A PoW-less Bitcoin with Certified Byzantine Consensus

An Unorthodox Foundation for a Central Bank Digital Currency (CBDC)?

Marco Benedetti, Francesco De Sclavis, Marco Favorito, Giuseppe Galano,
Sara Giammusso, Antonio Muci, Matteo Nardelli*

Technical Report CFC.CRYPTO.CS/2022/1
Applied Research Team (ART) - IT Department - Bank of Italy†

ABSTRACT

Distributed Ledger Technologies (DLTs), when managed by a few trusted validators, require most but not all of the machinery available in public DLTs. In this work, we explore one possible way to profit from this state of affairs. We devise a combination of a modified Practical Byzantine Fault Tolerant (PBFT) protocol and a revised Flexible Round-Optimized Schnorr Threshold Signatures (FROST) scheme, and then we inject the resulting proof-of-authority consensus algorithm into Bitcoin (chosen for the reliability, openness, and liveliness it brings in), replacing its PoW machinery. The combined protocol may operate as a modern, safe foundation for digital payment systems and Central Bank Digital Currencies (CBDC).

1 INTRODUCTION

In the past few years, the announcement of cryptoasset-inspired "stablecoins" by private companies (e.g., Diem by Meta[1]) and the prospective issuance by central banks of fiat currencies in digital format for retail use, or Central Bank Digital Currencies (CBDCs) [9, 21, 24, 60], coupled with the unabated diffusion of decentralised blockchain-based digital assets, have reignited the interest in alternative consensus protocols for blockchains, especially those amenable to permissioned settings, on which we focus here.

The trusted nodes in such arrangements may be interpreted as multiple computational nodes managed by the same actor. In this case, we would envision a service provider that centrally operates a modern, blockchain-based, programmable and transactional engine, exhibiting high-availability and strong fault tolerance.

That is a compelling motivation already, but there are far more interesting use cases, truer to the nature of a distributed ledger: Each node (or small group of nodes) may be managed by independent actors, far removed from each other, either geographically or legally. These actors may share a common interest that would be perfectly served, technically, by a distributed ledger with no centralization point: Everyone enjoys equal rights, duties, capabilities; no one "owns the system"; everyone contributes to its resilience. These actors may even reside in different jurisdictions, and conform to different laws, albeit under some shared regulatory framework [2].

These motivations hold for most DLTs, from Hyperledger[3] to Corda[4], and for both payment and non-payment domains.

However, in this work, we specifically focus on Bitcoin and on *digital payments*. We ask ourselves: Is the unorthodox notion of "*Precisely Bitcoin, minus its traditional consensus algorithm, plus trusted third parties, in a permissioned setting*" a technically consistent one?

Let's start by reviewing Bitcoin and its consensus protocol.

1.1 Consensus in Bitcoin

Bitcoin [52] is a peer-to-peer monetary network launched in 2009: It implements a digital asset which does not rely on trusted third parties to guarantee its *scarcity* or to prevent *double spending*. Instead of trusted parties, it employs a decentralized *consensus protocol* among anonymous participants, based on Proof-of-Work (PoW). In PoW, votes (on what the next state of the system is) can be cast by just anyone, but each vote implies a substantial consumption of real-world resources (e.g., time, hardware, energy) to solve certain hard problems related to the inversion of cryptographically strong hash functions, whose solution is required to make the vote valid. This "costly postage stamp" of sort is key to prevent *sybil attacks* in open, anonymous settings: Without it, malicious actors could compromise the consensus by surreptitiously creating at no (or little) cost a number of pseudonymous identities, through which a majority of apparently distinct votes, hence the system, are controlled[5].

Many other similar crypto-assets have emerged over time, such as Ethereum [18] and Monero [4], to name a few. Each of them brings in additional features (e.g., Ethereum adds a Turing-complete programming language), but for the most part they resolved to confront their large, decentralised, anonymous user base by inheriting the PoW idea made popular—and proven effective—by Bitcoin.

There is ample space for debate on whether the power-hungry PoW is inherently the best conceivable solution for large, decentralized, anonymous blockchains; perhaps the very same properties can be obtained by computationally lighter[6] means? For sure, in a permissioned setting with few validators, PoW alone is not going to work: The resources sufficient to outcompete a small network are likely within reach for any motivated and sponsored attacker.

So, the question becomes: How to disentangle Bitcoin from PoW, and by what means is the resulting blockchain supposed to keep exhibiting tolerance to faults, attacks, and censorship attempts?

*Email address of the authors are in the form [firstname].[lastname]@bancaditalia.it, except for giuseppe.galano2@bancaditalia.it.

†The views expressed in this paper are those of the authors and do not necessarily reflect those of the Bank of Italy.

[1] https://www.diem.com/en-us/

[2] One hypothetical case would be a CBDC whose high availability and fault/attack tolerance rest upon a distributed platform operated *cooperatively*—in a profound sense—by several Central Banks in a given monetary area.

[3] https://www.hyperledger.org/

[4] https://www.corda.net/

[5] To subjugate a PoW system, an attacker would have to outcompete the rest of the network in terms of available resources and willingness to sacrifice them. This so called 51% attack has been widely studied in the literature [48, 62, 80].

[6] Power-efficient alternatives for reaching a consensus in large peer-to-peer networks of anonymous participants have been explored. For example, in a Proof-of-Stake (PoS) system [43] such as Algorand [39], the voting rights are not proportional to the consumption of real-world resources but to the staking of virtual resources themselves.

1.2 All of Bitcoin but PoW

Our goal is to inherit *verbatim* all the algorithms, data structures, cryptography, and software from Bitcoin, getting rid of merely the ingredients (e.g., PoW) that are unnecessary or undesirable in a *permissioned setting managed by a few trusted actors*.

If it is possible to identify a small set of actors that end-users trust to cooperatively guarantee scarcity and to prevent any double spending, then a Bitcoin-like blockchain can be grown via, e.g., a consensus based on Proof-of-Authority (PoA), wherein validators are known in advance and trusted by all network stakeholders. They are "just" required to prove their identity by cryptographically strong means before appending any new blocks to the chain.

Of course, high availability and tolerance to faults and to malicious behaviors of some nodes (things that used to be guaranteed by decentralization and PoW) remain mandatory even in our smaller, permissioned, distributed setting. It turns out these properties can be recovered by borrowing and modifying existing consensus and signature algorithms from the literature. The difficult thing is to inject such new algorithmic ingredients into Bitcoin while striving to maximize the reuse of its existing technical apparatus.

That's in essence the idea we develop in this paper. As usual, the devil is in the details, and it takes a lot of work to devise a "*permissioned Bitcoin*" exhibiting all the features we call for.

The untold premise here is that there are enough virtues and strengths to be inherited from the Bitcoin codebase, even after PoW is excised, to be worth the trouble. Is this true?

1.3 Public strengths, in private

Bitcoin is the one platform to combine the following 5 features.

(1) **Focus**. The original focal point of Bitcoin—digital payments—aligns with our prospective use cases more than other DLTs focused on, e.g., programmability, tokenisation, decentralized finance (DeFi).
(2) **Reliability**. Bitcoin has been extensively studied by the Academy and has been open to attacks on the Internet for over 13 years; it has had arguably more scrutiny than any other DLT. So, we reuse a wealth of battle-tested software machinery in a permissioned setting—a good starting point for, e.g, mission critical payment applications.
(3) **Extensibility**. The scripting language of Bitcoin is a good trade-off between programmability and safety. It has seen the largest ever deployment of any decentralized programmable machine, while keeping a small surface of attack compared to Turing-complete DLTs. Fortunately, its programmability is strong enough to implement most (all?) second layer constructions that are relevant in, e.g., the payment domain.
(4) **Openness**. Most software/protocols in Bitcoin are open. There is a rich ecosystem of competences, software, and services around its blockchain, from open source communities and organizations—both large and small. This implies a level-playing field open to, e.g., small Fintechs, which is important in potential pro-competitive public shared platforms.
(5) **Liveliness**. The communities developing Bitcoin are large, lively, and diverse; a profusion of new features/updates are always in the work. And, while the *network effect* sustaining the Bitcoin stack is one of the largest in existence as far as DLTs are concerned, there is *no private organisation in key roles*: A nice attribute for applications such as CBDCs.

The most part of all these features are PoW-independent. There is much to reuse in a permissioned, payment-oriented blockchain.

1.4 Solution overview

We target settings where there are some (from 5 to 20) privileged and trusted nodes in charge of accepting and validating all transactions and growing an otherwise Bitcoin-like blockchain.

From the literature on distributed consensus algorithms, we select one Byzantine fault tolerant (BFT) scheme that suits our needs: the PBFT (*Practical Byzantine Fault Tolerant*) protocol. We analyzed the existing open implementations of this protocol, but none was fit for the purpose of being injected straight into Bitcoin; so we reimplemented the protocol in a high-level logical framework meant to mimic the original declarative specification [23] as close as possible, and we modified its features slightly to support the distributed update of an append-only, blockchain-aware state machine.

The acquisition of block signatures from a valid quorum of trusted nodes is performed by a custom variant of FROST [45] (*Flexible Round-Optimized Schnorr Threshold Signatures*). This is possible thanks to the recent introduction, within Bitcoin, of Schnorr signatures [64], via the Taproot soft-fork [78]

Unfortunately, a simple juxtaposition of PBFT and FROST is not enough: Issues arise during distributed signature, because the consequences of the possible reluctance of (faulty or malicious) nodes to sign blocks is something PBFT is unaware of and FROST alone is unable to deal with. In addition, both protocols have parameters on which the properties of the emerging system depend. These parameters have to be chosen to play nicely with each other. We work out a solution to this intermixing problem in the next sections.

1.5 Scope of this work

We focus only on the foundational issue of the consensus and signing protocol at the "on-ledger" layer, i.e., on designing and developing a working PoA-based BFT algorithm meant to sustain the growth of an *extremely* Bitcoin-like blockchain.

True to the nature of Bitcoin, we want our solution be as open and as subject to scrutiny and reuse as possible: A prototype implementation of the entire system is available in open source[7].

But, there is a lot more to any real-world Bitcoin-derived permissioned payment system than is dealt with in this paper. Two of the most pressing issues of the retail payment systems we are interested in supporting with our construction are *scalability* (to the order of tens of thousands of transactions per second) and *privacy* (of the payment metadata with respect to third parties and payment processors). Both properties are out of scope for the present work. They are meant to be achieved by *programming the core distributed machine* we devise into guaranteeing them. The so called second layer protocols we will leverage to obtain this result are ongoing research and are discussed as future work (cfr. Section 7).

