockchain to increase auditability.

With the autonomy property described, we are ready to define blockchains in terms of auditing.

Definition 6. A *blockchain* is a replicated, autonomous, Byzantine fault-tolerant log with consensus based on blocks that permits external auditing and lightweight nodes, and provides non-repudiation of the log entries.

Note that support of external audits and lightweight nodes can be directly inferred from the very term *blockchain* (i.e., organization of an audit log as a linear sequence of blocks of transactions that are linked within a block with the help of Merkle trees)[3]. Autonomy and having multiple consensus nodes are implied in the Bitcoin white paper and could be useful in most practical blockchain applications since these features provide internal auditability and fault tolerance. Similarly, accountability and non-repudiation are implicit goals of the Bitcoin Blockchain design and are achieved with proof of work and incentivization of the blockchain maintainers with the help of mining. We require these properties from blockchains, but do not enforce a particular way to implement them.

In the following discussion, we will consider a subset of blockchains with identifiable consensus nodes (permissioned blockchains); blockchains with anonymous consensus nodes (*permissionless blockchains*) are briefly considered in Appendix A.

Definition 7. *Permissioned blockchain* is an (atomically) consistent blockchain with the identifiable consensus nodes. Auditing and lightweight nodes in a permissioned blockchain may be identifiable according to security requirements on the system.

Atomic consistency property [66] is required in order for blockchains to fully satisfy Definition 5 with regard to transaction finality; BFT replicated logs with identifiable nodes are generally consistent by construction.

2.3 Accountability Threat Model

Per Section 2.1, permissioned blockchain designs take care of internal audits by making local attacks impossible provided security parameters of the system are chosen correctly (e.g., each component of the system has an appropriate level of redundancy). Therefore, we need to analyze susceptibility of blockchains to system-wide attacks.

For the purpose of external audits, the system looks as follows: There is a monolithic construct of system maintainers, which supply audit nodes with properly authenticated blocks (e.g., by using digital signatures of 2/3 of the consensus node set, as in Tendermint). The attacker is the colluding majority of system maintainers who control 2/3 or more of the consensus nodes. The goal of the attacker is one of the following:

- **Audacity attack:** make honest parties in the system accept an incorrect audit log
- **Revision attack:** retroactively modify the audit log and make honest parties accept the change
- **Equivocation attack** [31]: present various external nodes with the conflicting versions of the audit log (e.g., in order to hide service insolvency)

[3] Compare with the term *distributed ledger*, in which replication (or, more generally, distribution of data) among several consensus nodes is explicit, and other properties from Definition 5 may or may not apply.

By eliciting these kinds of attacks, we assume, per A. Lincoln's words, that the attacker "cannot fool all of the people all of the time," i.e., cannot continuously lie to all clients and auditors in the same manner for the prolonged period of time. If this assumption does not hold (e.g., if the blockchain in question has inappropriately lax consistency rules), neither blockchains nor alternative technologies can mitigate an attack. Additionally, we do not analyze **censorship attacks** since they do not violate blockchain consistency rules if a blockchain is autonomous; as censored transactions are not reflected on the blockchain, detecting censorship requires external sources of truth.

For the purposes of this paper, we restrict our definition of an attack in the following ways:

- The consistency rules of the system are known in advance and cannot be changed by the attacker
- The attacker does not control auditing and client nodes; e.g., there are no backdoors allowing to skip rule checks for the logs corrupted by the attacker on auditor and client nodes
- The attacker does not control the network. In particular, he does not control communication among honest parties, neither can he delay the deliverance of messages to honest parties for prolonged periods of time. Attack duration is much longer than a typical latency of message delivery in the system
- The attacker can lie consistently for a long period of time, but only if doing so is computationally and economically sound

In most cases, system-wide attacks cannot be considered impossible. We consider the attack thwarted if it is timely detected by honest parties and irrefutably attributed; this may prevent attacks by making their consequences (e.g., legal action) unfavorable for the attacker. Additionally, we want to make the direct attack costs as high as possible.

The above restrictions on the attacker may be too optimistic. For example, an authoritarian state government may coerce citizens to accept revisions to the blockchain-based public registry of immovable property and not be held liable despite compelling evidence; after all, blockchain technology is a human tool. Despite this, blockchain technology provides security aspects making it beneficial even in the worst case:

- Because blockchain data is distributed to lightweight nodes and auditor nodes out of the control of the attacker, the attacker would not be able to destroy the evidence pertaining to the state of the blockchain before the attack. The snapshot of the blockchain before the attack could still be accessed by clients and independently verified as authentic
- As blockchains are based on universal mathematical laws instead of trust in system maintainers, clients' appeal to courts could be more likely to succeed

2.4 Blockchain Receipts

Basic blockchain accountability could be provided with the help of *blockchain receipts* – a particular kind of electronic receipts. The general idea of a blockchain receipt is to provide a succinct summary

regarding a particular transaction or a part of a system state authenticated by system maintainers. Blockchain receipts are designed to be independently verifiable; e.g., they could be used by a client to prove a payment or his balance in third-party applications. At the same time, receipts improve accountability in the following ways:

- Blockchain receipts allow to efficiently verify that information provided to a client corresponds to the information provided to the system auditors
- The construction of blockchain receipts is such that malicious actions by system maintainers would render previously issued blockchain receipts invalid, therefore negatively impacting performance and increasing indirect costs of a system-wide attack

In order to deduce the framework for blockchain receipts, we use the following assumption.

Assumption 4. There exists at least one correct auditing node in the network at each moment of time.

Because the attacker cannot prevent auditing nodes communicating with each other, all auditing nodes will have the same version of the blockchain at all times. We assume the attacker cannot supply an auditor with an incorrect version of the blockchain because that would provide the auditor with the sufficient evidence of the attack.

Theorem 1. *Under Assumption 4, any system-wide attack defined per the threat model in Section 2.3 involves presenting some clients with a version of blockchain different from the one presented to auditors.*

Proof.

The attacker cannot present an auditing node with an incorrect version of the blockchain; hence, if an attack involves creating an incorrect audit log, it will differ from the version seen by the auditors. The attacker cannot revise auditors' version of the blockchain, hence, if the attack includes log revision, it also creates several versions of the blockchain. If the attacker's goal is to create several versions of the blockchain, the proof is self-evident. ■

Therefore, we can formulate the problem of thwarting a system-wide attack as follows.

Problem 1. The attacker provides system auditors with a cryptographically authenticated statement $\langle A, \sigma_A \rangle$, where A is a full audit log, and σ_A is its cryptographic authentication, and a client with an authenticated receipt $\langle B, \sigma_B \rangle$. The client wants to be able to check whether B contradicts to A.

If B is a free-form statement not tied to the blockchain in any way, the task of checking compatibility between A and B could be difficult, especially if the blockchain is fully private, i.e., the client has no information about A. In this case, the client may verify B only by submitting it to the auditors, which would arguably take place only if the client is already suspicious about the system operation. In order to simplify the check, receipts could be made electronic.

Definition 8. A *blockchain receipt* (called *SPV proof* in other research) for a certain transaction is a chain of block headers up to the block containing the transaction in question, together with the Merkle

path [61] to the transaction and the transaction itself in the same digital form as used in the blockchain (Fig. 1). Similarly, a blockchain receipt for a part of the system state at a certain block is a chain of block headers up to the block, a path in the Merkle Patricia tree and the state itself in the digital form.

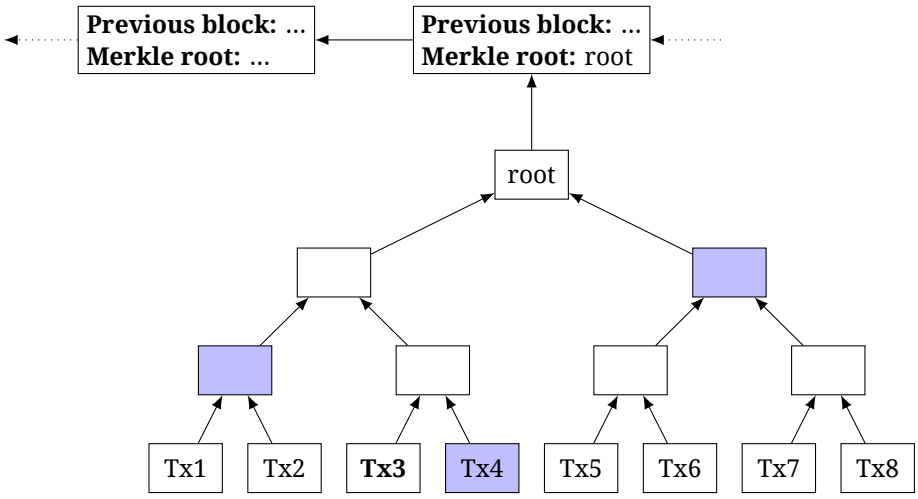

Figure 1: The structure of a blockchain receipt for a transaction **Tx3**. Hashes included into the Merkle path are marked with fill

A blockchain receipt can be verified in several steps:

1. Check that a transaction or a system state in the receipt is valid on its own right
2. Check that the hash of the transaction or the system state and the Merkle path in the receipt yield the same root hash as mentioned in the header of the corresponding receipt block B_{rec}
3. Check that each block header in the chain is valid according to the consistency rules (e.g., properly authenticated)
4. Check that the chain of block headers forms a link to the first blockchain block (*genesis block*) B_{gen} specified by the consistency rules
5. Verify that auditors have the same state of the chain. This could be accomplished by supplying the hash of B_{rec} to a web service maintained by an auditor

As per digital timestamping research, blockchain receipts could be made compact by introducing a non-trivial block *linking scheme* (i.e., by making each block reference a deterministic subset of previous blocks instead of the preceding block in the chain) [67, 19]. Linking schemes allow creating receipts with $\mathcal{O}(\log B_{rec}.h)$ block headers, where $B_{rec}.h$ is the height of the receipt block in the blockchain.

