

Certified Byzantine Consensus with Confidential Quorum for a Bitcoin-derived Permissioned DLT*

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

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 FBFT, a certified Byzantine consensus protocol that combines a modified Practical Byzantine Fault Tolerance (PBFT) protocol and a revised Flexible Round-Optimized Schnorr Threshold Signatures (FROST) scheme. We inject the resulting Proof-of-Authority (PoA) Byzantine consensus protocol into Bitcoin, thus replacing its PoW machinery. The resulting blockchain may offer a novel, modern, and safe foundation for digital payment systems used in stablecoins, sidechains, and Central Bank Digital Currencies (CBDCs). Lastly, we evaluate our solution using a prototype implementation of the full system, which we release in open-source.

Keywords

Bitcoin, blockchain, Byzantine consensus, threshold signature scheme

1. Introduction

The announcement of cryptoasset-inspired “stablecoins” by private companies [1] and the prospective issuance of Central Bank Digital Currencies (CBDCs) [2, 3, 4, 5] for retail use, coupled with the unabated diffusion of blockchain-based digital assets, have reignited the interest in consensus protocols amenable to permissioned blockchains. In this paper, we envision a distributed service provider that operates a modern, blockchain-based, programmable, and transactional engine, exhibiting high-availability and strong fault tolerance. Nodes may be managed by independent actors, which do not necessarily trust each other, but share a common interest, which would be perfectly served, technically, by a distributed ledger with no centralization point: Everyone enjoys equal rights, duties, and capabilities, and contributes to the system resilience. Nodes may even reside in different jurisdictions and conform to different laws, albeit under some shared regulatory framework¹. These motivations hold for

DLT'23: 5th Distributed Ledger Technology Workshop, May 25–26, 2023, Bologna, Italy


*All views are those of the authors and do not necessarily reflect the position of Bank of Italy.


✉ marco.benedetti@bancaditalia.it (M. Benedetti); francesco.desclavis@bancaditalia.it (F. De Sclavis);

marco.favorito@bancaditalia.it (M. Favorito); giuseppe.galano2@bancaditalia.it (G. Galano);

sara.giammusso@bancaditalia.it (S. Giammusso); antonio.muci@bancaditalia.it (A. Muci);

matteo.nardelli@bancaditalia.it (M. Nardelli)

 0000-0001-9566-3576 (M. Favorito); 0000-0002-9519-9387 (M. Nardelli)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

¹One hypothetical case would be a CBDC whose high availability and fault/attack tolerance rests upon a distributed platform operated *cooperatively*—in a profound sense—by several Central Banks in a given monetary area.

most permissioned DLT platforms, 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 identifiable third parties, in a permissioned setting*” a technically consistent one? Our goal is precisely to inherit *verbatim* all the algorithms, data structures, cryptography, and software from Bitcoin, getting rid of merely the ingredients (chiefly, PoW) that are unnecessary/undesirable in a *permissioned setting*. If it is possible to identify a small set of actors that end-users trust to cooperatively guarantee scarcity and to prevent double spending, then a Bitcoin-like blockchain can be grown via, e.g., a consensus based on Proof-of-Authority (PoA). High availability and tolerance to faults and malicious behaviors remain mandatory even in our smaller, permissioned setting. These properties can be recovered by borrowing and modifying existing consensus and signature algorithms from the literature. The difficult thing is to inject such algorithmic ingredients into Bitcoin while maximizing its codebase reuse, in order to inherit its virtues and strengths even after PoW is excised.

Contribution: Our major contribution is to show how three fairly sophisticated protocols, coming from different communities—namely PBFT [6] (*Practical Byzantine Fault Tolerance*) from the distributed system research, FROST [7] (*Flexible Round-Optimized Schnorr Threshold Signatures*), a result of recent cryptography studies, and Bitcoin from the crypto-assets realm—can be combined to make them interlock neatly with one another. From such pooling, a permissioned Bitcoin-derived DLT emerges, with strong fault tolerance and a confidential quorum certificate. 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. The major technical challenge to overcome is that a simple juxtaposition of PBFT and FROST does not work: Issues arise during distributed signature, because the possible reluctance of (faulty or malicious) nodes to sign blocks is something PBFT is unaware of and FROST is unable to deal with.

This work extends [8], where our core ideas were sketched. In this paper we detail our Certified Byzantine Consensus protocol, named FBFT, that combines PBFT and FROST to finalize a block with a quorum of signatures aggregated into a single one (improving confidentiality and block space efficiency). We also evaluate our solution using a prototype implementation of the full system, which is made available in open-source². Further details can be found at [9].