1.6 Structure and contribution of this paper

Our major contribution is to show how 3 fairly sophisticated protocols, coming from different communities—namely Bitcoin, PBFT,

[7] Link to the OS repository will be available here soon.

and FROST—can be altered to make them interlock neatly with one another. From such pooling, a permissioned Bitcoin-like DLT emerges, with strong fault tolerance and confidential aggregation of signatures. A detailed specification and an open-source implementation are contributed.

To the best of our knowledge, this is the first time algorithms such as PBFT and FROST are combined and adapted to a PoA setting that retains the wealth of technical tools accrued by Bitcoin.

A high-level overview of the architecture and interlinks of the system is provided in Section 2, together with a discussion of the desired requirements for such a PoA-based mining federation.

Then, one section is devoted to each of the three protocols, with the aim of describing the adaptations and enhancements they go through in order to smoothly engage with each other:

- Section 3 reviews and characterizes the changes we apply to Bitcoin, and show how much (or how little) our flavor of the protocol differs from the public one;
- Section 4 describes the features and implementation of a BFT consensus algorithm modified for use by a mining network in cooperation with a suited signature algorithm; we move from a reference protocol (PBFT) and modify it to fit our architecture and requirements;
- Section 5 presents 5-FBFT and 3-FBFT, two novel FROST-derived protocols to aggregate a quorum of block signatures into a single one (with advantages in terms of confidentiality and space efficiency) during PBFT consensus rounds.

Finally, we review the related literature in the areas of permissioned DLTs, custom BFT designs, and threshold signature schemes, comparing our work with the state of the art (Section 6) and we list a few extensions and future developments that would allow our new architecture to be used in real-world scenarios (Section 7).

2 SYSTEM MODEL AND REQUIREMENTS

In this section, we describe at a high-level the system architecture and the desired properties it is expected to fulfill.

2.1 High-level architecture

Our architecture is composed of two networks with different properties: a *participant network* and a *mining network*—see Figure 1.

Participant network. The participant network is composed of a set of *participant nodes*, noted $P_0, P_1, \ldots, P_{M-1}$, which run the modified Bitcoin protocol from Section 3. Each participant node receives, validates, and stores a copy of our Bitcoin-like blockchain. Participants form a spontaneous, *permission-less* peer-to-peer network, without a predefined topology or size. The bidirectional communication channels among them (dotted lines in Figure 1) are used to propagate blocks and messages via gossiping, just like in Bitcoin.

Mining Node. The rounded rectangles inside the grey area are our mining[8] nodes, or "miners" (there are four of them in Figure 1). Each miner $M_i = (B_i, C_i)$ is composed of a *bridging node* B_i and a *consensus node* C_i, running on the same host and connected by synchronous *bridging channels* (see next). Each miner is controlled

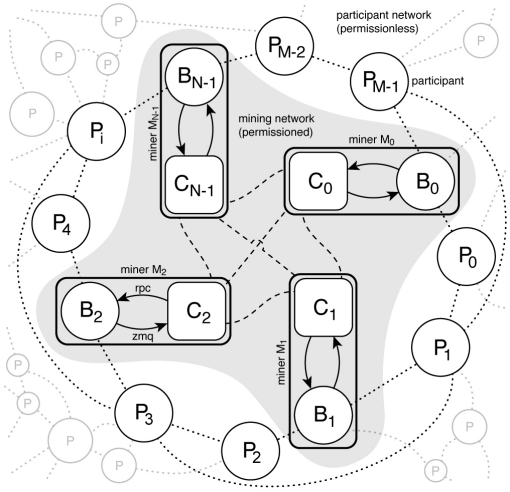

Figure 1: A permission-less participant network and a permissioned mining network with $N = 4$ nodes.

and operated by one member of a federation of N trusted, identifiable actors, called *validators*. While the bridging node of each miner runs the same protocol as any other participant node (in particular, it collects transactions to be validated from participants and propagates new valid blocks to others as soon as it gets aware of them), the consensus node runs the modified version of PBFT and FROST described in Section 4 and Section 5, respectively.

Mining network. Miners are connected to each other in a full mesh topology; the resulting *permissioned* network is called the *mining network* (everything within the gray area in Figure 1). This is a peer-to-peer network too: Mining nodes are equivalent to each other, with no one playing any special role. The communication links among mining nodes (dashed lines) are bidirectional channels used to exchange authenticated[9] messages required by the PoA consensus and signing protocols (as per Section 4 and Section 5).

Bridging channels. In between the bridging node and the consensus node of each miner, there are 2 host-local, synchronous channels (solid, oriented arcs in Figure 1): They act as *bridging channels* between the Bitcoin realm and the PBFT/FROST one. One uses an RPC protocol, whereby the consensus node takes the initiative to interact with the corresponding bridging component to, e.g., obtain a candidate template block, sign a block, ask to broadcast a block (see Section 4.4 and all the self-loop messages in Figure 2). The other bridging channel adopts a publish/subscribe model over the ZMQ protocol: The consensus node subscribes to the bridging node in order to get the mining federation notified of occurrences of new signed blocks, which act as PBFT checkpoints (Section 4.5). These protocols and modes of interaction were chosen because the corresponding endpoints are already exposed by the standard Bitcoin core APIs offered by B_i, i.e., for maximum Bitcoin reuse.

Roles and coupling. The mining network is a service provider: Its goal is to collect transactions from participants, reach a robust

[8]The terms "miner" and "mining" are etymologically incongruous in the context of our architecture, where trusted nodes do not operate to *mining* any reward; however, we stick to them for historic reasons and for their close association with Bitcoin.

[9]Each mining node has a private key that is used to sign messages; the corresponding public key are used by other mining nodes to verify the origin and authenticity of each consensus message they receive.

consensus on which to include in new valid blocks, and then deliver signed blocks back to participants, thus growing their shared and trusted blockchain. Thank to the properties of the consensus and signing protocols, the mining network appears to the participants as a single mining entity. Dually, the network of participants acts as a unique, virtual client submitting transactions to the blockchain managed by the mining nodes, and expecting such transactions to be timely validated with cryptographic strength. The client (network of participants) is reliably connected to the server (network of miners) via a few standard Bitcoin-like (P_j, B_k) channels freely established by at least some participant P_j towards one or more of the bridging nodes B_k; these channels are indistinguishable from regular peer-to-peer channels within the permissionless network.

Failures. We assume a Byzantine failure model where F_B mining nodes can fail[10] arbitrarily. In addition to Byzantine failures, we assume that further F_C nodes may crash. The consensus algorithm we employ rely on synchrony to provide liveness, but not to provide safety. In order to avoid the FLP impossibility result on consensus [30], we assume that (dashed) communication channels are weakly synchronous: Message delays among correct miners do not grow too fast and indefinitely. This is considered a pretty faithful model of a real production system network, where faults are eventually repaired. As long as the network is in a failed state, it may fail to deliver messages, delay/duplicate them, or deliver them out of order. We assume an adversary that can coordinate faulty nodes, but cannot subvert the cryptographic primitives in use.

2.2 Requirements

We call for our PoA consensus to exhibit the following properties.

R1 **Correctness**. All blocks need to have a content that is valid according to the rules of the blockchain application, e.g., valid hash of the previous block, valid signatures, non-negative balances. Each block must transition the blockchain from one valid state to another, and the mining network must prevent invalid blocks to be produced. This property is also called *validity* in the context of blockchain networks, and *consistency* in the context of database systems (ACID properties).

R2 **Safety**. In the particular case of a blockchain, safety forbids chain forks, i.e., different but valid versions of the most recent blocks of the same blockchain. We use the Common Prefix property definition from [33, 34], instantiated with parameter $k \in \mathbb{N}$, which states that for any pair of honest nodes P_i, P_j, adopting the chains C_i, C_j at heights $h_i \leq h_j$, it holds that the chain resulting from the pruning of the k rightmost blocks of C_i is a prefix of C_j. In our model, safety is not probabilistic[11] and this requires the Common Prefix Property to hold $\forall k \geq 0$, which is equivalent to the absence of forks. This property is also called *agreement* in the context of distributed systems, and *finality* in the context of payment systems[12] (e.g., Libra [12]).

R3 **Liveness**. The mining network must produce new blocks at each round. We use the Chain Growth property definition from [33, 34], with parameters $\tau \in \mathbb{R}$ and $r \in \mathbb{N}$: for any honest party P that has a chain C, it holds that after any r consecutive rounds it adopts a chain that is at least $\tau \cdot r$ blocks longer than C.

R4 **Calmness**. The pace at which blocks are produced is upper-bounded. A stable block time helps participants to form expectations on their computational requirements (e.g., disk space, bandwidth). If a Byzantine miner creates blocks at a rate significantly higher than τ, it can cause participants to run out of resources, effectively carrying out a denial-of-service attack.

R5 **Confidentiality**. This property requires that, at each round carried on *with no faulty/Byzantine miners*, the mining network does not reveal information to the participants, other than the new block and its solution[13]. In other words, the participants should not learn anything other than the fact that a provably valid new block was added. Other information, such as the miner who forged the block, the active miners/signers who approved it, the total number of miners/signers in the federation, should be kept hidden from the participants. This property makes targeted attacks against the mining network harder.

R2-R3 have been already defined and studied in the context of blockchains [34], whereas R4-R5 are somewhat peculiar to ours.

2.3 Optional requirements

There are at least two other secondary requirements that could be taken in account, but that are out of scope for this work. One is the *scalability* of the BFT solution in terms of number of messages exchanged among the mining nodes at each round, as a function of the number of mining nodes[14]. In this paper, and considering our target use cases and the typical size of the mining network, this notion of scalability is not of paramount importance. Another property is that of *fairness*, i.e., preventing a Byzantine miner from delaying/censoring certain transactions from the network (can only happen under specific conditions). The measures to include to eliminate this specific kind of unfairness are known, but they unnecessarily complicate the baseline construction we are interested in presenting here, and are left as future work (cfr. Section 7).

3 AMENDING THE BITCOIN PROTOCOL

This section describes the main changes to the Bitcoin protocol.

Block validity. Our blocks are valid only if they include the solution to a specific "block challenge", as in the Bitcoin Signet [3]. It could be expressed either as a Bitcoin script via `OP_CHECKMULTISIG` (as in a Bitcoin Signet), or as an aggregated public key (see Section 5). The challenge is distributed among participant nodes at setup time,

[10] A mining node fails if and only if the mining component fails, or the participant component fail, or any bridging link fails.

[11] In Bitcoin and other "*Nakamoto consensus protocols*", transaction finality is a *probabilistic* concept: The mere appearance of a transaction in a new block does not offer solid guarantee that it will not be (possibly maliciously) reverted. There is still the possibility that an attacker builds an alternative blockchain that "reorganizes" the latest (i.e., the longest) accepted version of the ledger to subvert or revert any transaction. However, once a transaction is included in a block, the probability that an attacker with finite resources succeeds in building such an alternate reality drops exponentially with the number of blocks appended to the chain.