Because of the properties of Merkle trees, receipts are not falsifiable: it is impossible to create a valid receipt for a transaction or a part of the system state not present in the blockchain. It is also impossible to construct two different sets of transactions or Merkle Patricia trees for the system state, which would hash to the same root. Blockchain receipts ascribe blame in the following sense: if the

receipt is incorrect (on its own right, or together with some other receipts, e.g., as a result of a double-spend), or the chain of block headers in the receipt contradicts the one supplied to auditors, the blame is attributed to specific consensus nodes.

In order to make blockchain receipts more compact, block headers could be made public. This would not harm confidentiality, as block headers do not include any confidential information; in particular, headers contain no recoverable information about transactions in the block or the system state. A public sequence of block headers could serve another purpose: if the access to block headers is anonymous, the attacker cannot create multiple blockchain versions, as any of the requests to read block headers may belong to an auditor. Thus, block verification performed by clients would no longer require interaction with the auditors.

With public block headers, a chain of block headers in receipts could be replaced with a single block header hash (and possibly the block height to facilitate lookups). Lightweight nodes essentially implement this logic and collect client receipts together with replicating the chain of block headers. If a non-trivial block linking scheme is used, a lightweight node may skip downloading and verifying headers for blocks that contain no transactions related to the node (which, as stated previously, could be determined based on authentication or Bloom filters).

If the attacker revises the blockchain, all receipts in the blocks following the modification become invalid, because all hashes of succeeding blocks change. For this reason, it is not sufficient to include just the block height into a receipt, as Merkle roots and/or Merkle Particia roots in some succeeding blocks after the attack may remain the same. All lightweight nodes effectively cease functioning after a revision attack because they no longer accept block headers proposed by the attacker; because of the transaction finality property, any valid blockchain update would only append new blocks, and not replace them. Depending on the problem domain, non-functioning receipts and lightweight nodes may be harmful enough to deter a revision attack.

2.5 State Commitments

State commitments briefly described in Section 2.2 prevent a minority of malicious consensus or auditing nodes from withholding transactions from clients equipped with lightweight nodes. This is because nodes cannot fake state commitments; thus, withheld transactions would make the system state reported to the client inconsistent with that inferred by transactions received by the client.

Example. Consider a blockchain implementing an electronic currency; a state commitment in this case could be implemented as a Merkle Patricia tree mapping all accounts in the system to their balance. Client Alice connects to a malicious node (Mallory) using a lightweight node. Another client Bob then sends to Alice some currency. If there is no state commitment, Mallory can easily withhold this transaction from Alice; in order to detect foul play, Alice would have to connect to other nodes. On the other hand, if there is a state commitment containing all client balances, Alice can watch her balance and request a proof that her balance is indeed committed in the block header. Because of properties of Merkle Patricia trees, Mallory will not be able to construct such a proof for the old Alice's

balance once a transaction from Bob to Alice is included into the blockchain.

2.6 Whistleblowing

The setup described in Section 2.2 allows for effective whistleblowing in the case of malicious activity being perpetrated by system maintainers. A whistleblower can submit a *fraud proof* in the form of contradicting blockchain receipts (or, in the case of an equivocation attack, conflicting block headers at the same height) to the system auditors or leak the proof anonymously. Because of the structure of block headers and blockchain receipts, the revealed fraud proof does not require any additional authentication. A fraud proof warrants an investigation by the auditors, as it unambiguously signifies that the blockchain system is operating incorrectly: either the system maintainers are participating in a system-wide attack on the system, or an external perpetrator has successfully forged a fraud proof, which is only possible if he has compromised more than 1/3 of consensus nodes.

3 Increasing Accountability: Blockchain Anchoring

In some cases, Assumption 4 may be too optimistic. In particular:

- Auditors may be non-existent, may not function properly (e.g., due to the collusion with the attacker) or not to perform sufficient blockchain checks
- The attacker may control all sources supplying block headers to users, correctly identify the origination of each request and lie to them correspondingly. This is especially easy if access to the blockchain is authenticated and provided exclusively by the system maintainers

In order to prevent system-wide attacks under these conditions, blockchains may utilize anchoring and/or proof of work described below. Even if the system falls under the optimistic assumptions, additional measures to increase accountability, especially anchoring, could be quite cost-effective and would help diversify the security of the system.

Anchoring and proof of work help against equivocation and revision attacks; on their own, they do not prevent system maintainers from including contradicting transactions into the blockchain (audacity attacks in Section 2.3). Intuitively, audacity attacks may be overcome either by having functioning auditing nodes, or by using a public blockchain with encrypted sensitive data (Section 4.2).

3.1 Generic Anchoring

In traditional timestamping services, service accountability can be accomplished using *anchors*. An anchor is a cryptographic hash of the current system state (in terms of blockchains, the latest block header) published using an anchoring service – usually a printed medium (e.g., a newspaper). Anchors are published periodically, e.g., daily.

In a timestamping scheme with anchors, a user can check the existence a link between the receipt block B_{rec} and any anchored block B_{anc} (which may be situated in the blockchain before or after

B_{rec}) and ensure that the anchor is indeed present in the printed medium. A link between B_{rec} and B_{anc} could be provided interactively by system maintainers; in the case the sequence of block headers is public, it could be obtained non-interactively. In the simplest case, the link is the chain of block headers between B_{rec} and B_{anc}; using linking schemes, it is possible to shorten the length of the link to $\mathcal{O}(\log |B_{rec}.h - B_{anc}.h|)$. In order to increase reliability of the result, the verifier may try several anchored blocks B_{anc}; verification must succeed for all of them. To ensure accountability, it suffices for every client to check a link to the first anchor published after B_{rec} [19].

Because of the blockchain construction, it is infeasible to create a link from an anchored block to a block not present in the same chain. Thus, it is guaranteed that all receipts with the receipt block up to the latest anchor belong to the same version of the blockchain. On the other hand, receipts created *after* the latest anchor are unreliable in the sense that the attacker may produce seemingly valid receipts for data on several alternative chains (Fig. 2).

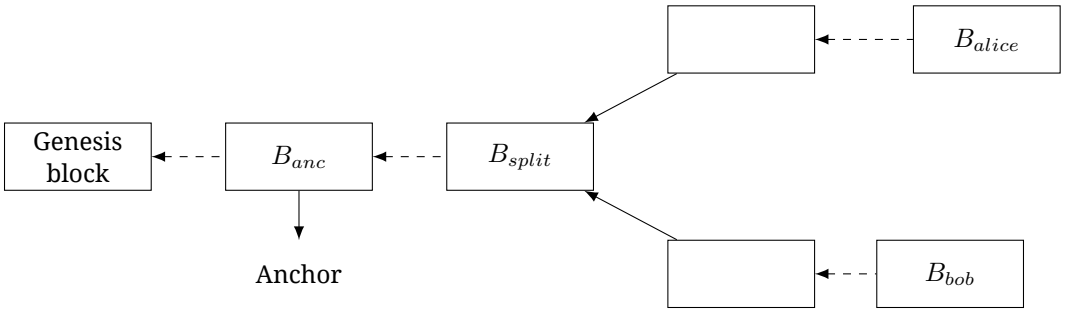

Figure 2: Seemingly valid, but contradicting receipts provided by EveBank for clients Alice (with the chain of blocks ending at B_{alice}) and Bob (with the chain of blocks ending at B_{bob}). To produce contradicting receipts, EveBank creates two versions of the blockchain with the split starting at the block B_{split}, which is located after the latest anchored block B_{anc}. Any anchor checks performed by Alice and Bob will succeed, despite them seeing different versions of the blockchain. Notice that in order to perform the next anchoring, EveBank needs to choose a single blockchain version; thus, at least one of Alice's and Bob's receipts will become invalid after the next anchoring.

The security of anchoring comes from the following assumptions:

- The printed medium is publicly available, meaning verification could be performed by any user
- Each copy of the printed medium contains no more than a single anchor
- The printed medium cannot identify its reader and display different anchor values to different readers (i.e., demonstrate Byzantine behavior). Creating separate versions of the medium for specific readers may be risky and economically costly
- In order to retroactively modify the blockchain, the attacker would need to reprint all medium copies after the retroactive modification and destroy the existing ones. This could be practically infeasible or very costly for a comparatively popular printed media

While these assumptions are sound, using printed media for anchoring has certain drawbacks. Most printed media are issued no more often than daily; thus, the pool of unreliable latest transactions not secured by an anchor may be quite large. Additionally, accessing anchors in printed form could be inconvenient; it introduces the human factor into the verification procedure.