The rest of this paper is organized as follows: Sect. 2 offers a high-level view of our architecture; Sect. 3 recalls some preliminary concepts; Sect. 4 presents FBFT; Sect. 5 evaluates the proposed solution; Sect. 6 reviews the related literature, and Sect. 7 concludes the paper.

2. System Model and Requirements

2.1. High-level architecture

Our architecture is composed of a *participant network* and a *mining network*, each with different properties (see Fig. 1).

Participant and mining network. The *participant network* is composed of *participant nodes*,

²<https://bancaditalia.github.io/itcoin>

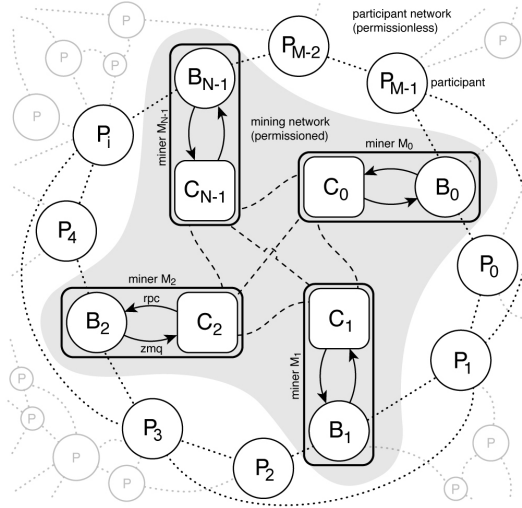


Figure 1: A permissioned mining network ($N = 4$) and a permissionless participant network.

noted P_0, \dots, P_{M-1} , which run a modified Bitcoin protocol (Sect. 4.3). Each participant node receives, validates, and stores a copy of the blockchain. The participants form a *permission-less* network, without a predefined topology or size. The bidirectional communication channels among them (dotted lines in Fig. 1) are used to propagate blocks and messages via gossiping, as in Bitcoin. The rounded rectangles inside the gray area are *mining*³ nodes, or “miners” (there are 4 of them in Fig. 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 operated by one member of a federation of N trusted actors⁴, 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 certified Byzantine consensus protocol described in Sect. 4. 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 Fig. 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 messages required by the consensus protocol (see Sect. 4).

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 Fig. 1), acting as *bridging channels*: 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 Sect. 4.1 and the self-loop messages in Fig. 2). The other bridging channel adopts a publish/subscribe model over the ZMQ protocol: The consensus node

³The term “mining” is etymologically incongruous in our context 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.

⁴We target settings in which N is expected to be between 4 and ≈ 20 .

subscribes to the bridging node in order to get the mining federation notified of occurrences of new signed blocks. These interaction models and protocols allow maximum Bitcoin reuse because they leverage the standard Bitcoin core APIs exposed by B_i .

Roles and coupling. The mining network is a service provider: Its goal is to collect transactions from participants, reach a consensus on which ones to include in new blocks, and then deliver signed blocks back to participants, who will add them to their local blockchains. Thanks to the properties of the consensus and signing protocols, this network appears to the participants as a single mining entity. Dually, the participant network acts as a single virtual client submitting transactions to the blockchain managed by the mining nodes, and expecting such transactions to be timely validated. The participant network is reliably connected to the miner network 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 channels within the permissionless network.

Failures. We assume a Byzantine failure model whereby F_B nodes can fail arbitrarily, with $F_B = \lfloor (N - 1)/3 \rfloor$. The consensus we employ relies on synchrony to provide liveness, but not to provide safety. To avoid the FLP impossibility result [10], we assume that (dashed) communication channels are weakly synchronous: Message delays among correct miners do not grow too fast and indefinitely, because a Global Stabilization Time (GST) event [11] eventually happens, after which the mining network behaves synchronously. Moreover, as in [12], we assume that all participants are able to synchronize in the course of a “round”, and that each round includes a GST event. 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 cryptographic primitives.

2.2. Requirements

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

- R1 **Correctness** (or, validity, consistency). Each block needs to have content that is valid according to the rules of the blockchain, and must transition the blockchain from one valid state to another.
- R2 **Safety** (or, agreement, deterministic finality). In a permissioned blockchain, safety forbids chain forks, i.e., different but valid versions of the most recent blocks of the same blockchain. This requires the Common Prefix Property [12] to be deterministic instead of probabilistic. Specifically, at the end of a round, if a honest participant “prunes” $k \geq 0$ blocks from the tip of its chain, the probability that the resulting pruned chain is not a prefix of another honest participant chain is exactly 0 (instead of exponentially decreasing with k , as in PoW blockchains).
- R3 **Liveness**. Within each round, new blocks must be produced every *block time* and propagated to the participants network every *round time*. We use the Chain Growth property [12], with parameters $\tau = \frac{\text{round time}}{\text{block time}} \in \mathbb{R}$ and $s \in \mathbb{N}$: For any honest participant, it holds that after any s consecutive rounds it adopts a chain that is at least $\lfloor \tau \cdot s \rfloor$ blocks longer.