[12] The deterministic finality of transactions we have regained is a feature of no little bearing on the legal status of the digital assets that may be transacted on the platform.

[13] This property is impossible to guarantee in rounds where Byzantine failures happen in our architecture, since the network configuration is known to each participant, and a Byzantine node can choose to reveal extra information to the outside world.

[14] In the literature, two properties related to scalability have been considered: *linearity* and *responsiveness* (e.g., see [81]). Linearity (a condition considered optimal in the context of BFT) guarantees that creating new blocks incurs only a linear communication cost, even when leaders rotate. Responsiveness means that the leader has no built-in delay steps and advances as soon as it collects responses from validators.

and set in the configuration file of each node they run. For each block, the data that satisfy the challenge, called "block solution", is stored in a special `OP_RETURN` output of the coinbase transaction, so it is automatically propagated to the peer-to-peer network via the standard mechanisms used for blocks and transactions.

As we shall see (Section 5), in our case the solution represents an "aggregated signature", i.e., a set of signatures from a valid but opaque quorum of trusted signers in the mining network who agreed (Section 4) to append that specific block at a specific height.

To accommodate for our safety requirement (R2) in the context of a PBFT-derived consensus[15] (see Section 4), it is necessary to alter the Signet validation rule to exclude the block solution from the computation of the Merkle root for transactions[16].

In addition, it is necessary to include the PoW fields `nBits` and `nNonce` in the block signature. This is because we want to prevent a (malicious) miner with SHA-256 hashing power to be able to cause a fork by tweaking them: If the PoW fields were not signed, then a miner could change `nNonce` to imply more work, and its block would replace the legitimate one by the Bitcoin rules[17].

Block mining. The steps for creating blocks become as follows:

(1) Upon request by the consensus node of a miner, the corresponding bridging node assembles a block template, i.e., it selects a set of transactions from the mempool, and adds a coinbase transaction with an empty block solution. At the end of this step, *the block Merkle root is finalized*.
(2) The miner *grinds* the block, i.e., it finds a nonce that fulfills a trivial PoW-like challenge, which is purposely included for backward compatibility with the original Bitcoin protocol. At the end of this step, *the block hash is finalized*.
(3) A quorum of miners signs the block, i.e., it appends a valid block solution. At the end of this step, *the transactions are finalized*: the Merkle root and block hashes are unaffected.

Block interval. The interval between (1 MB) blocks is fixed to one minute, instead of the self-stabilizing[18] 10-minute interval of the public Bitcoin network. We could increase the rate or the block size to improve the throughput, but this would limit the valuable ability of all network participants to stay in sync, especially those with low bandwidth. At any rate, raw transactional scalability for end-users is meant to be achieved off-chain (see Section 8).

Block subsidy. Differently from the public Bitcoin, we allow any value for the block subsidy, i.e., for the freshly minted coin that is output by any coinbase transaction. This is done by removing the block subsidy checks from the code base of participant nodes[19].

Coinbase maturity. In Bitcoin, coinbase transaction outputs can only be spent after a certain number of new blocks (100 in the public network). This number is called *coinbase maturity*. Its existence is a countermeasure to rule out certain inconsistencies and disservices to end users in case of blockchain reorganizations[20]. In our settings, no forks occur as per our safety requirement, and no reorganization can happen, so the coinbase maturity is safely set to 0.

A suggestive (if insubstantial) appraisal of how much of the public Bitcoin code we retain is obtained by measuring the syntactic scope of our changes; they amount to (a) the replacement of ≈ 20 lines of code and (b) the addition of $\approx$ 500 lines. This patch is sufficient to glue the latest Bitcoin core[21] to the implementation of the protocols described in the rest of this paper, and to tie up all loose ends. Such custom protocols add another $\approx 8k$ lines of code, i.e., less than 2% of the current Bitcoin core size.

4 ORDERING BLOCKS WITH PBFT

Our liveness (R3) and calmness (R4) requirements call for the mining network to produce a block per minute (cfr. Section 2). We assume that our technological infrastructure is up to the task in terms of computing power and network bandwidth/latency, and we design our block creation process around a Practical Byzantine Fault Tolerant (PBFT) protocol [22]. The preference for PBFT, in lieu of other BFT algorithms, is motivated by the following consideration: PBFT sacrifices linear communication (i.e., the number of messages exchanged is not linear in the cluster size) in return for a simpler implementation; however, our cluster is small enough, by design, to make the superlinear communication complexity a minor drawback, whereas simplicity in the implementation helps a lot the cohabitation with the complex Bitcoin protocol.

4.1 PBFT in a nutshell

PBFT is a state machine replication algorithm: it relies on a set of *replicas* to maintain a service state and to implement a set of *operations* onto it. The replicas move through a succession of configurations called *views*, which are numbered consecutively. In a view, one replica is the *primary* and the others are *backups*. *View changes* are carried out when it appears that the primary has failed.

[15] In presence of delays or failures, it is impossible to know in advance the specific quorum of validators that will agree to sign a block at a specific height, as it would imply foreseeing if and how failures will occur.

[16] Otherwise, different sets of signers for the same block (see Note 15) could lead to different, valid block solutions. This would result in different coinbase transactions and block hashes, which in turn would lead to chain forks, eventually triggering a chain reorganization. By our safety requirement (R2), this possibility is to be ruled out.

[17] An alternative solution to this problem would be similar to the one used with the block signatures, i.e., to exclude the PoW fields from the block hash altogether. However, this approach would call for a much more invasive modification to the existing Bitcoin code base, and this is contrary to our goal of maximizing its reuse.

[18] The Bitcoin network employs a closed-loop feedback to stabilize the 10-minute interval in the presence of a fluctuating and generally unknown global hashing power: A decreasing (increasing) average mining time is taken as a proxy of an increased (decreased) global hash capacity, hence the mining difficulty is proportionally increased (decreased) to get back to 10 minutes. This entire difficulty adjustment mechanism is not relevant in a permissioned PoA network, so it is disabled.

[19] It is worth recalling that the block subsidy plays a very specific role in the public Bitcoin, i.e., to provide incentive to miners, who pay for the resources they invest in mining (and then accrue some revenue) by the market value hopefully recognized to the very subsidy tokens they mine. Especially in early times (when transaction fees play almost no role as there are few users transacting) this was an essential bootstrapping mechanism, subject to a well known ballistic halving procedure, still underway, meant to smoothly transition the system from a "self-referential bet" into a full-fledged and largely used service with a fee-based sustainability model. All these motivation, cost-recovering, and bootstrapping phenomena are *non-existent in our permissioned setting*: They are replaced by external incentives, agreements, and monetary flows specific to the supported use case and to the specific mining federation. The block subsidy maintain one last, key role though: It is the one technical mechanism through which the asset issuer(s) in the mining federation inject freshly minted tokens/money into the blockchain, subject to, once again, *external* policies and arrangements.

[20] Without a large maturity value, the coinbase transactions of orphan blocks would become invalid in case of a reorganization, together with any subsequent transactions that depend on their outputs, causing severe inconveniences to end users.

[21] To fully profit from features (4) and (5) in Section 1.3, our open source patch and code are always kept up to date with the latest release of the public code. At the time of writing, we are "permissioning" the Bitcoin core version v23.0.

Service operations are invoked by *clients*, which send *requests* to the primary. Then a three-phases protocols begins, that allows replicas to agree among them on the order in which requests are to be executed: (i) in the *pre-prepare* phase, the primary assigns a sequence number to the request and multicasts it request to the backups; (ii) in the *prepare* phase, the backups agree on the sequence number proposed by the primary; (iii) in the *commit* phase, the replica confirm that an agreement on the request and its sequence number has been reached by a *PBFT quorum* of replicas. Then, each replica executes the operation and replies to the client. The client waits for $F_B + 1$ replies from different replicas with the same result, where F_B is the maximum number of replicas that may be Byzantine.

In the following sections we describe a specialized version of PBFT that deals with block selection and block signing in presence of Byzantine and crash failures. The algorithm contains additional blockchain-specific steps, but also some simplifications, based upon the following considerations: (i) Our state machine exposes only a single operation, that is the appending of a new block; (ii) Our mining network has a single abstract client that is the network of participants; (iii) There are no parallel mining requests, since each miner expects to mine only the next block (the one after the next block cannot be mined if the next block is not mined yet); (iv) The *checkpoint* and state propagation mechanisms can rely on the block propagation process already present in the participants network.

4.2 Quorum size

Suppose we want to tolerate, at most, F_B Byzantine failures, and F_C crash failures in the mining network. We replicate the service across $N = 3F_B + 2F_C + 1$ nodes, and we choose a PBFT quorum of $Q = 2F_B + F_C + 1$. For example, suppose the blockchain operators want to keep mining blocks after the private key of one node is compromised (Byzantine failure) and at the same time there is ongoing maintenance on another node (crash failure). It is $F_B = 1$ and $F_C = 1$, so we need $N = 6$ nodes and a PBFT quorum of $Q = 4$. This network configuration employs the minimum N and Q guaranteeing that (i) two different quorums always intersect in at least one non-Byzantine mining node, i.e., $Q = \lceil (N + F_B + 1)/2 \rceil$, which is a necessary condition for Safety (P2) and (ii) a quorum of non-faulty miners can be reached also if $F_B + F_C$ nodes fail, i.e., $N - Q = F_B + F_C$, which is a necessary condition for Liveness (R3)—provided delays among correct nodes do not grow indefinitely.

In addition to the PBFT quorum, we need to consider the *Signet quorum*, i.e., the number of signatures which are required for a block to be valid before the network of participants. Assuming that each node controls a single Signet key, then a safe Signet quorum also depends on F_B. The minimum Signet quorum is $F_B + 1$, which corresponds to the number of agreeing replies a PBFT client needs to collect from replicas. This quorum can be used if nodes sign the block after they have received a PBFT quorum of commit messages.

In this paper, we assume a more restrictive condition: We require the Signet quorum to be equal to the PBFT quorum, and we refer generically to them as quorum. This allows us to design a PBFT algorithm that does not require additional rounds for block signature (see Section 4.4), because the signing operation can be safely anticipated to the commit phase. Also, we can use the participant network gossiping for checkpoint propagation (see Section 4.5).

4.3 The client

In our setting, the PBFT client is just a single virtual entity: the participant network. This implies a set of changes (simplifications, for the most part) to the duties and operations of our PBFT client.

The original PBFT relies on a client to send request messages to the primary (and, if needed, to other replicas), in order to invoke operations on the replicated state machine. Once the operation is executed, replicas send back the result directly to the client. Clients are assumed to be trusted, since the PBFT safety property is insufficient to protect against faulty clients. The original PBFT request message is $\langle REQUEST, o, t, c \rangle_{\sigma_c}$, where o is the state machine operation, t is a request timestamp, and c is a client identifier.