Publicly available block headers replicated by auditors and lightweight nodes could be viewed as a particular case of anchoring. If the blockchain is replicated only by the auditors, and clients are not aware of blockchain data, the anchoring procedure is no longer effective against equivocation attacks; as the auditing nodes are identified within the system (either explicitly or implicitly as all nodes except for the consensus nodes), the colluding system maintainers can provide the auditors with consistently falsified information.

Timestamping authorities (TSAs) [68] could be treated as a subtype of anchoring services. A TSA is a trusted third party providing reliable timestamping; in the blockchain context, one or more TSAs could be used to periodically timestamp the latest block header. Utilizing TSA services in a blockchain system does not solve the problem of accountability completely, simply shifting it from the blockchain maintainers to the TSAs. Furthermore, TSAs introduce additional trusted parties into the system. Nevertheless, TSAs could complement the accountability measures described below, as well as employ these measures themselves.

Anchoring provides retrospective non-repudiation even if a public key cryptosystem used by the blockchain (but not hash functions used for linking transactions) is compromised[4]. In this case, anchors can help provide a definitive answer whether a given blockchain receipt was produced at the implied time. As it is currently understood, quantum computers would compromise presently used public key cryptosystems (e.g., RSA and elliptic curves), but not hash functions; therefore, non-repudiation provided by anchoring can be quite meaningful.

3.2 Proof of Work

In the blockchain design proposed by Satoshi Nakamoto, accountability is achieved without relying on third-party anchoring services, which is specifically addressed in Sections 3, 4 of the Bitcoin white paper. Instead, Nakamoto's design relies on proof of work (PoW): the cryptographic hash of each block header treated as an unsigned integer needs to be less than a specific threshold which depends on the consensus-based network difficulty. The difficulty is automatically adjusted to keep the expected time interval between blocks constant (10 minutes in Bitcoin). The consensus algorithm assumes the blockchain with the greatest cumulative block difficulty as valid.

System maintainers in PoW blockchains are not identified for consensus. The maintainers are incentivized with cryptocurrency native to the blockchain, by discovering (mining) valid blocks (*block reward*), and by including transactions into a block (*transaction fees*).

PoW consensus design and miner incentivization provides economic accountability:

[4] Non-repudiation accounting for possible weaknesses in hash functions may be achieved by utilizing measures akin to those described for *evidence records* in RFC 4998 [69].

- As the blockchain is fully public, conflicting versions of the blockchain are easily detectable
- System maintainers spend real-world resources (e.g., electricity) on proof of work. Therefore, equivocating is *measurably* costly; playing against the rules would result in system maintainers losing their revenue. Similarly, retroactively modifying the blockchain requires resources to produce a blockchain that overrides the blockchain produced by honest system maintainers

The cost of retroactively modifying a PoW blockchain measurably increases with time passed since the intended modification (i.e., PoW blockchains are designed with long-term non-repudiation in mind). The older the modified transaction, the more computational work to create a blockchain accepted by honest nodes (Appendix B; cf. the number of issues needed to be reprinted and swapped during an attack on printed media anchoring).

Because PoW is *secret-free* (i.e., its security is not contingent upon non-disclosure of information, such as private key material when using public key cryptosystems), there is no way to reduce attack costs short of breaking the hash function that PoW is based on. Compare with traditional financial systems, in which the attack costs can be substantially reduced with access to secrets and/or negligence by the personnel responsible for the manual verification of transactions (for an example, see the compromise of the SWIFT network in 2016 [70]).

In order to remove, add or modify a sufficiently old transaction (>1 day old) from the PoW blockchain, an attacker would need to:

1. Acquire enough proof-of-work mining equipment in order to compete with the honest system maintainers. Rational maintainers are unlikely to participate in long-term attacks on the system, as it would cause substantial problems with the blockchain operation for a prolonged period of time and would therefore negatively impact the price of the blockchain cryptocurrency. Therefore, cryptocurrency rewards obtained as a result of the attack is unlikely to cover attack costs for the participating maintainers.
2. Provide infrastructure for the mining equipment (electricity supply, cooling, datacenter housing, etc.). Mining process could need to be geographically distributed for fault tolerance.
3. Operate the mining equipment continuously for a long period of time in order to override the blockchain produced by honest transaction processors. If the attacker has two times the hashrate of honest system maintainers, he needs the same time as the age of a modified transaction at the start of the attack (e.g., a 1-year long attack for a year-old transaction) [37].

Thus, the cost of a long-term attack is proportional to the cost of mining equipment securing the PoW blockchain. In blockchains with memory-hard proof of work (e.g., Ethereum), mining is mostly performed on general-purpose GPUs; therefore, an attacker could decrease attack costs by renting equipment for the attack duration, acquiring it with the help of botnets or selling the equipment after the attack is accomplished. In the case of Bitcoin, the attack cost is increased by the fact that mining equipment is highly specialized. The expenses on designing, producing, and maintaining bitcoin mining equipment necessary for a long-term attack would be prohibitively high for many practical

applications (they could be estimated as a billion US dollars), and they steadily increase as a result of market competition among bitcoin miners.

3.3 Blockchain Anchoring

Observe that requirements on anchoring services described in Section 3.1 are similar to that of auditable logs. Indeed, an anchoring service requires (anchor) finality, reliable timestamping and uniqueness capabilities. Therefore, it could make sense to use another blockchain as an anchoring service, provided that the latter is public and sufficiently distributed (in order to prevent crafting data for a particular user) and is accountable on its own. As shown in Section 3.2, blockchains with proof of work (e.g., Bitcoin) satisfy accountability without requiring anchoring. Thus, these blockchains could be used for blockchain anchoring.

Definition 9. *Anchored blockchain* is a blockchain that requires anchoring (e.g., a public registry). *Target blockchain* is a PoW blockchain that acts as the anchoring service (e.g., the Bitcoin Blockchain).

The basic blockchain anchoring procedure is as follows.

1. Obtain the compact form of the current state of the anchored blockchain, i.e., a cryptographic hash of the header of the latest block in the anchored chain
2. Create an anchoring transaction for the target blockchain authorized by the supermajority of consensus nodes specified according to security assumptions on the anchored blockchain. Per ordinary Byzantine fault tolerance assumptions, the anchoring transaction should be authorized by more than 2/3 of consensus nodes
3. Broadcast the anchoring transaction to the target blockchain, or send this transaction directly to known maintainers on the target blockchain who may provide a SLA for anchoring
4. Wait until the anchoring transaction is sufficiently confirmed according to security assumptions for the target blockchain. For example, a Bitcoin transaction may be considered practically final after it has 6 confirmations [71] (i.e., when there are 5 blocks built on top of the block including the anchoring transaction)
5. (Optional) Add into the anchored blockchain an SPV proof for the placement of the anchoring transaction on the target blockchain

Anchoring could be performed periodically (e.g., every 30 minutes). Depending on the problem domain, blockchain anchoring could be considered mandatory for blockchain validity (e.g., the blockchain may be considered invalid if no anchor was produced for the latest 12 hours). Otherwise, a failure to produce anchors for the extended period of time could warrant the investigation by the auditors and carefully worded alerts for the clients.

Compared to traditional medium anchoring, blockchain anchoring has the following advantages:

- Blockchain anchors could be verified automatically by auditing and lightweight nodes (e.g., by using a lightweight node for the target blockchain) and be more accessible than printed anchors

- Blockchain anchoring could be significantly more frequent, meaning there is less non-anchored transactions at each moment of time
- Blockchain anchoring is measurably cheap (its price per anchor is generally determined by transaction fees on the target blockchain) and permissionless, meaning anchoring could be performed without reaching an agreement with the anchoring service
- Blockchain anchoring could provide more measurable attack costs than traditional anchoring
- Blockchain anchoring allows for easier anchor authorization

As we argued in Section 3.2, the Bitcoin Blockchain is the most resilient among PoW blockchains in terms of attack costs. The attack would most likely fail to be covert because of the necessary preparations and, in any case, would fail to covertly replace the anchors (as the target blockchain is fully public), therefore defeating the attack purpose. An analogue of this "attack" in the case of printed media anchoring would be an attacker buying the printed media or disrupting its operation; in both traditional and blockchain anchoring, such an "attack" would be obvious to the auditors and clients.

The rest of the section will be dedicated to anchoring on the Bitcoin Blockchain specifically.