In our protocol, we instantiate the client and its requests as follows. The state machine has a single operation, that appends a new block. The content of the block (e.g., the set of transactions) represents a form of non-determinism and its value will be selected by the primary in the pre-prepare phase. For these reasons, we omit the operation in the request message, since it is always an implicit "append", and the client identifier, since it is unique.

The participants network expects a new valid block to be mined every τ seconds on average. Being T_0 an initial timestamp that is known in advance by all the replicas, we calculate the request message timestamps by adding T_0 to multiples of the desired block time. In lights of these considerations, all replicas know in advance all valid request messages, that have the form $\langle REQUEST, T_0 + n\tau \rangle$: $n \in \mathbb{N}^+$, where n is the block height for all blocks (except the genesis one, which is fixed and not mined via the consensus algorithm).

Non-faulty replicas will generate a request for block n when it is expected to be mined, i.e., when their local clock value is at the nominal timestamp of the block minus a delta (that allows replicas to carry out the PBFT round, and depends on communication delays).

The original PBFT reply message is $\langle REPLY, v, t, c, i, r, \rangle_{\sigma_c}$, where v is the current view number, t is the timestamp of the corresponding request, i is the replica identifier, and r is the result of the operation. In our protocol, a block mined at a given height is the result of the append operation of the corresponding request. Given that valid blocks are broadcast to the participants network once a quorum of signatures by replicas is achieved, we omit the reply messages from our specialized protocol, and replace it with the Bitcoin block propagation mechanism to the participants network.

4.4 Normal operation (no faulty primary)

We describe the PBFT normal case operations of the mining network, with a special focus on the modifications that allow miners to create, sign and propagate a new valid block. Figure 2 shows the normal case operations with a non-primary faulty replica.

The process starts with a request to append a new block, self-generated by the primary. The operation is non-deterministic, as its result depends on the content of the block to append. As suggested in [22], we make sure that the primary selects such content independently, and concatenates it with the associated request.

In the pre-prepare phase, the primary M_0 gathers a set of transactions from its mempool and forms a template for the next block to be appended. The primary assigns a sequence number n to the block, that corresponds to the height at which the block is expected

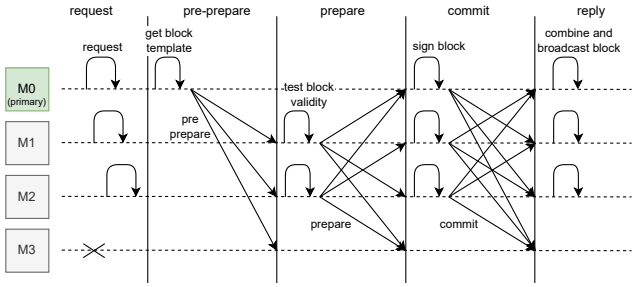

Figure 2: Mining network normal operation with $N = 4$ nodes, M_0 as primary, and M_3 as faulty backup.

to be added, then includes the block in the pre-prepare message and broadcasts it to backups for signature[22].

A backup (i.e., M_1, M_2, or M_3 in figure) accepts a pre-prepare message if the request timestamp is valid, not too far in the future with respect to the local clock of the replica, and the block template is also valid. The block template validity is checked by the participant node which is co-located with the replica, and prevents invalid blocks from being signed. If the request or the block is invalid, the replica ignores the pre-prepare message. This may happen if, e.g., the primary or the replica are not synchronized to the blockchain network or to Coordinated Universal Time. In addition to the above checks, a replica checks the original PBFT conditions, to prevent different blocks from being signed at the same height. If a backup accepts the pre-prepare message, then it enters the prepare phase and broadcasts the prepare message to all other replicas.

A replica (primary or backup) accepts a prepare message if all the standard conditions [22] are true; no additional checks are present at this stage. A block is said to be *prepared* at replica i in view v and height n iff replica i has received a pre-prepare proposal to append block in view v at height n from the primary, and $Q - 1$ backups have acknowledged the proposal. The pre-prepare and prepare phases of the algorithm guarantee that non-faulty replicas agree on the block height within a view. When replicas reach an agreement on a block and its height, they proceed to the commit phase, in which they actually sign the prepared block.

In the commit phase, a replica (primary or backup) signs a block, includes the signature in the commit message, and broadcasts the message to other replicas. The addition of the signature to the commit message is a difference with respect to the original PBFT. A replica accepts a commit message if it contains a valid signature for its corresponding block, and the PBFT conditions are met.

A block is *locally committed* at replica i in view v at height n iff replica i has accepted a PBFT quorum of commit messages, possibly including its own. The quorum of commit messages contains a valid Signet quorum of block signatures, and is sufficient to assemble a valid block solution, which can be accepted by the participants network. In other words, when a quorum of commit messages is collected, a replica can concatenate the signatures from the commit messages, and append the combined signatures to the block solution, in the coinbase transaction; if the replica is also synchronized with the blockchain, it can subsequently broadcast the new block to the participant network. It is possible that different replicas broadcast the same block (in terms of transactions), at the same height, but with different sets of signatures, because they received commit messages from different replicas. This does not matter for the safety guarantee of the blockchain, and does not cause a blockchain fork or a reorganization, because block signatures, even if placed in the coinbase transaction, are excluded from the calculation of the root of the transactions Merkle tree, as described in Section 3.

The PBFT invariants guarantee that if a new block is locally committed at a non-faulty replica, then it is propagated to the network and will eventually be received by all non-faulty participants.

[22] A faulty primary might send the same, invalid block to all replicas. Therefore, replicas must be able to assess—independently and deterministically—whether the value is correct (and what to do if it is not) based on their current value of the state.

4.5 Checkpoints

The PBFT checkpoint is a mechanism used to discard messages from the log. It is used to guarantee the correctness of the service state that is synchronised among replicas, even when messages have been already discarded from the log. The original PBFT checkpoint message is $\langle CHECKPOINT, n, val_i, lastrep_i, lastrept_i, i\rangle_{\sigma_i}$, where n is the sequence number, val_i is the state machine value, $lastrep_i$ and $lastrept_i$ are the content of the reply message and the timestamp of respective request, i is a client identifier, and the message is signed by the replica. A checkpoint is said to be *stable* when its content has been signed by a quorum of different replicas, and the signatures are the proof of correctness of the checkpoint.

In our protocol, we rely on the Bitcoin block propagation mechanism of the underlying participants network for the checkpoint propagation among replicas. The checkpoint value n is the block height, while the val_i is the whole blockchain content, that we summarize as its tip (or best block hash), and the reply message is the blockchain block at height n. Each block appended to the blockchain is by design signed by a quorum of replicas, and the block solution represents the proof of correctness for the checkpoint. This implies that all checkpoints are stable by design, i.e., they come with a proof that they are the result of the execution of requests by a quorum of replicas. For this reason, there is no equivalent for an unstable checkpoint in our algorithm: Each replica maintains a single copy of the service state, i.e., the one resulting from the last stable checkpoint, or the last block. Moreover, the propagation of checkpoints happens via the participants peer-to-peer Bitcoin network. Each replica, upon receiving a new signed block at height n from the participants, including the ones propagated by the replica itself, does the following: (i) discards all pre-prepare, prepare, and commit messages with sequence number less than or equal to n; and (ii) updates the PBFT parameters (namely, low and high watermarks) to signal the replica is ready to append a new block at height $n + 1$.

4.6 View change

The view-change protocol provides Liveness (R3) by allowing the mining network to make progress when the primary fails. In our protocol, as in PBFT, view changes are triggered by timeouts that prevent backups from waiting indefinitely for new blocks. Our view change is not different from the PBFT one, with the exception that request messages are self-generated by replicas.

Each replica self-generates requests for each block, just before these blocks are expected. A backup starts a timer when it self-generates a request and the timer is not already running. A backup is waiting for a request if it self-generated a request and has not

executed it. It stops the timer when it is no longer waiting to execute the request, but restarts it if at that point it is waiting to execute some other request. If the timer of backup i expires in view v, the backup starts a view change to move the system to view $v + 1$.

The same considerations on view-change timeout values that are necessary to achieve Liveness in the original PBFT apply.

4.7 Realized properties of the mining network

Correctness is guaranteed, because a non-faulty replica does not propose (if primary) or accept (if backup) a block that is invalid according to the rules of the participant network.

Safety and *Liveness* hold under the same assumption as PBFT [22]. *Calmness* is guaranteed because a correct replica does not send (if primary) or accept (if backup) a pre-prepare message with requests containing an invalid (or too distant in the future) timestamp. *Confidentiality* is *not* achieved, because the network configuration is known to the participants network, and the actual signatures are concatenated and published in the coinbase transaction of each block. Section 5 describes an algorithm that also achieves confidentiality (under the assumption that no Byzantine failures occur).

5 CERTIFIED BYZANTINE CONSENSUS

We address the problem of signing blocks in an aggregated manner. Moving from a threshold signature scheme known as FROST (Section 5.1), we design a novel protocol that combines aggregate signatures with a BFT consensus protocol (Section 5.2).

5.1 The FROST Signature Scheme

We consider a threshold signature scheme with a group of n participants and a predefined threshold value k, with $k < n$ [65]. We rely on the additive property of Schnorr signatures to quickly combine signatures into an aggregated one [64]. The FROST signature scheme requires three main protocols: (i) a *key generation protocol* that creates secret shares for participants as well as public keys for signature verification; (ii) a *commitment protocol* that creates nonce/commitment share pairs for all participants; these commitments allow to prevent known forgery and replay attacks; (iii) a *signature protocol* coordinates the generation of the aggregated signature by signers.

We define these protocols following the FROST specification [45]. Hereafter, each participant M_i has a unique identifier $m_i \in \{1, \ldots, n\}$. Let $\mathbb{G}$ be a group of prime order q, g be a generator of $\mathbb{G}$, and let H_1 and H_2 be cryptographic hash functions mapping to $\mathbb{Z}_q^*$. We denote by $x \leftarrow S$ that x is uniformly randomly selected from set S.

Key Generation. Before signing any block, participants need to define secret and public keys. They share the same ciphersuite, which specifies the underlying prime-order group details (e.g., ristretto255, P-256) and cryptographic hash function (e.g., SHA-512). Each participant runs the FROST KeyGen protocol [45], an enhancement of the Pedersen Distributed Key Generation (DKG) [57], which performs n parallel executions of the Feldman's VSS [29]. Hence, no centralized trusted dealer is needed.

The FROST KeyGen consists of two rounds where every participant exchanges messages with every other participant. We assume that messages use a secure channel and will be eventually delivered. At the end of the protocol, each participant M_i, with $i \in \{1, \ldots, n\}$, owns a secret share s_i, a public verification share $Y_i = g^{s_i}$, and the group's public key Y. The public verification share Y_i allows others to verify the participant signature shares. The group's public key enables the aggregate threshold signature verification. It depends on the set of participants n and the configured threshold k. Note that, independently from the set of signers, if at least k correct signature shares are provided, a valid aggregate signature will be produced, which can be verified using the group's public key.