3.3.1 Anchoring Transaction

The structure of anchoring transactions is determined as a part of consensus rules on the anchored blockchain. In the basic case, the anchoring transaction consists of one input and two outputs. The transaction spends on transaction fees a certain number of bitcoins associated with an address jointly managed by consensus nodes. (We consider below two practical approaches of joint address management: Bitcoin multisignatures and threshold ECDSA signatures, each of which has its own pros and cons.) The change is returned to the same address or another address also jointly controlled by consensus nodes. Besides the change output, the anchoring transaction contains a provably unspendable **RETURN** output, which is used to store the anchored information. The approximate structure of this output is as follows:

- A short string serving as an identifier of the anchored blockchain (≈4 bytes)
- Height of the anchored block to facilitate anchor verification (≈4 bytes)
- Anchored block header hash (32 bytes)

The consensus rules should determine all anchoring transaction contents except for authorization. For example, the consensus rules should deterministically select an unspent transaction output for spending in the case several are available, transaction fees, etc. Authorization of the anchoring transaction generally requires interaction among the consensus nodes and thus needs to be a part of the consensus algorithm.

3.3.2 Bitcoin Multisignatures

The most straightforward scheme for Bitcoin anchoring is to utilize Bitcoin multisignature transaction capabilities [65]. In this scheme, every consensus node on the anchored blockchain independently generates a distinct key pair $\{(sk_i, pk_i)\}_{i=1}^{3f+1}$ in the secp256k1 elliptic curve cryptosystem. (Recall that f is the maximum number of Byzantine nodes in the system and $3f + 1$ is the minimum number of consensus nodes required to tolerate this number of Byzantine failures.) Public node keys pk_1, pk_2, ..., pk_{3f+1} are then publicly announced in order to facilitate audits; they may be certified by certificate authorities. A pay-to-script-hash is formed from the public keys:

$$\textbf{HASH160}\ H_M\ \textbf{EQUAL},$$

where H_M is the 20-byte hash of a standard multisignature script, which allows for a transaction to be authorized by any $2f + 1$ consensus nodes:

$$H_M = \mathsf{hash160}(\{2f + 1\ pk_1\ pk_2\ \ldots\ pk_{3f+1}\ 3f + 1\ \textbf{CHECKMULTISIG}\}).$$

(Hereafter, figure braces in Bitcoin Script correspond to the parts of the script jointly serialized as a single data item.) H_M determines Bitcoin address M jointly managed by the consensus nodes, which is used to pay transaction fees for anchors. The balance of the address could be replenished automatically or manually by maintainers of the anchored blockchain or by third parties.

To authorize an anchoring transaction, consensus nodes independently sign it and submit the signatures, which are then agreed upon as a part of the consensus (Fig. 3). In a PBFT/Tendermint-like consensus, agreement could be reached within one block after the block being anchored. Indeed, during voting for the next block B_{next} after the anchored block B_{anc}, consensus nodes can share among themselves individual signatures on the anchoring transaction similarly to the signatures on B_{anc} (which are to be included into B_{next}). When B_{next} is finalized, the set of signatures on the anchoring transaction for a supermajority of consensus nodes is finalized as well. After that, the transaction can be submitted into the Bitcoin network by any consensus node.

Inputs	**Outputs**
(**Amount:** balance)	**Amount:** balance − fee
Script: $0\ \sigma_{i_1}\ \sigma_{i_2}\ \ldots\ \sigma_{i_{2f+1}}\ \{2f + 1$	**Script:** **HASH160** H_M **EQUAL**
$pk_1\ pk_2\ \ldots\ pk_{3f+1}\ 3f + 1$ **CHECKMULTISIG**$\}$	
	Amount: 0
	Script: **RETURN** $\{I_{chain}\ B_{anc}.h\ H(B_{anc})\}$

Figure 3: The structure of the anchoring transaction using Bitcoin multisignature capabilities. $\{\sigma_j\}_{j=1}^{2f+1}$ are signatures of consensus nodes with monotonically increasing indices $1 \le i_1 < i_2 < \cdots < i_{2f+1} \le 3f + 1$; I_{chain} is the anchored chain identifier; $B_{anc}.h$ and $H(B_{anc})$ are the height and the hash of the anchored block, respectively. Signatures σ_j are agreed upon among consensus nodes as a part of the consensus algorithm on the anchored blockchain.

In practice, Bitcoin native multisignature capabilities are somewhat limited by anti-DoS measures put forward by the notion of *standard transactions* [72]. Non-standard transactions are commonly not relayed by Bitcoin nodes; most bitcoin transaction processors currently create blocks with standard transactions only. According to the current Bitcoin specification [73], standard transactions can contain no more than 15 public keys in a single multisignature, which effectively yields the restriction on the number of Byzantine nodes $f \leq 4$. Correspondingly, Bitcoin multisignatures could be only used for anchoring blockchains with a small number of consensus nodes. Even if the restrictions put forward by standard transactions are lifted, anchoring transaction size linearly increases with the number of consensus nodes on the anchored blockchain; i.e., managing anchoring transactions becomes more complicated.

3.3.3 Threshold ECDSA Multisignatures

Threshold ECDSA signatures [74, 75, 76] allow to compress anchoring transaction authorization into a single ECDSA signature. Correspondingly, the anchoring transaction would use an ordinary pay-to-pubkey-hash address corresponding to a certain public key pk. Signatures on transactions keyed by pk is produced by the supermajority of consensus nodes (Fig. 4).

Inputs	**Outputs**
(**Amount:** balance)	**Amount:** balance $-$ fee
Script: σ pk	**Script:** **DUP HASH160** hash160(pk)
	EQUALVERIFY CHECKSIG
	Amount: 0
	Script: **RETURN** $\{I_{chain}$ $B_{anc}.h$ $H(B_{anc})\}$

Figure 4: The structure of the anchoring transaction using threshold ECDSA signatures. Braces correspond to the parts of Bitcoin Script jointly serialized as a single number. Signature for the transaction input is obtained as a part of the consensus algorithm on the anchored blockchain.

Threshold signatures use Shamir's secret sharing technique [77] to distribute *shares* of the private key sk corresponding to pk, among the consensus nodes. During signing, every node creates a *partial signature*, which are then exchanged among nodes to compute the combined signature keyed by pk. Unlike native Bitcoin multisignatures, threshold signatures are interactive (i.e., require cooperation among the consensus nodes to be produced); thus, the time and computational resources spent on producing a combined threshold signature could be greater than in the case of Bitcoin multisignatures.

The threshold signature protocol described by Gennaro et al. [76] satisfies Byzantine fault tolerance assumptions, i.e., can tolerate up to f Byzantine faults in the system with $3f + 1$ nodes. Therefore, threshold signatures could be used for authorizing anchoring transactions without relaxing security assumptions on consensus nodes. To further increase resilience of the signing scheme, the threshold signature protocol allows proactively refreshing private key shares among consensus nodes while

keeping the combined private and public keys the same [78].

Compared to native Bitcoin multisignatures, threshold signatures are more compact. The size of anchoring transactions does not depend on the number of consensus nodes; thus, threshold signatures could be used for anchoring blockchains with hundreds of these nodes. This improvement is achieved at the cost of requiring $\mathcal{O}(f^2)$ dedicated computations performed on each consensus node [76].

3.3.4 No Multisignatures

A simpler version of anchoring could be obtained by relaxing fault tolerance of the process, i.e., not requiring a 2/3 supermajority of the consensus nodes to form an anchoring transaction on step 2 of the procedure on p. 24. For example, anchoring transactions could be submitted by a single consensus node rotated according to a certain protocol, as it is performed in Factom. Compared to the fault-tolerant methods described above, single-node anchors may require more communication among nodes and may necessitate out-of-band methods to interpret certain events (e.g., if a consensus node submits an anchor of a seemingly incorrect blockchain state).

4 Alternatives and Improvements

4.1 Trusted Hardware

System maintainers could install untamperable hardware security modules (HSMs) [79] that can be certified by the system auditors. These HSMs could implement the blockchain logic in whole or in parts; e.g., an HSM could provide consensus logic and associated digital signatures for the blockchain. The HSMs could be periodically inspected by the auditors and/or provide the auditors with online updates via encrypted channels that cannot be tampered with by the system maintainers.

One could say that proof of work hardware is a special type of HSM, the operation of which is transparent to external observers, as proofs of work are impossible to fake and are easy to verify without compromising security of the system. At the same time, most of operations performed by ordinary HSMs (e.g., digital signing) rely on trusting the HSM properties (such as the impossibility to extract a private key from the HSM) that can hardly be verified remotely and independently. In general, the HSM approach could be seamlessly combined with other accountability means, including blockchain receipts, anchoring and proof of work.

4.2 Fully Public Blockchain

Instead of making the chain of block headers public (which is proposed in Section 2.4 to decrease the length of blockchain receipts), the whole blockchain could be made public, thus allowing every interested party with sufficient computational resources to act as an independent auditor. The concept of *partial verification nodes* [12] is aimed to provide accountability with the moderate amount of computational resources allocated for each auditing node. A partial verification node verifies a

comparatively small percentage of transactions (e.g., 1%); in the event incorrect blockchain operation is discovered, the node broadcasts a fraud proof functioning similar to ones described in Section 2.4.

The mentioned approach is utilized in cryptocurrency blockchains, such as Bitcoin. Solutions developed by the cryptocurrency community provide confidentiality of transaction amounts and/or transacting entities [80, 81, 82], obfuscation of relations among clients (hierarchical deterministic wallets [83], Bitcoin payment protocol [84]), confidential smart contracts [85], etc. We expect for new, more mature solutions in the area to emerge in the short to medium term. These solutions would allow to prevent audacity attacks (Section 2.3) that cannot be addressed by blockchain anchoring.

4.3 Blockchain as a Service

Instead of deploying an application-specific private blockchain, some applications may utilize existing blockchains (whether permissionless or permissioned) as a specialized cloud platform – *blockchain as a service* (BaaS); see [47] for a description of a BaaS platform for digital asset management. From the economical point of view, blockchain as a service may be a more cost-effective and secure solution for small and medium enterprises compared to the deployment of an application-specific blockchain; moreover, in the former case, the enterprise could benefit from the network effect, availability of third-party blockchain applications, etc. On the other hand, BaaS solutions may not be viable because of domain-specific requirements; e.g., a public registry could hardly be organized on a permissionless blockchain (at least in the nearest future) because of the legal obligations of the registry maintainer.

The accountability in BaaS is partially relegated to third parties (system maintainers of the BaaS blockchain). The BaaS maintainers guarantee transaction finality, reliable timestamping, blockchain uniqueness, etc. If the customer base for the BaaS is diversified, maintainers could be less likely to perform system-wide attacks on the system for the sake of a single application than in the case the same application is hosted on a separate blockchain, as the discovery of the attack would have a negative effect on the system in whole.

Application-specific accountability in BaaS could be achieved with the help of expressive means of the underlying blockchain and cryptographic primitives such as Merkle trees. For example, solutions by G. Maxwell [86] and J. Bonneau et al. [87] both provide tools for bitcoin exchanges to prove their solvency while preserving a sufficient level of confidentiality.

5 Conclusion

Accountability and universal auditing capabilities are strong points of the blockchain design proposed by Satoshi Nakamoto. The term *blockchain* is derived from a hash-based linking among transactions, the main purposes of which are:

- Make blockchain revisions and equivocation detectable and costly (i.e., ensure accountability of block producers)
- Enable audits by computationally and space-constrained lightweight clients

Thus, the term *blockchain* puts an emphasis on accountability, whereas some proposed alternative terms (e.g., *distributed ledger*) focus on distribution and sharing of data (for which alone blockchain technology may not be necessary). In the light of calls for service transparency and accountability, boosted by computerization and the spread of the Internet, blockchains could provide a necessary framework to implement these properties.

In the Nakamoto's design, the accountability of system maintainers is obtained through economical mechanisms, meaning that the attack costs could be independently assessed with a reasonable degree of accuracy. An alternative approach – Byzantine fault-tolerant replication – offers tamper proofness for a substantial range of attacks by an external computationally bounded adversary, but does not provide a clear estimate of attack costs, which critically depend on the security of node identification. Additionally, Byzantine fault tolerance on its own is not effective against attacks perpetrated by the colluding system maintainers; without a proper setup, such attacks cannot be timely detected by clients and auditors. The allure of blockchain technology lies in providing protection against these kinds of attacks and attaining in this way a greater degree of transparency and accountability.

In this paper, we demonstrated how accountability is implicitly present in proposed blockchain designs that use BFT replication, and how it could be augmented with the help of blockchain receipts and anchoring (including blockchain anchoring). Blockchain receipts and anchoring provide the tamper evidence property, i.e., timely detection of attacks on the blockchain system (including attacks perpetrated by the system maintainers). Anchoring on a PoW blockchain also provides tamper resistance, i.e., a measurable economic expense required to even attempt an attack on the system. These security properties, together with tamper proofness against external threats, could justify the use of blockchains in a variety of security-sensitive applications, including scenarios with a blockchain being used as a specialized PaaS.

Blockchain receipts could provide a basic level of accountability and serve as an alternative to traditional receipts. Using linking schemes inspired by timestamping services and/or publishing the chain of block headers (which contains no recoverable confidential information about the blockchain operation) could make blockchain receipts compact and easy to verify. Receipts together with anchoring could provide strong guarantees for non-repudiation, including the case when a public key cryptosystem used by the blockchain system is compromised.

Blockchain anchoring does not require substantial expenses from system maintainers and at the same time provides a substantial degree of resilience against attacks. The Bitcoin Blockchain is the most cost-effective blockchain for anchoring because of the greatest network hashrate and the use of ASIC-friendly proof of work. Anchoring could be deployed on a blockchain together with traditional anchoring on printed media, trusted timestamping and/or hardware security modules in order to diversify security and accountability of the system.

As an auxiliary accountability service, blockchain anchoring does not constitute a single point of failure. Even in the extraordinary event that the blockchain with anchors ceases functioning or undergoes a hostile takeover, the anchored blockchain would continue normal operation, as the event

would be observable and interpretable in the same way by the system maintainers, auditors and clients. Thus, blockchain anchoring could be used in applications without sacrificing availability.

Appendix A Accountability in Permissionless Blockchains

The popularity of Bitcoin has given rise to other cryptocurrencies, with various consensus algorithms and, correspondingly, varying accountability of participants (Table 2). The main three approaches to consensus in cryptocurrencies are proof of work (PoW), proof of stake (PoS) and the web of trust (WoT) approach; there are also hybrid consensus models using proof of work and proof of stake. Accountability properties of PoW were considered in Section 3.2. First proposed PoS versions, in which every cryptocurrency holder can create blocks with the probability proportional to his balance, suffer from impaired accountability (so called "nothing at stake" problem). We address the reader to [24, 88] for the analysis of simple PoS schemes and their comparison with PoW. In the more advanced versions of proof of stake (referred to as *deposited-based*, or *punitive*, proof of stake – DPoS), consensus nodes make security deposits, which are confiscated in the case of incorrect behavior. The consensus weight of a node is proportional to the amount of the deposit. Deposits address the accountability problem and make accountability in delegated PoS measurable in terms of the blockchain cryptocurrency (cf. mining equipment and electricity costs in PoW both measurable in terms of the *real-world* resources). Finally, in the WoT approach, each client selects a set of nodes he trusts; thus, accountability is trust-based and does not differ much from existing financial services and other applications, in which blockchains could be used.

Table 2: Accountability and a category under the CAP theorem [89] for various kinds of consensus algorithms used in permissionless blockchains. Availability is understood more loosely than in the general case (see in text)

Consensus	Examples	CAP category	Accountability
Proof of work	Bitcoin, Ethereum (Ethash), Litecoin	AP	Economical through proof of work; deviations from the protocol make a maintainer waste real-world resources
Proof of stake (simple)	Peercoin, Nxt	AP	Impaired – "nothing at stake"; deviations from the protocol are easy
Delegated proof of stake	Tendermint, Ethereum (Casper)	CP	Economical through deposits; deviations from the protocol make a maintainer lose in-blockchain currency
Web of trust	Ripple, Stellar	CP	Trust-based

Blockchain utilizing PoW or simple versions of PoS do not fully satisfy Definition 5 of auditable logs; they allow for blockchain reorganizations, thereby not enjoying atomic consistency. Intuitively, the consensus algorithm not making any assumptions about the behavior of consensus nodes cannot be atomically consistent. Indeed, in such systems, it is impossible to discern a network split with the

case when the substantial portion of consensus nodes has permanently abandoned their duties. Thus, the best course of action of consensus nodes that perceive a split from the majority of the network is to continue working, taking for granted the risk that their work may be voided after partition healing.

Therefore, PoW and PoS blockchains fall under the AP (available and partition-tolerant) category under the CAP theorem, with availability redefined per the realities of permissionless blockchains as the finite time for a transaction *with sufficient fee* to be included into a block; if we required for the network to process *all* transactions in finite time as per the ordinary definition of availability, it would be vulnerable to transaction spam. DPoS and WoT consensus algorithms achieve consistency at the cost of identifying nodes and/or assuming their fault tolerance:

- DPoS algorithms identify nodes to assign them weights according to security deposits. The real-world identities of consensus nodes may or may not be known; intuitively, stakeholders with large security deposits in a popular DPoS system would probably have known identities
- In WoT systems, consensus nodes are identified in order to build trust relationships. Intuitively, the real-world identities of the consensus node owners should be known, as it would be strange for users to trust the nodes otherwise

Proof of stake and web of trust blockchains are not as attractive as a medium for anchoring as proof of work blockchains. Indeed, PoS itself suffers from impaired accountability, and WoT gives an inherently subjective view of the system. DPoS blockchains could potentially be better in this regard, although long-range attacks for them could be easier than for PoW blockchains [88]; presently there are no DPoS blockchains that could offer the same level of economic accountability as Bitcoin. Additionally, neither (D)PoS nor WoT algorithms are secret-free like PoW. Their security depends on secrets in the form of private keys, which are necessary for identification of accounts in the case of (D)PoS and identification of trusted nodes in the WoT approach.