Commitment. In the commitment protocol, participants generate (secret) nonces for signatures and exchange their public commitments. The latter allow to verify the correct use of nonces. Although nonces will be used in the signing process, they can be computed upfront because they do not depend on the specific message to sign. Importantly, a nonce must not be used multiple times, otherwise the secret share is compromised. Following the FROST preprocess protocol, each participant M_i, $i \in \{1, \ldots, n\}$, generates π nonce/commitment share pairs. Specifically, M_i first generates the nonce share pair $(d_{ij}, e_{ij}) \leftarrow \mathbb{Z}_q^* \times \mathbb{Z}_q^*$, for $j \in \{1, \ldots, \pi\}$, and then derives the commitment shares $(D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}})$. M_i saves his nonces $\langle (d_{ij}, e_{ij}) \rangle_{j=1}^{\pi}$ for later use, and publishes his list of commitments $\langle (i, D_{ij}, E_{ij}) \rangle_{j=1}^{\pi}$. When participants generate a single nonce/commitment share pair at a time, i.e., $\pi = 1$, we simplify the notation and refer to the public commitments of M_i as (D_i, E_i).

Aggregated Signature. The aggregate signature protocol works in two rounds. First, each participant generates his signature share. Then, all participants' shares are combined to obtain the final signature. Let S be the set of participants to the signing process; the cardinality of S is α, with $k \leq \alpha \leq n$. Let L be the list of α participants' commitments, i.e., $L = \langle (l, D_l, E_l) \rangle_{l=1}^{\alpha}$. When M_i receives the message to sign m, he can use his secret share s_i and L to compute his signature share z_i, which can then be sent to all other participants. Formally, M_i computes the set of binding values $\rho_l = H_1(l, m, L)$, $l \in \{1, \ldots, \alpha\}$, and derives the group commitment $R = \prod_{l=1}^{\alpha} D_l \cdot (E_l)^{\rho_l}$ and the challenge $c = H_2(R, Y, m)$. Then, M_i computes his signature share z_i on m by computing $z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$, using (d_i, e_i) corresponding to $(D_i, E_i) \in L$, and S to determine the i-th Lagrange coefficient $\lambda_i = \prod_{j=1, j \neq i}^{\alpha} \frac{m_j}{m_j - m_i}$.[23] Since nonces cannot be used multiple times, M_i deletes the $((d_i, D_i), (e_i, E_i))$ pair from his local storage. To conclude the first round, M_i sends his signature share z_i to every other participant M_l, with $l \in S$.

The second round starts when M_i receives all other signature shares z_l; he will verify and aggregate the signatures. For verification, for each $l \in S$, M_i derives ρ_l, R_l, and c (as defined before). Then, he checks if the equality $g^{z_l} = R_l \cdot Y_l^{c \cdot \lambda_l}$ holds for each signing share z_l. If the verification is successful, M_i aggregates the signature shares locally by computing $z = \sum_{i \in S} z_i$. Each participant saves the aggregate signature $\sigma = (R, z)$ along with m.

[23] In the KeyGen protocol, M_i defines a $t-1$ degree polynomial $f_i(x)$, with k randomly sampled coefficients $(a_{i0}, \ldots, a_{i(t-1)}) \leftarrow \mathbb{Z}_q$. The group's secret will result to be $a_0 = \sum_{i=1}^{n} a_{i0}$. M_i securely sends to each other participant M_l the point $(m_l, f_i(m_l))$. Each participant then calculates his private signing share $s_i = \sum_{l=1}^{n} f_l(m_i)$, public verification share $Y_i = g^{s_i}$, and group's public key $Y = \prod_{j=1}^{n} g^{a_{j0}}$. The Lagrange interpolation that reconstructs the secret $a_0 = \sum_j a_{j0}$ takes place in the exponent (under the Decisional Diffie-Hellman assumption, it is not feasible to extract a_0).

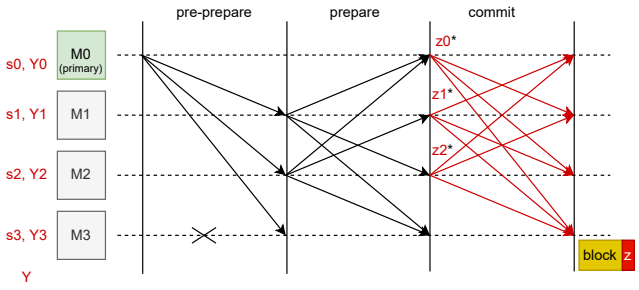

Figure 3: Normal operation (no faulty primary) of 3-FBFT. Replicas exchange a set of potential signature shares in the commit phase. When a replica receives these signature shares combines them to reconstruct the aggregate signature for the block in a decentralized manner.

5.2 FROSTing PBFT

The challenge of combining FROST with PBFT arises from the presence of Byzantine nodes that may refuse to sign blocks and from the weakly synchronous communication model. A naïve integration of FROST would not work in practice due to the difficulty of defining upfront the quorum of participants S that will collaborate to compute the aggregated signature z. Integrating FROST with BFT requires to review the commitment and aggregate signature protocols (Section 5.1). We need rules for exchanging (D_i, E_i) pairs and the set of signers S, two critical information for determining the binding values ρ_i, the challenge c, and Lagrange coefficients λ_i that reconstruct the secret used to sign messages.

Two protocols are presented, called 3-FBFT and 5-FBFT. The first represents the trivial way to use FROST in an asynchronous setting, and optimizes the communication by shipping potential signature shares with the PBFT commit messages. Unfortunately, this solution requires an exponential number of potential commitments and signature shares and can be used only when the set of signers is small. The second introduces new rounds at the end of PBFT consensus, which allow to minimize the information exchanged to produce a valid block signature. To guarantee the protocol liveness in presence of Byzantine nodes that may refuse to sign, this protocol uses the idea proposed by ROAST [61]. Both protocols enhance nodes with the cryptographic primitives presented in Section 5.1. Either way, at the end of the protocol, each participant has a block m with the related signature $\sigma = (R, z)$. The integrity of m can be validated using the traditional Schnorr verification algorithm [64].

5.2.1 3-FBFT *(3-Phase Frosted-BFT).* This protocol aims to optimize the number of rounds by allowing replicas to directly exchange multiple, potential signature shares. The replicas themselves will figure out autonomously how to combine the signature shares to compute the correct aggregated signature for the block. Unlike 5-FBFT, this protocol decouples the commitment protocol from the PBFT consensus. Each replica generates nonces and exchanges information with other replicas to let them derive the public commitments needed to sign blocks in 3-FBFT. This commitment protocol uses a hierarchical deterministic key derivation [31, 77], which allows determining (D_i, E_i) for each participant i starting from an extended public commitment of i and locally available information. From here on, we assume each participant knows its own nonces and the public commitments of all replicas. The number of commitments to generate considers that, for each block request, a participant will exchange γ signature shares with all other participants. The γ parameter is defined observing that determining the set of block signers S in advance is not possible, due to the presence of Byzantine replicas. However, computing an aggregated signature requires only k signature shares, with the threshold equal to the Signet quorum size. In this case, the Signet quorum size equals the PBFT quorum size Q, because signature aggregation takes place in the commit phase that, in turn, waits for Q messages as indicated in Section 4.2. Each participant M_i determines all k-combinations of n known participants $\mathcal{S} = \langle S^{(j)} \rangle_{j=1}^{\gamma}$, where $\gamma = \binom{n}{k}$ is the number of combinations. Since the number of signature shares γ grows almost as $O(2^n)$, this protocol is feasible only when n is small, e.g., in a small mining network. The nonces and public commitments are indexed by the participant identifier, the PBFT sequence number (i.e., block height), and the identifier of a specific combination of signers (i.e., $j \in \{1, \ldots, \gamma\}$). In this way, a participant will use a different nonce for each generated signature share. Moreover, each participant can readily retrieve the correct nonce/commitment pairs for each combination of signers in a non-interactive manner.

In 3-FBFT, the signature aggregation protocol is executed at once with the PBFT consensus protocol. The 3-FBFT protocol has indeed the same number of phases as the traditional PBFT. Commit messages are extended to transfer also signature shares. As usual, we assume that each replica of the consensus protocol participates in the signing process. Figure 3 shows 3-FBFT, highlighting the modified messages in red. The pre-prepare and prepare phases of 3-FBFT exactly match the phases from Section 4. When the commit phase starts, M_i determines the list of public commitments for every combination of k participants in $\mathcal{S}$ that include M_i. M_i computes a signature share $z_i^{(j)}$ for each of these combinations, obtaining $\mathcal{Z}_i = \langle z_i^{(j)} \rangle_{j=1}^{\gamma}$, which is then sent to all other replicas via the commit message. When the quorum in the commit phase is reached, each replica has all the information to aggregate signature shares of others and create a valid block certificate. Each participant uses the received commit messages to identify a set of k participants, whose index is j^*, and accordingly extract the signature share $z_i^{(j^*)}$ from $\mathcal{Z}_i$ for each $i \in S^{(j^*)}$. So, he retrieves all k participants' public commitments, indexed by j^*, and reconstructs R (as per Section 5.1). To complete 3-FBFT, participants aggregate the signature shares, $z = \sum_i z_i^{(j^*)}$, and save $\sigma = (R, z)$ as the certificate of block m.

5.2.2 5-FBFT *(5-Phase Frosted-BFT).* This variant introduces two main changes to the BFT protocol (Section 4). First, it blends the commitment protocol and the aggregate signature protocol into the (normal case) PBFT. Second, it extends PBFT with additional rounds to guarantee liveness in case Byzantine nodes play the role of signers. We assume that each replica of the consensus protocol is a participant in the signing process. Replicas collaborate to apply the threshold signature on the block agreed upon consensus. As per Figure 4, 5-FBFT introduces new rounds in PBFT, namely commitment-share and sign. The commitment-share and

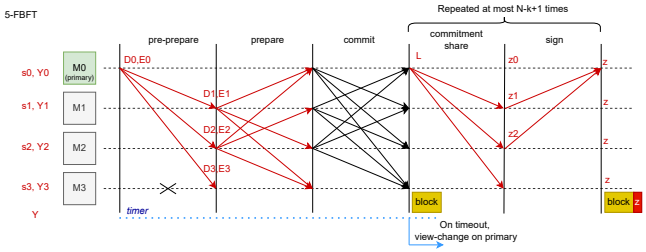

Figure 4: Normal operation (no faulty primary) of 5-FBFT. Replicas exchange their public nonce commitments in the prepare phase. In the commitment share phase, the primary defines the L parameter enabling the creation of partial signatures. The latter are exchanged in the sign phase, enabling replicas to aggregate the signature in a decentralized manner.

sign rounds are executed according to ROAST (RObust ASynchronous Threshold signatures), which wraps the FROST protocol as described in [61]. When a replica i receives the pre-prepare message from primary, it runs the *commitment protocol* to randomly determine the nonce/commitment share pairs $((d_i, e_i), (D_i, E_i))$. Public commitments (D_i, E_i) are piggybacked to the prepare message and exchanged with other replicas leveraging the PBFT protocol. The primary holds a list of responsive signers, among which the set of candidate signers will be defined. Replicas that send the public commitments in their prepare message are considered as part of the initial set of responsive signers, and will be considered as candidate for the a signing session in the commitment-share phase.