Appendix B Cost of Attack on Blockchain Anchoring

Assume an attacker decides to retroactively modify the anchor on a permissionless cryptocurrency blockchain that utilizes proof of work (e.g., the Bitcoin Blockchain). In order to accomplish this, the attacker would need to overwrite the blockchain starting from the block containing the targeted anchor. According to proof-of-work blockchain consensus rules, the attacker would need to produce an alternative chain of blocks with more cumulative proof of work than that created by honest maintainers. Furthermore, the attacker would need to keep his version of blockchain secret until it becomes preferable to the blockchain generated by the honest part of the network. An ordinary majority attack (e.g., censoring all blocks not generated by the attacker) does not change the blockchain history, hence it would not accomplish the attacker's goals.

Note that the attack would have obvious issues:

- Full nodes would retain the version of the blockchain maintained by the honest miners after the attack. Thus, existing blockchain users (including the users of the anchored blockchain) would

be able to verify that the attack took place, significantly diminishing its utility for most use cases (cf. anchoring on printed media, the successful attack on which would necessitate unnoticeably swapping existing printed issues of the medium). The attack itself would require significant amount of preparation, further diminishing the chance to accomplish it covertly

- Because the attack would be highly noticeable, the attacker would be unlikely to gain profit from selling mined cryptocurrency, as its exchange rate would probably significantly drop after the attack. Hence, it is unlikely that the attack would be supported by rational maintainers on the target blockchain

Assume that the attack begins at the moment $t = 0$, and the attacked anchor corresponds to $t = -t_a < 0$ (i.e., t_a is the anchor age). The honest hashrate and the attacker's hashrate are described by functions $g, h : \mathbb{R} \to [0, +\infty)$ respectively, with the condition $\forall t < 0\ h(t) = 0$. We further assume that h and g are both monotonically nondecreasing: $\forall t\ \dot{h}(t) \geq 0, \dot{g}(t) \geq 0$.

The initial cumulative difficulty handicap of the attacker's chain is $\delta \stackrel{\text{def}}{=} \int_{-t_a}^{0} g(t)\,dt$. The attack ends at the moment τ such that the cumulative difficulty of the attacker's chain reaches that of the honest network, i.e.,

$$\int_0^\tau h(t)\,dt = \delta + \int_0^\tau g(t)\,dt. \tag{1}$$

The attack costs

$$J(h, \tau) = R + Ch(\tau) + O\int_0^\tau h(t)\,dt \to \min \tag{2}$$

consist of three factors:

- R, measured in \$, are inelastic capital expenses on the production of hashing equipment
- C, measured in \$/(GH/s), are elastic capital expenses of developing, producing and deploying a unit of hashing equipment
- O, measured in \$/GH, are operating expenses of maintaining a unit of hashing equipment

Naturally, R, C and O are positive.

Observe that the integral part of (2) can be simplified using (1), resulting in a one-sided optimal control problem for the control h:

$$\begin{gathered} J = R + O\delta + Ch(\tau) + O\int_0^\tau g(t)\,dt \to \min_{\tau,h} \quad \text{s.t.} \\ \int_0^\tau h(t)\,dt = \delta + \int_0^\tau g(t)\,dt; \\ h(0) = 0; \quad \forall t \in (0, \tau) \quad \dot{h}(t) \geq 0. \end{gathered} \tag{3}$$

Rather than trying to solve this problem in the generic case (which can be accomplished numerically), we examine a simple partial case.

Assumption 5. $h(t) = h^*$ is constant on the interval $(0, \tau]$ (i.e., the attacker starts the attack after an initial equipment procurement and does not increase the amount of the equipment *during* the attack).

The transition to the constant attacker's hashrate may be justified by the following observation.

Statement 1. The constant attacker's hashrate $h(t) = h^*$ is optimal in (3) for any fixed τ.

Proof.

Assumption 5 leads to the constraint (1) simplified as

$$h^*\tau = \delta + \int_0^\tau g(t)\,dt,$$

thus yielding

$$J(h^*, \tau) = R + O\delta + C\delta/\tau + (O + C/\tau)\int_0^\tau g(t)\,dt, \tag{4}$$

which is now dependent only on τ and not on h^*.

As h is nondecreasing,

$$h(\tau) \equiv \frac{1}{\tau}\int_0^\tau h(\tau)\,dt \geq \frac{1}{\tau}\int_0^\tau h(t)\,dt = \frac{\delta}{\tau} + \frac{1}{\tau}\int_0^\tau g(t)\,dt. \tag{5}$$

Replacing $h(\tau)$ in (3) per (5) yields a lower bound estimate $J(h, \tau) \geq J(h^*, \tau)$ with h^* understood as a constant function. $J(h, \tau) = J(h^*, \tau)$ holds iff $h = h^*$. ∎

Minimizing J in (4) for τ, we obtain

$$\frac{\partial J}{\partial \tau} = \left(O + \frac{C}{\tau}\right) g(\tau) - \frac{C}{\tau^2}\left(\delta + \int_0^\tau g(t)\,dt\right) = 0. \tag{6}$$

Statement 2. If $g(t)$ is continuous, Equation (6) has the only solution on the interval $\tau \in (0, +\infty)$.

Proof.

$$\lim_{\tau\to+0}\frac{\partial J(\tau)}{\partial \tau} = Og(0) + \lim_{\tau\to+0}\left(\frac{Cg(0)}{\tau} - \frac{C\delta}{\tau^2}\right) = -\infty.$$

As $g(t)$ is a nondecreasing function,

$$\frac{\partial J(\tau)}{\partial \tau} \geq \left(O + \frac{C}{\tau}\right) g(\tau) - \frac{C}{\tau^2}\left(\delta + \int_0^\tau g(\tau)\,dt\right) = Og(\tau) - \frac{C\delta}{\tau^2} > 0 \text{ with } \tau \to +\infty.$$

Hence, $\partial J/\partial \tau$ has different signs on the ends of the explored interval, and since it is a continuous function, there is at least one point τ, at which $\partial J(\tau)/\partial \tau = 0$.

Next, observe that (6) has the same solutions on $\tau \in (0, +\infty)$ as the equation

$$\tau^2\frac{\partial J}{\partial \tau} \equiv O\tau^2 g(\tau) + C\tau g(\tau) - C\delta - C\int_0^\tau g(t)\,dt = 0. \tag{7}$$

Differentiate the left part of (7) for τ:

$$\frac{\partial}{\partial \tau}\left(\tau^2\frac{\partial J}{\partial \tau}\right) = \left(O\tau^2 + C\tau\right)\frac{\partial g(\tau)}{\partial \tau} + 2O\tau g(\tau) > 0 \quad \forall \tau \geq 0,$$

because $\partial g/\partial \tau \geq 0$, and $g(\tau) > 0$. Hence, the left part of (7) monotonically increases meaning that (7) and (6) may have no more than a single solution. This completes the proof. ∎

Consider the simplest instantiation of the honest hashrate function $g(t)$: constant $g(t) = g_0$ for all t; in this case, the initial handicap of the attacker's chain $\delta = g_0 t_a$. Equation (6) is simplified as

$$\left(O + \frac{C}{\tau}\right) g_0 - \frac{C(\delta + g_0\tau)}{\tau^2} = 0,$$

from which the optimal attack duration $\tau^* = \sqrt{C\delta/Og_0}$, and the optimal expenses

$$J^* = \underbrace{R + O\delta + Cg_0}_{J_0} + \underbrace{2\sqrt{O\delta \cdot Cg_0}}_{J_{const}}.$$

A part of expenses J_0 can be viewed as the costs spent on anchor security by the honest miners by the time $t = 0$, and J_{const} is the additional penalty. A much-desired property is that J_{const} can be comparable to J_0, i.e., costs spent on securing the anchor are significantly less than the costs to attack it (Fig. 5).

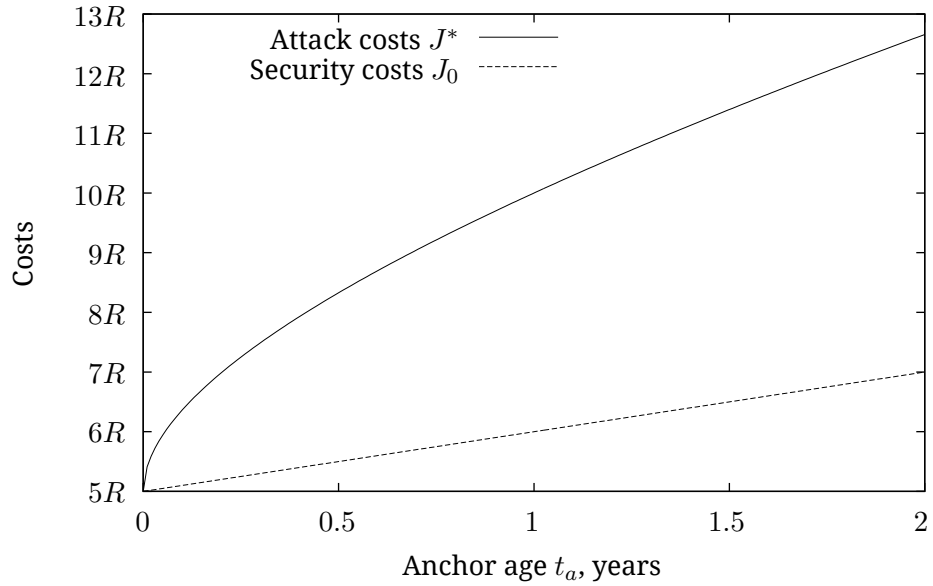

Figure 5: Costs to attack anchoring on a blockchain with the static honest hashrate g_0 depending on the anchor age t_a. It is assumed that the cost factors in (2) $R = Og_0 \cdot 1$ year $= Cg_0/4$, which is by our estimations close to the current distribution of the factors for the Bitcoin Blockchain.

References

[1] *Visa Europe* (2015). Why 2015 was the year for payments
URL: `http://vision.visaeurope.com/why-2015-was-the-year-for-payments/`

[2] *Pete Rizzo* (2015). Hands on with Linq, Nasdaq's private markets blockchain project. In: CoinDesk
URL: `http://www.coindesk.com/hands-on-with-linq-nasdaqs-private-markets-blockchain-project/`

[3] *Pete Rizzo* (2016). DTCC to use digital asset tech for blockchain post-trade trial. In: CoinDesk
URL: `http://www.coindesk.com/dtcc-use-digital-asset-tech-blockchain-post-trade-trial/`

[4] *Pete Rizzo* (2016). KPMG: Blockchain could be 'antidote' to high cost of regulation. In: CoinDesk
URL: http://www.coindesk.com/kpmg-blockchain-antidote-regulatory-costs/

[5] *Pete Rizzo* (2015). Deloitte explores blockchain tech for client auditing. In: CoinDesk
URL: http://www.coindesk.com/deloitte-blockchain-auditing-consulting/

[6] *Pete Rizzo* (2016). PwC director: blockchain impact could create winners and losers. In: CoinDesk
URL: http://www.coindesk.com/pwc-fintech-director-disruptive-blockchain-could-create-winners-and-losers/

[7] *Laura Shin* (2016). Republic of Georgia to pilot land titling on blockchain with economist Hernando de Soto, BitFury. In: Forbes
URL: http://www.forbes.com/sites/laurashin/2016/04/21/republic-of-georgia-to-pilot-land-titling-on-blockchain-with-economist-hernando-de-soto-bitfury/#73ea77666550

[8] *Stan Higgins* (2015). Blockchain startup raises $2 million for intellectual property solution. In: CoinDesk
URL: http://www.coindesk.com/blockchain-startup-2-million-intellectual-property/

[9] *Grace Caffyn* (2015). Everledger brings blockchain tech to fight against diamond theft. In: CoinDesk
URL: http://www.coindesk.com/everledger-blockchain-tech-fight-diamond-theft/

[10] *Yessi Bello Perez* (2015). How Provenance is channeling the blockchain for social good. In: CoinDesk
URL: http://www.coindesk.com/provenance-channeling-blockchain-social-good/

[11] *Mett Levine* (2015). Blockchain for banks probably can't hurt. In: Bloomberg View
URL: http://www.bloombergview.com/articles/2015-09-01/blockchain-for-banks-probably-can-t-hurt

[12] *Satoshi Nakamoto* (2008). Bitcoin: a peer-to-peer electronic cash system
URL: https://bitcoin.org/bitcoin.pdf

[13] *Marshall Pease, Robert Shostak, Leslie Lamport* (1980). Reaching agreement in the presence of faults. In: Journal of the Association for Computing Machinery, vol. 27 (2), pp. 228–234.
doi:10.1145/322186.322188

[14] *Leslie Lamport, Robert Shostak, Marshall Pease* (1982). The Byzantine generals problem. In: ACM Transactions on Programming Languages and Systems, vol. 4 (3), pp. 382–401.
doi:10.1145/357172.357176

[15] *Muguel Castro, Barbara Liskov* (1999). Practical Byzantine fault tolerance. In: Proc. 3rd Symposium on Operating Systems Design and Implementation, OSDI '99, pp. 173–186.

[16] *Josh Benaloh, Michael de Mare* (1994). One-way accumulators: a decentralized alternative to digital signatures. In: Proc. Advances in Cryptology – EUROCRYPT '93, LNCS vol. 765, pp. 274–285
doi:10.1007/3-540-48285-7_24

[17] *Stuart Haber, W. Scott Stornetta* (1990). How to time-stamp a digital document. In: Proc. Advances in Cryptology – EUROCRYPT '90, LNCS vol. 537, pp. 437–455
doi:10.1007/3-540-38424-3_32

[18] *David Chaum, Amos Fiat, Moni Naor* (1990). Untraceable electronic cash. In: Proc. Advances in Cryptology – EUROCRYPT '88, LNCS vol. 409, pp. 319–327
doi:10.1007/0-387-34799-2_25

[19] *Ahto Buldas, Helger Lipmaa, Berry Schoenmakers* (2000). Optimally efficient accountable time-stamping. In: Proc. 3rd Intl. Workshop on Practice and Theory in Public Key Cryptosystems – PKC 2000, LNCS vol. 1751, pp. 293–305
doi:10.1007/978-3-540-46588-1_20

[20] Non-repudiation. In: English Wikipedia
URL: https://en.wikipedia.org/wiki/Non-repudiation

[21] Public key infrastructure. In: English Wikipedia
URL: https://en.wikipedia.org/wiki/Public_key_infrastructure

[22] *Christopher Allen* (2015). The path to self-sovereign identity
URL: https://github.com/ChristopherA/self-sovereign-identity/blob/master/ThePathToSelf-SovereignIdentity.md

[23] Verifiable claims working group frequently asked questions (editor's draft)
URL: http://w3c.github.io/webpayments-ig/VCTF/charter/faq.html

[24] *Andrew Poelstra* (2015). On stake and consensus
URL: https://download.wpsoftware.net/bitcoin/pos.pdf

[25] *Vitalik Buterin* (2014). Proof of stake: How I learned to love weak subjectivity. In: Ethereum Blog
URL: https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/

[26] *Vitalik Buterin* (2014). Slasher: a punitive proof-of-stake algorithm. In: Ethereum Blog
URL: https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/

[27] *Jae Kwon* (2015). Tendermint: consensus without mining
URL: http://tendermint.com/docs/tendermint.pdf

[28] *Vlad Zamfir* (2015). Introducing Casper "the friendly ghost". In: Ethereum Blog
URL: https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/

[29] Open Blockchain. IBM developerWorks
URL: https://developer.ibm.com/open/ibm-open-blockchain/

[30] *Maurice Herlihy, Mark Moir* (2016). Enhancing accountability and trust in distributed ledgers
arXiv:1606.07490

[31] *Byung-Gon Chun, Petros Maniatis, Scott Shenker, John Kubiatowicz* (2007). Attested append-only memory: making adversaries stick to their word. In: Proc. 21st ACM SIGOPS Symposium on Operating Systems Principles – SOSP '07, vol. 41 (6), pp. 189–204.
doi:10.1145/1294261.1294280

[32] *Manuel Araoz*. What is proof of existence?
URL: https://proofofexistence.com/about

[33] *Paul Snow, Brian Deery, Jack Lu et al.* (2014). Factom: business processes secured by immutable audit trails on the blockchain
URL: https://github.com/FactomProject/FactomDocs/raw/master/Factom_Whitepaper.pdf

[34] Tierion
URL: https://tierion.com/

[35] Openchain documentation
URL: https://docs.openchain.org/en/latest/general/overview.html

[36] *Bitpay* (2015). ChainDB: a peer-to-peer database system
URL: https://bitpay.com/chaindb.pdf

[37] *Bitfury Group* (2015). Public versus private blockchains. Part 1: permissioned blockchains
URL: http://bitfury.com/content/5-white-papers-research/public-vs-private-pt1-1.pdf

[38] *Andrew Miller, Yu Xia, Kyle Croman et al.* (2016). The honey badger of BFT protocols. Cryptology ePrint Archive, Report 2016/199
URL: https://eprint.iacr.org/2016/199

[39] Finite-state machine. In: English Wikipedia
URL: https://en.wikipedia.org/wiki/Finite-state_machine

[40] *Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, Carl Landwehr* (2004). Basic concepts and taxonomy of dependable and secure computing. In: IEEE Transactions on Dependable and Secure Computing, vol. 1 (1), pp. 11–33.
doi:10.1109/TDSC.2004.2

[41] Audit. In: English Wikipedia
URL: https://en.wikipedia.org/wiki/Audit

[42] *Steve Randy Waldman* (2015). Soylent blockchains
URL: http://www.interfluidity.com/uploads/2015/02/soylent-blockchains-to-share.pdf

[43] *Kate Gibson* (2015). In rare admission of guilt, Wall St. banks say they rigged markets. In: CBS Moneywatch
URL: http://www.cbsnews.com/news/in-rare-admission-of-guilt-wall-st-banks-admit-they-rigged-markets/