After replicas exchange the commit messages, a set of *commitment-share* phases of 5-FBFT takes place. The primary defines a set of signers S among active replicas; the selection policies for defining the set S follow the rules of ROAST. S has cardinality α with $\alpha = k$, including the primary itself, and we require $k = F_B + 1$ (see Section 4.2); in such a configuration, at least a honest signer is included, thus preventing forgery of aggregate signatures over invalid blocks. Even though the primary might exclude nodes suspected to be potentially unresponsive, malicious nodes are unknown a priori, so they could still be included in S. For this reason the primary can initiate multiple and concurrent commitment-share sessions, maintaining a set of responsive signers, i.e., signers that have responded to all previous signing requests. As soon as there are at least k responsive signers in the set, the primary will initiate a new commitment-share session. When the primary determines S, he creates and sends the list of signers' public commitment $L = \langle (l, D_l, E_l) \rangle_{l \in S}$ to other replicas. Knowing L (and, consequently, S), other replicas can determine all the information to compute the signature share on the block, namely ρ_l, λ_l, and c, for $l \in S$.

The *sign* phase of 5-FBFT allows replicas to run the aggregate signature protocol presented in Section 5.1. They create and exchange with the primary the signature shares z_i, with $i \in \{1, \ldots, \alpha\}$, together with a new public commitment to be possibly used in another commitment-share session. If any signature share z_i is not valid, the primary marks the replica as malicious, so that it will not be included in subsequent commitment-share phases. When the primary receives all other signature shares z_i, with $i \in S$, it can combine them to derive the aggregate signature $z = \sum_i z_i$. The aggregate signature $\sigma = (R, z)$ certifies the block m. When the aggregate signature is correctly defined, the primary broadcasts the block and complete 5-FBFT. As demonstrated in ROAST, the commitment-share sessions will eventually finish, and a non-faulty primary will receive all the signatures, in at most $N - k + 1$ commitment-share sessions, under the hypothesis that the number of possible backup replica failures $F_B + F_C$ is at most $N - k$. The view-change protocol described in Section 4.6 allows to provide liveness also in presence of a faulty primary, which delays (but does not compromise) the ROAST protocol. When a view change is triggered by the block timeout, the (possibly new) primary replica will act as a new semi-trusted coordinator, that will run again the aggregate signature protocol. Note that the view-change cannot change values the quorum has agreed upon, so the block content cannot be updated. It is also worth pointing out that, thanks to the properties of threshold signatures, even if the set of signers S and related parameters change, a valid aggregate signature will be produced. This property follows from Lagrange interpolation: A different set of participants basically provides a different set of points over the same $t - 1$ degree polynomial, where the signature secret lies.

6 RELATED WORK

This work draws on ideas from three research areas: permissioned DLTs, fault tolerant consensus algorithms, and signature schemes.

6.1 Permissioned DLTs

Even if Bitcoin is primarily designed for the public network, there exist previous examples of Bitcoin-like ledger meant for private networks. In particular, Elements[24], whose production deployment—the "Liquid" sidechain [53]—uses a BFT consensus algorithm within a permissioned mining network consisting of cryptocurrency businesses[25]. Elements provides additional features with respect to Bitcoin, including confidential assets [58] and more advanced programming capabilities, which may be further expanded [54].

Elements is the closest piece of previous work in terms of technologies and Bitcoin reuse goals. However, to the best of our knowledge, no public specification for its proprietary BFT approach exists, and the open sourced components[26] do not include its implementation.

The second largest DLT born public and then adapted to permissioned settings is Ethereum [76]. For example, an Ethereum-like ledger designed for private networks is Hyperledger Besu[27], which supports a PoA consensus based on Istambul BFT [51]. An implementation is available in open source[28]. Another Ethereum-like ledger designed for private networks is Concord[29]. This is possibly the closest work to ours, in spirit, but (i) it implements SBFT [41] instead of PBFT as a consensus algorithm (a quality open source implementation exists[30]); (ii) it works with BLS signatures instead

[24] https://blockstream.com/elements
[25] https://help.blockstream.com/hc/en-us/articles/900003013143-What-is-the-Liquid-Federation-
[26] https://github.com/ElementsProject
[27] https://www.hyperledger.org/use/besu
[28] https://github.com/hyperledger/besu/tree/main/consensus/ibft
[29] https://blogs.vmware.com/opensource/2018/08/28/meet-project-concord
[30] https://github.com/vmware/concord-bft

of Schnorr to generate a quorum certificate; and (iii) it has Ethereum instead of Bitcoin as a foundation.

Point (iii) is a profound differentiator, for the reasons we discuss in Section 1.3. In particular, with respect to the "focus", "reliability", and "extensibility" dimensions, it is worth noting that Ethereum exhibits a Turing-complete language as a key feature, and focuses on the development of complex decentralized applications via smart contracts, more than on digital payments. Among the relevant applications of its smart contracts, we find the issuance of cryptocurrency tokens [75], NFTs [27], the creation of financial businesses that do not rely on intermediaries (e.g., decentralized exchanges [6], DeFi applications). Several of these decentralized applications have experienced attacks by cybercriminals [69], not rarely executed by leveraging the subtleties of the scripting language.

There is a host of other relevant permissioned DLTs, whose main difference with respect to our approach is the absence—by design—of any attempt to profit from existing code bases and software from major public blockchains: They are custom DLTs, that often redesign/reimplement much from scratch. Notable examples are:

- *Hyperledger Fabric*[31], a general purpose DLT that enables the development of enterprise applications, not necessarily financial; it is designed particularly for private networks. Among its components, there is a BFT consensus module based on BFT-SMaRT [14], but its development appears to have been stopped[32].
- *Corda*[33], a DLT designed for the financial industry. It has a token-based data model (UTXO) like Bitcoin, and a Turing-complete programming language like Ethereum. Its main applications are the issuing of digital assets, or currencies, payments, and global trade. It has a pluggable consensus mechanism: Its "*notaries*" can run either a crash fault tolerant (CFT) consensus, such as RAFT, or a BFT consensus, like PBFT. Neither the BFT specification nor its implementation are available in the open source repository[34], and apparently no aggregated signature scheme is included.
- *Hyperledger Sawtooth*[35] allows to deploy private DLT networks with a variety of consensus algorithms, including PBFT and Proof of Elapsed Time [25], a recent proposal which uses a trusted execution engine to securely generate a random waiting time and then choose a node with the smallest waiting. Sawtooth has an open source implementation[36], with an incomplete PBFT implementation and no FROST-like signature aggregation.
- *Diem*[37] (formerly Libra) is a novel DLT platform which implements a Turing-complete programming language designed for safe and verifiable transaction-oriented computation. It employs a custom BFT algorithm called DiemBFT [12] based on Hotstuff [81], which has an open source implementation[38].

Finally, there is Hamilton [50], which is close to our approach in terms of use cases: It is a DLT designed to support payments in a private network. It exhibits a token-based data model, and it has an open source implementation[39]. It inherits certain elements from Bitcoin (such as the UTXO model and even the elliptic curve used in all cryptographic primitives). Relevant differences: its ledger is not a blockchain; its consensus protocol is not Byzantine but CFT; its ledger is meant to stay private; its main focus is on obtaining transactional scalability on-ledger.

[31] https://www.hyperledger.org/use/fabric
[32] https://github.com/bft-smart/fabric-orderingservice
[33] https://www.corda.net
[34] https://github.com/corda
[35] https://www.hyperledger.org/use/sawtooth
[36] https://github.com/hyperledger/sawtooth-pbft
[37] https://developers.diem.com/docs/welcome-to-diem
[38] https://github.com/diem/diem/tree/main/consensus
[39] https://github.com/mit-dci/opencbdc-tx

6.2 Byzantine Fault Tolerance

Lamport et al. [47] firstly introduced the problem of a distributed system reaching agreement in the presence of Byzantine failures. Different surveys review the most relevant BFT consensus protocols (e.g., [5, 32, 55, 63, 79]). PBFT by Castro and Liskov [22, 23] is considered the reference solution for practical implementations.

In PBFT, replicas exchange messages using an all-to-all communication pattern, hence PBFT does not scale well; attempts to optimize it exist, along different directions, such as communication pattern (e.g., [41, 81]), leader rotation (e.g., [2, 17, 71, 74]), view-change optimization (e.g., [17, 19]), pipelining (e.g., [19, 81]), and speculative/optimistic execution (e.g., [1, 7, 41, 46, 56, 71]).

The adoption of "collectors" reduces the number of exchanged messages, obtaining a linear communication pattern. A collector is a designated replica (usually the leader) that receives and broadcasts messages to all the other replicas. Based on this idea, SBFT [41] uses a dual-mode protocol with an optimistic fast path (when replicas are mostly in sync) and a fallback slow path (more PBFT-like). This dual-mode protocol increases complexity in favor of performance.

To avoid dual-mode, HotStuff [81] uses a collector in combination with threshold signatures to generate a quorum certificate for each protocol phase. This is the most closely related work to ours. By using a single collector, HotStuff increases the protocol complexity to avoid weakening its robustness, to face cases when, e.g., the collector himself is a Byzantine node.

DiemBFT [71] enhances HotStuff to improve throughput. Using a Pacemaker mechanism, whose design is however not fully specified, DiemBFT synchronizes the consensus phases to simplify the consensus protocol. However, DiemBFT, just like HotStuff, does not aggregate quorum signatures in a fully decentralized manner.

In PBFT, the leader changes only when a problem is detected. As an alternative, the rotating leader strategy [74] proposes to change the leader after every attempt to commit (e.g., [17, 81]). To efficiently solve the leader selection problem, deterministic as well as more sophisticated policies have been proposed (e.g., [2, 71]). We postpone as future work the design of a more sophisticated consensus policy with better fairness properties (cfr. Section 7).