[44] *Katy Burne* (2015). Big banks agree to settle swaps lawsuit. In: The Wall Street Journal
URL: http://www.wsj.com/articles/banks-wall-street-groups-agree-to-settle-credit-swaps-antitrust-case-1441988741

[45] *The Financial Crisis Inquiry Commission* (2011). The financial crisis inquiry report
URL: https://www.gpo.gov/fdsys/pkg/GPO-FCIC/pdf/GPO-FCIC.pdf

[46] Libor scandal. In: English Wikipedia
URL: https://en.wikipedia.org/wiki/Libor_scandal

[47] *Bitfury Group* (2016). Digital assets on public blockchains
URL: http://bitfury.com/content/5-white-papers-research/bitfury-digital_assets_on_public_blockchains-1.pdf

[48] *Diego Ongaro, John Ousterhout* (2014). In search of an understandable consensus algorithm. In: Proc. 2014 USENIX Annual Technical Conference – USENIX ATC '14, pp. 305–319
URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[49] *Leslie Lamport* (1998). The part-time parliament. In: ACM Transactions on Computer Systems, vol. 16 (2), pp. 133–169
doi:10.1145/279227.279229

[50] *Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson et al.* (2005). Fault-scalable Byzantine fault-tolerant services. In: Proc. 20th ACM symposium on Operating systems principles – SOSP '05, pp.59–74
doi:10.1145/1095809.1095817

[51] *Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin et al.* (2009). Zyzzyva: speculative Byzantine fault tolerance. In: ACM Transactions on Computer Systems, vol. 27 (4), Article no. 7
doi:10.1145/1658357.1658358

[52] *Pierre-Louis Aublin, Sonia Ben Mokhtar, Vivien Quema* (2013). RBFT: Redundant Byzantine fault tolerance. In: Proc. IEEE 33rd International Conference on Distributed Computing Systems – ICDCS '13, pp. 297-306
doi:10.1109/ICDCS.2013.53

[53] *Allen Clement, Edmund Wong, Lorenzo Alvisi et al.* (2009). Making Byzantine fault tolerant systems tolerate Byzantine faults. In: Proc. 6th USENIX Symposium on Networked Systems Design and Implementation – NSDI '09
URL: https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/clement/clement.pdf

[54] *Michael Fischer, Nancy Lynch, Michael Paterson* (1985). Impossibility of distributed consensus with one faulty process. In: Journal of the ACM, vol. 32 (2), pp. 374–382
doi:10.1145/3149.214121

[55] *Cynthia Dwork, Nancy Lynch, Larry Stockmeyer* (1988). Consensus in the presence of partial synchrony. In: Journal of the ACM, vol. 35 (2), pp. 288–323
doi:10.1145/42282.42283

[56] *Christian Cachin, Klaus Kursawe, Victor Shoup* (2005). Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography. In: J. of Cryptology, vol. 18 (3), pp. 219–246
doi:10.1007/s00145-005-0318-0

[57] *Tushar Deepak Chandra, Sam Toueg* (1996). Unreliable failure detectors for reliable distributed systems. In: Journal of the ACM, vol. 43 (2), pp. 225–267
doi:10.1145/226643.226647

[58] *Gabriel Bracha, Sam Toueg* (1985). Asynchronous consensus and broadcast protocols. In: Journal of the ACM, vol. 32 (4), pp. 824–840
doi:10.1145/4221.214134

[59] *Mike Hearn, Matt Corallo* (2014). Connection Bloom filtering (BIP 37)
URL: https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki

[60] *Arthur Gervais, Ghassan O. Karame, Damian Gruber, Srdjan Capkun* (2014). On the privacy provisions of Bloom filters in lightweight Bitcoin clients. Cryptology ePrint Archive, Report 2014/763
URL: https://eprint.iacr.org/2014/763

[61] *Ralph C. Merkle* (1988). A digital signature based on a conventional encryption function. In: Proc. Advances in Cryptology – CRYPTO '87, LNCS vol. 297, pp. 369–378
doi:10.1007/3-540-48184-2_32

[62] *Gavin Wood* (2014). Ethereum: a secure decentralised generalised transaction ledger. Homestead revision
URL: http://gavwood.com/Paper.pdf

[63] Content delivery network. In: English Wikipedia
URL: https://en.wikipedia.org/wiki/Content_delivery_network

[64] Bitnodes: global Bitcoin nodes distribution (retrieved on Sep 07, 2016)
URL: https://bitnodes.21.co/

[65] *Vitalik Buterin* (2014). Bitcoin multisig wallet: the future of Bitcoin
URL: https://bitcoinmagazine.com/articles/multisig-future-bitcoin-1394686504

[66] *Maurice Herlihy, Jeannette Wing* (1990). Linearizability: a correctness condition for concurrent objects. In: ACM Transactions on Programming Languages and Systems, vol. 12 (3), pp. 463–492
doi:10.1145/78969.78972

[67] *Ahto Buldas, Peeter Laud, Helger Lipmaa, Jan Villemson* (1998). Time-stamping with binary linking schemes. In: Proc. Advances in Cryptology – CRYPTO '98, LNCS vol. 1462, pp. 486–501
doi:10.1007/BFb0055749

[68] Trusted timestamping. In: English Wikipedia
URL: https://en.wikipedia.org/wiki/Trusted_timestamping

[69] *Tobias Gondrom, Ralf Brandner, Ulrich Pordesch* (2015). Evidence record syntax (RFC 4998)
doi:10.17487/rfc4998

[70] *Jim Finkle* (2016). Bangladesh Bank hackers compromised SWIFT software, warning issued. In: Reuters Technology News
URL: http://www.reuters.com/article/us-usa-nyfed-bangladesh-malware-exclusiv-idUSKCN0XM0DR

[71] *Meni Rosenfeld* (2012). Analysis of hashrate-based double-spending
URL: https://bitcoil.co.il/Doublespend.pdf

[72] Standard transactions: In: Bitcoin Developer Guide
URL: https://bitcoin.org/en/developer-guide#standard-transactions

[73] src/policy/policy.cpp. In: Bitcoin Core GitHub repository (retrieved on Sep 07, 2016)
URL: https://github.com/bitcoin/bitcoin/blob/master/src/policy/policy.cpp

[74] *Steven Goldfeder, Joseph Bonneau, Edward W. Felten et al.* (2014). Securing bitcoin wallets via threshold signatures
URL: http://www.cs.princeton.edu/~stevenag/bitcoin_threshold_signatures.pdf

[75] *Steven Goldfeder* (2014). Threshold signatures and bitcoin wallet security: a menu of options. In: Freedom to Tinker Blog
URL: https://freedom-to-tinker.com/blog/stevenag/threshold-signatures-and-bitcoin-wallet-security-a-menu-of-options/

[76] *Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, Tal Rabin* (2001). Robust threshold DSS signatures. In: Information and Computation, vol. 164 (1), pp. 54–84
doi:10.1006/inco.2000.2881

[77] *Adi Shamir* (1979). How to share a secret. In: Communications of the ACM, vol. 22 (11), pp. 612–613
doi:10.1145/359168.359176

[78] *Amir Herzberg, Markus Jakobsson, Stanisław Jarecki et al.* (1997). Proactive public key and signature systems. In: Proc. 4th ACM Conf. on Computer and Communications Security – CCS '97, pp. 100–110
doi:10.1145/266420.266442

[79] Hardware security module. In: English Wikipedia
URL: https://en.wikipedia.org/wiki/Hardware_security_module

[80] Confidential transactions. In: The Elements Project
URL: https://www.elementsproject.org/elements/confidential-transactions/

[81] *Eli Ben-Sasson, Alessandro Chiesa, Christina Garman et al.* (2014). Zerocash: decentralized anonymous payments from Bitcoin (extended version). Cryptology ePrint Archive, Report 2014/349
URL: https://eprint.iacr.org/2014/349

[82] *Nicolas van Saberhagen* (2013). CryptoNote v 2.0
URL: https://cryptonote.org/whitepaper.pdf

[83] *Pieter Wuille* (2013). Hierarchical deterministic wallets (BIP 32)
URL: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

[84] *Gavin Andresen, Mike Hearn* (2013). Payment protocol (BIP 70)
URL: https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki

[85] *Ahmed Kosba, Andrew Miller, Elaine Shi et al.* (2015). Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. Cryptology ePrint Archive, Report 2015/675
URL: https://eprint.iacr.org/2015/675

[86] *Zooko Wilcox-O'Hearn* (2014). Proving your bitcoin reserves. In: Zak Wilcox's Namespace
URL: https://iwilcox.me.uk/2014/proving-bitcoin-reserves

[87] *Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau et al.* (2015). Provisions: privacy-preserving proofs of solvency for bitcoin exchanges. Cryptology ePrint Archive, Report 2015/1008
URL: https://eprint.iacr.org/2015/1008

[88] *Bitfury Group* (2015). Proof of stake versus proof of work
URL: http://bitfury.com/content/5-white-papers-research/pos-vs-pow-1.0.2.pdf

[89] *Seth Gilbert, Nancy Lynch* (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services In: ACM SIGACT News, vol. 33 (2), pp. 51–59
doi:10.1145/564585.564601