As we have discussed, it was the advent of blockchains to renew the interested in consensus algorithms (e.g., [17, 19, 40, 51, 71]). Notably, PoS [43] was quickly proposed as an alternative to PoW, known to have energy consumption issues, especially for permissionless blockchains (e.g., [11, 13, 19]). Research on PoS led to two main approaches: One proposes a chain-based PoS that mimics PoW (e.g., [13, 42]); the other proposes a BFT-based PoS, where randomly selected nodes participate in a multi-round voting protocol to determine the next blocks to append (e.g., [19, 40, 67]). BFT-inspired protocols are preferred due to their deterministic block finality.

Other notable work in this area include: Istanbul BFT [51], a variant of PBFT tailored for blockchains; Pass and Shi [56] work on a consensus protocol for permissioned and permissionless blockchains

that combines a fast path and a slow one (as SBFT does); Algorand [40] and its proposal to scale PBFT by seeking consensus only on a subset of two-thirds of nodes, selected using PoS. These works do not consider quorum certificates or aggregated block signatures. Byzcoin [44] combines the use of PoW with BFT protocols to realize highly-performant open consensus protocols.

In contrast to all these scalability-oriented works, our setting considers a limited number of miners and expresses the calmness requirement (P4). Therefore, we seek protocols that favor simplicity and robustness, and focus on their tight integration with Bitcoin.

6.3 Threshold signatures

In threshold signature schemes, at least k participants over n, with $k \leq n$, collaborate for generating a valid signature *on-behalf* of the group. Shoup [66] defined one of the most used threshold signature schemes, based on RSA (e.g., [20, 41, 70, 81]). It requires a trusted, centralized dealer for key generation, and then uses non-interactive signature share generation and signature verification protocols.

Unfortunately, both the RSA signature scheme and the trusted dealer make this solution unsuited to our Bitcoin-derived setting.

Gennaro et al. [38] propose a threshold DSA signature scheme, with $k < n/2$, where a trusted centralized dealer is adopted. In [37], Gennaro et al. propose a dealer-less approach supporting the case $k < n$. However, DKG is costly and impractical. Then, Gennaro and Goldfeder [35, 36] presented an ECDSA-based protocol supporting efficient DKG, obtaining faster signing than [37] and requiring less data to be transmitted. In a closely related work, Lindell et al. [49] propose an efficient threshold ECDSA scheme, which employs different methods to neutralize any adversarial behavior.

A detailed (and more extensive) review of threshold ECDSA schemes can be found in [8]. Although ECDSA is fast and secure, aggregated signatures cannot be easily obtained with it, so we avoided this route.

Conversely, BLS [16] and Schnorr [64] schemes can be easily transformed into threshold versions by supporting the sum of partial signatures with no overhead [28].[40] In particular, Boldyreva [15] proposed the most widely adopted approach for threshold BLS signatures. DKG does not require a trusted dealer, and the signature generation does not require participant interaction (or any zero-knowledge proof). It can only tolerate up to $k < n/2$ malicious parties, but it allows to periodically renew the secret shares.

Most BFT protocols with a collector use threshold BLS signatures (e.g., [41, 71, 81]). Recently, Tomescu et al. [73] proposed a more efficient BLS signature scheme, that improves signing and verification time. Threshold BLS signature schemes rely on pairing-based cryptography [16]; however, in practice, this may be undesirable, due to the challenging implementation and to the need to maintain backwards compatibility with existing signature schemes.

Schnorr signatures received increased interest recently, and they have been included in the Bitcoin protocol. Komlo and Goldberg [45] propose FROST, an efficient Schnorr-based threshold scheme, whereby signing can be performed in two rounds, or optimized to a single round with preprocessing. FROST is currently considered the most efficient scheme for generating threshold Schnorr signatures [26], and is the one we have adopted in the present work. Ruffing et al. [61] propose ROAST (RObust ASynchronous Schnorr Threshold Signatures), a wrapper protocol around FROST that provides liveness guarantees in presence of malicious nodes and asynchronous networks. The 5-FBFT protocol we propose considers the idea introduced by ROAST to guarantee liveness while aggregating signature shares in the case of Byzantine signers.

[40] Schnorr and BLS signature schemes natively work with elliptic curves. While signature verification in BLS is more computationally demanding than ECDSA and Schnorr, signature generation in BLS is completely deterministic (thus preventing tampering).

7 FUTURE WORK

We are working on two primary research topics, both of which aim at making our prototype closer to a deployable, real-world solution: One concerns the improvement of the core PoA consensus algorithm itself ("*Dynamic federation*" and "*Improved fairness*" sections); the second one involves programming the blockchain to generate layer-2 constructions that contribute the missing features we expect of actual retail payment systems ("*Privacy and Scalability*").

Dynamic federation. Our mining network configuration is currently static: Miners are assigned their role at the beginning and cannot be changed without recreating the whole blockchain. This is unacceptable for real-world use, since dynamic reconfigurations are bound to happen for a variety of reasons: e.g., a member may need to change its public key periodically; a new member may need to join the federation; members may be removed. One hypothetical solution is to represent voting rights as non-fungible tokens (NFT) in our very chain: Block validity conditions would be the same as the spending condition of an NFT on the chain, and a transfer of the NFT would represent a change in the mining configuration.

Improved fairness. When BFT algorithms are used in blockchains, full fairness[41] is not achieved unless the primary can generate at most a single block before a view change. This would prevent a Byzantine primary to censor transactions. A fair mining network needs to rotate the leader at each block, e.g., using an election technique based on cryptographic sortition via Verifiable Random Functions from Algorand [39].

Privacy and Scalability. Our prospective model assumes that most of the privacy and scalability issues of actual retail payments are solved at the 2^{nd} layer. In particular we are experimenting with a dedicated Payment Channel Network similar to the Bitcoin Lightning [59], but with a less spontaneous topology, one that better fits our permissioned scenario and that is programmed over the distributed engine presented in this paper. Some assessments of, e.g., the scalability of Lightning Network already exist [72]; however, to get the whole picture we need to consider how different factors interplay: the raw throughput of the network, its business-driven topology, the level of privacy achieved by participants, and the costs of locking liquidity into channels by the routing intermediaries.

In addition, we are setting up a realistic scenario in which to measure the latency and volumes supported by our network at the first and second layer, including simulated Byzantine failures[42].

[41] We use the Chain Quality property definition from [33, 34], with parameters $\mu \in \mathbb{R}$ and $l \in \mathbb{N}$, which states that, for any honest party P with chain C, it holds that, for any consecutive l blocks of C, the ratio of honest blocks is at least μ. In our model, ideal fairness requires the Chain Quality property to hold for $\mu = 1 - F_B/N$ (ideal chain quality). Note that we do not need to include crash failures since they cannot contribute negatively to the chain quality.

[42] Test frameworks for Byzantine failures such as Twins [10] are being used.

8 CONCLUSIONS

We presented a Bitcoin-like, permissioned, distributed ledger in which valid blocks are signed by a federation of trusted actors and transactions enjoy deterministic finality. Block signatures are aggregated via a threshold scheme based on FROST, that preserves the confidentiality of the mining configuration and quorum. We showed how such a federation, via PBFT, could operate correctly also under Byzantine failures of a subset of the nodes.

Our design embodies one way of inheriting all the algorithms, data structures, and software of Bitcoin—but its PoW-based consensus protocol—in order to make its full technological stack openly available to permissioned settings managed by trusted actors.

What for? Bitcoin has inspired innumerable other blockchains and is perhaps the closest thing we have to an open standard for payments in the "crypto domain". It is possible that its technological stack—constantly scrutinized, improved, evolved[43]—will one day percolate[44] into the blockchain-friendly portion of the banking and financial ecosystem for old and new use cases. Solutions such as the one we present here pave the way for such a possibility.

REFERENCES

[1] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. 2017. Revisiting Fast Practical Byzantine Fault Tolerance. (2017). arXiv:1712.01367

[2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC '19)*. ACM, 337–346. https://doi.org/10.1145/3293611.3331612

[3] Karl-Johan Alm and Anthony Towns. 2019. Signet. *Bitcoin Improvement Proposal* 325 (2019). https://github.com/bitcoin/bips/blob/master/bip-0325.mediawiki

[4] Kurt M Alonso and Koe. 2020. *Zero to Monero.* Technical Report. Monero.

[5] Shikah J. Alsunaidi and Fahd A. Alhaidari. 2019. A Survey of Consensus Algorithms for Blockchain Technology. In *Proceedings of the International Conference on Computer and Information Sciences (ICCIS)*. 1–6. https://doi.org/10.1109/ICCISci.2019.8716424

[6] Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. 2019. An analysis of Uniswap markets. *arXiv:1911.03380* (2019).

[7] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2015. The Next 700 BFT Protocols. *ACM Trans. Comput. Syst.* 32, 4, Article 12 (2015), 45 pages. https://doi.org/10.1145/2658994

[8] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. 2020. A Survey of ECDSA Threshold Signing. *IACR Cryptol. ePrint Arch.* 2020 (2020), 1390.

[9] European Central Bank. 2020. Report on a digital euro. *ECB publications* (2020). https://www.ecb.europa.eu/pub/pdf/other/Report_on_a_digital_euro~4d7268b458.en.pdf

[10] Shehar Bano, Alberto Sonnino, Andrey Chursin, Dmitri Perelman, and Dahlia Malkhi. 2020. Twins: White-glove approach for BFT testing. *arXiv:2004.10617* (2020).

[11] Imran Bashir. 2020. *Mastering Blockchain: A Deep Dive into Distributed Ledgers, Consensus Protocols, Smart Contracts, Dapps, Cryptocurrencies, Ethereum, and More.* Packt Publishing Ltd.

[12] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. 2019. *State machine replication in the Libra blockchain.* Technical Report. The Libra Association.

[43] As an example of the forward-thinking surrounding the eldest blockchain, consider that work exists suggesting how to transform its cryptographic apparatus into a quantum-resistant one, even on-the-fly during a quantum attack, via a soft fork [68].

[44] This perspective has famed historical precedents. It is not unlike reusing the exact same technological stack from a decentralized, public network (the Internet) into private, "permissioned" networks (intranets): TCP/IP. At the dawn of the networking era, the idea of a convergent public/private stack was unheard of, and a host of custom, proprietary networking suites were deployed for "permissioned" use cases. But eventually, the good-enough and widely adopted TCP/IP won over most specialized (and mutually incompatible) protocols. It became a *de facto* standard. We are a very long way from a similar turn of events in the realm of digital payment systems. And, it may well be argued that a shared technological ground is unlikely to ever materialize. Still, we proved the idea makes technical sense, and we would better be ready.

[13] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. 2016. Cryptocurrencies Without Proof of Work. In *Proceedings of the 20th Internation Conference on Financial Cryptography and Data Security*. Springer, 142–157.

[14] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. 2014. State machine replication for the masses with BFT-SMART. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 355–362.

[15] Alexandra Boldyreva. 2002. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography — PKC 2003*, Yvo G. Desmedt (Ed.). Springer, 31–46.

[16] Dan Boneh, Ben Lynn, and Hovav Shacham. 2004. Short Signatures from the Weil Pairing. *J. Cryptol.* 17, 4 (2004), 297–319. https://doi.org/10.1007/s00145-004-0314-9

[17] Ethan Buchman. 2016. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains.* Ph. D. Dissertation. University of Guelph.

[18] Vitalik Buterin. 2014. *Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform.* Technical Report. Ethereum. https://ethereum.org/en/whitepaper/

[19] Vitalik Buterin and Virgil Griffith. 2019. Casper the Friendly Finality Gadget. *arXiv:1710.09437* (2019).

[20] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *J. Cryptol.* 18, 3 (2005), 219–246. https://doi.org/10.1007/s00145-005-0318-0

[21] Liang Cai, Yi Sun, Zibin Zheng, Jiang Xiao, and Weiwei Qiu. 2021. Blockchain in China. *Commun. ACM* 64, 11 (2021), 88–93.

[22] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*. USENIX Association, USA, 173–186.

[23] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* 20, 4 (2002), 398–461. https://doi.org/10.1145/571637.571640

[24] David Chaum, Christian Grothoff, and Thomas Moser. 2021. How to issue a central bank digital currency. *Available at SSRN 3965032* (2021).

[25] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. 2017. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 282–297.

[26] Elizabeth Crites, Chelsea Komlo, and Mary Maller. 2021. How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures. *Cryptology ePrint Archive* (2021). https://ia.cr/2021/1375

[27] William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs. 2018. EIP-721: ERC-721 non-fungible token standard. *Ethereum Improvement Proposals* 721 (2018).

[28] Sinan Ergezer, Holger Kinkelin, and Filip Rezabek. 2020. *A survey on threshold signature schemes.* Technical Report.

[29] Paul Feldman. 1987. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*. 427–438. https://doi.org/10.1109/SFCS.1987.4

[30] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

[31] Daniele Fornaro. 2018. *Elliptic curve hierarchical deterministic private key sequences: Bitcoin standards and best practices.* Technical Report. Politecnico di Milano. https://www.politesi.polimi.it/handle/10589/140112

[32] Juan Garay and Aggelos Kiayias. 2020. SoK: A Consensus Taxonomy in the Blockchain Era. In *Topics in Cryptology – CT-RSA 2020*, Stanislaw Jarecki (Ed.). Springer, 284–318.

[33] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT (2) (Lecture Notes in Computer Science, Vol. 9057)*. Springer, 281–310.

[34] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2020. Full Analysis of Nakamoto Consensus in Bounded-Delay Networks. *IACR Cryptol. ePrint Arch.* (2020), 277.

[35] Rosario Gennaro and Steven Goldfeder. 2018. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. ACM, 1179–1194. https://doi.org/10.1145/3243734.3243859

[36] Rosario Gennaro and Steven Goldfeder. 2020. One Round Threshold ECDSA with Identifiable Abort. *Cryptology ePrint Archive, Report 2020/540* (2020). https://ia.cr/2020/540

[37] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. 2016. Threshold-optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. In *Applied Cryptography and Network Security*, Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider (Eds.). Springer, 156–174.

[38] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2001. Robust Threshold DSS Signatures. *Information and Computation* 164, 1 (2001), 54–84. https://doi.org/10.1006/inco.2000.2881

[39] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*. 51–68.

[40] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 51–68. https://doi.org/10.1145/3132747.3132757

[41] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: A Scalable and Decentralized Trust Infrastructure. In *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 568–580. https://doi.org/10.1109/DSN.2019.00063

[42] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-stake Blockchain Protocol. In *Advances in Cryptology – CRYPTO 2017*, Jonathan Katz and Hovav Shacham (Eds.). Springer International Publishing, Cham, 357–388.

[43] Sunny King and Scott Nadal. 2012. *Ppcoin: Peer-to-peer crypto-currency with proof-of-stake*. Technical Report. https://www.peercoin.net/whitepapers/peercoin-paper.pdf

[44] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *USENIX Security Symposium*. USENIX Association, 279–296.

[45] Chelsea Komlo and Ian Goldberg. 2020. FROST: Flexible Round-optimized Schnorr Threshold Signatures. *IACR Cryptol. ePrint Arch.* 2020 (2020), 852.

[46] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2010. Zyzzyva: Speculative Byzantine Fault Tolerance. *ACM Transactions on Computer Systems* 27, 4, Article 7 (2010), 39 pages. https://doi.org/10.1145/1658357.1658358

[47] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* (1982), 382–401.

[48] Suhyeon Lee and Seungjoo Kim. 2020. Short Selling Attack: A Self-Destructive But Profitable 51% Attack On PoS Blockchains. Cryptology ePrint Archive. https://ia.cr/2020/019.

[49] Yehuda Lindell and Ariel Nof. 2018. Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. ACM, 1837–1854. https://doi.org/10.1145/3243734.3243788

[50] James Lovejoy, Cory Fields, Madars Virza, Tyler Frederick, David Urness, Kevin Karwaski, Anders Brownworth, and Neha Narula. 2022. A High Performance Payment Processing System Designed for Central Bank Digital Currencies. *Cryptology ePrint Archive* (2022).

[51] Henrique Moniz. 2020. The Istanbul BFT Consensus Algorithm. *arXiv:2002.03613* (2020).

[52] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). http://bitcoin.org/bitcoin.pdf

[53] Jonas Nick, Andrew Poelstra, and Gregory Sanders. 2020. *Liquid: A Bitcoin Sidechain*. Technical Report. Liquid. https://blockstream.com/assets/downloads/pdf/liquid-whitepaper.pdf

[54] Russell O'Connor. 2017. Simplicity: A new language for blockchains. In *Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security*. 107–120.

[55] Sunny Pahlajani, Avinash Kshirsagar, and Vinod Pachghare. 2019. Survey on Private Blockchain Consensus Algorithms. In *Proceedings of the 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*. 1–6. https://doi.org/10.1109/ICIICT1.2019.8741353

[56] Rafael Pass and Elaine Shi. 2018. Thunderella: Blockchains with Optimistic Instant Confirmation. In *Advances in Cryptology – EUROCRYPT 2018*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer International Publishing, Cham, 3–33.

[57] Torben Pryds Pedersen. 1991. A Threshold Cryptosystem without a Trusted Party. In *Advances in Cryptology – EUROCRYPT '91*, Donald W. Davies (Ed.). Springer, 522–526.

[58] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. 2018. Confidential assets. In *International Conference on Financial Cryptography and Data Security*. Springer, 43–63.

[59] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments.

[60] Federal Reserve. 2022. Money and Payments: The US Dollar in the Age of Digital Transformation. *Federal Reserve publications* (2022). https://www.federalreserve.gov/publications/money-and-payments-discussion-paper.htm

[61] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. 2022. ROAST: Robust Asynchronous Schnorr Threshold Signatures. *Cryptology ePrint Archive* (2022).

[62] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and David Mohaisen. 2020. Exploring the Attack Surface of Blockchain: A Comprehensive Survey. *IEEE Communications Surveys Tutorials* 22, 3 (2020), 1977–2008. https://doi.org/10.1109/COMST.2020.2975999

[63] Lakshmi Siva Sankar, M. Sindhu, and M. Sethumadhavan. 2017. Survey of consensus protocols on blockchain applications. In *Proceedings of the 4th International Conference on Advanced Computing and Communication Systems*. 1–5. https://doi.org/10.1109/ICACCS.2017.8014672

[64] Claus-Peter Schnorr. 1989. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*. Springer, 239–252.

[65] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613. https://doi.org/10.1145/359168.359176

[66] Victor Shoup. 2000. Practical Threshold Signatures. In *Advances in Cryptology – EUROCRYPT 2000*, Bart Preneel (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 207–220.

[67] Anping Song, Jing Wang, Wenjing Yu, Yi Dai, and Hongtao Zhu. 2019. Fast, Dynamic and Robust Byzantine Fault Tolerance Protocol for Consortium Blockchain. In *Proceedings of the 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 419–426.

[68] Iain Stewart, Daniel Ilie, Alexei Zamyatin, Sam Werner, MF Torshizi, and William J Knottenbelt. 2018. Committing to quantum resistance: a slow defence for Bitcoin against a fast quantum computing attack. *Royal Society open science* 5, 6 (2018), 180410.

[69] Liya Su, Xinyue Shen, Xiangyu Du, Xiaojing Liao, XiaoFeng Wang, Luyi Xing, and Baoxu Liu. 2021. Evil Under the Sun: Understanding and Discovering Attacks on Ethereum Decentralized Applications. In *30th USENIX Security Symposium (USENIX Security 21)*. 1307–1324.

[70] Quang Tung Thai, Jong-Chul Yim, Tae-Whan Yoo, Hyun-Kyung Yoo, Ji-Young Kwak, and Sun-Me Kim. 2019. Hierarchical Byzantine Fault-tolerance Protocol for Permissioned Blockchain Systems. *Journal of Supercomputing* 75, 11 (2019), 7337–7365.

[71] The Diem Team. 2021. *DiemBFT v4: State Machine Replication in the Diem Blockchain*. techreport. Facebook, Inc. https://developers.diem.com/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2021-08-17.pdf

[72] Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. 2020. A quantitative analysis of security, anonymity and scalability for the lightning network. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 387–396.

[73] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. 2020. Towards Scalable Threshold Cryptosystems. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy*. 877–893. https://doi.org/10.1109/SP40000.2020.00059

[74] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. 2009. Spin One's Wheels? Byzantine Fault Tolerance with a Spinning Primary. In *Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems*. 135–144. https://doi.org/10.1109/SRDS.2009.36

[75] Fabian Vogelsteller and Vitalik Buterin. 2015. Eip 20: Erc-20 token standard. *Ethereum Improvement Proposals* 20 (2015).

[76] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[77] Pieter Wuille. 2012. Hierarchical Deterministic Wallets. *Bitcoin Improvement Proposal* 32 (2012). https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

[78] Pieter Wuille, Jonas Nick, and Anthony Towns. 2020. Taproot: SegWit version 1 spending rules. *Bitcoin Improvement Proposal* 341 (2020). https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki

[79] Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. 2020. A Survey of Distributed Consensus Protocols for Blockchain Networks. *IEEE Communications Surveys Tutorials* 22, 2 (2020), 1432–1465. https://doi.org/10.1109/COMST.2020.2969706

[80] Congcong Ye, Guoqiang Li, Hongming Cai, Yonggen Gu, and Akira Fukuda. 2018. Analysis of Security in Blockchain: Case Study in 51%-Attack Detecting. In *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*. 15–24. https://doi.org/10.1109/DSA.2018.00015

[81] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC '19)*. ACM, New York, NY, USA, 347–356. https://doi.org/10.1145/3293611.3331591