R4 **Calmness.** The pace of block production is upper-bounded, which helps participants to form expectations on their resource requirements. If a Byzantine miner creates blocks at a rate significantly higher than $\frac{1}{\text{block time}}$, it can cause participants to run out of resources, effectively carrying out a denial-of-service attack. We require that after any s consecutive rounds it adopts a chain that is at most $\lceil \tau \cdot s \rceil$ blocks longer.

R5 **Confidentiality.** At each round carried on *with no faulty miners*,⁵ the mining network does not reveal information other than new valid blocks to the participants. Other information, e.g., the mining network configuration and the consensus quorum for block validation, should be kept hidden from the participants. This property can be used in addition to other anonymization mechanisms (e.g., at network level) to make targeted attacks against the miners harder.

R1-R3 have been already defined and studied in the context of blockchains [13], whereas R4-R5 are peculiar to ours.

3. Preliminaries: The FROST Signature Scheme

A (k, n) *threshold signature* scheme, with $k \leq n$, requires that at least k participants over n cooperate to create a valid signature, i.e., it is not possible to create a valid signature with less than k participants. FROST is a threshold signature scheme that leverages the additive property of Schnorr signatures to quickly combine signatures into an aggregated one [7]. The FROST signature scheme defines 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 briefly introduce these protocols, whose complete definition can be found in the original work [7].

Each participant M_i has a unique identifier $m_i \in \{1, \dots, n\}$. Let G be a group of prime order q in which the Decisional Diffie-Hellman problem is hard, g be a generator of G , and let H_1 and H_2 be cryptographic hash functions mapping to \mathbb{Z}_q^* . We denote by $x \leftarrow A$ that x is selected uniformly randomly from set A .

Key Generation. Before signing any block, participants need to define secret and public keys. They share the same cipher suite that specifies the underlying prime-order group details and cryptographic hash function. The KeyGen protocol consists of two rounds. Afterwards, each participant M_i , with $i \in \{1, \dots, n\}$, 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 Y enables the aggregate threshold signature verification, which depends on the set of participants n and the configured threshold k .

Commitment. In the commitment protocol, participants generate (secret) nonces for signatures and exchange their public commitments, which allow verifying the correct use of nonces. Each

⁵This property is impossible to guarantee in rounds where Byzantine failures happen since the network configuration is known to each participant, and a Byzantine node can choose to reveal extra information to the outside world.

participant M_i , $i \in \{1, \dots, n\}$, generates a pair of nonces $(d_i, e_i) \leftarrow \mathbb{Z}_q^* \times \mathbb{Z}_q^*$ and derives the public commitment shares $(D_i, E_i) = (g^{d_i}, g^{e_i})$.

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 in the signing process; the cardinality of S is α , with $k \leq \alpha \leq n$. Let $L = \langle (l, D_l, E_l) \rangle_{l=1}^\alpha$ be the list of α participants' commitments. 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 $\rho_l = H_1(l, m, L)$, $l \in \{1, \dots, \alpha\}$, 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 on m as $z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$, using (d_i, e_i) corresponding to $(i, D_i, E_i) \in L$, and S to determine the i -th Lagrange coefficient λ_i . Since nonces cannot be used multiple times, M_i deletes the $((d_i, D_i), (e_i, E_i))$ pair from his local storage. Then, M_i sends z_i to every other participant in S .

The second round starts when M_i receives all other signature shares z_l . For verification, M_i checks if the equality $g^{z_i} = R_i \cdot Y_i^{c \cdot \lambda_i}$ holds for each received z_l . If the verification is successful, M_i aggregates the signature shares locally by computing $z = \sum_{i \in S} z_i$. The resulting aggregated signature of m is $\sigma = (R, z)$, that can be verified as single-party signature.

4. Certified Byzantine Consensus

In our permissioned setting, blocks need to be authenticated in front of the participants network. Therefore, we employ a *Certified Byzantine Consensus Algorithm*, which allows reaching consensus on a sequence of blocks even in face of Byzantine faults and produces a proof of validity for each block, which enables the participants to verify the whole blockchain validity. To reach a consensus on blocks, we leverage a modified version of PBFT, while to produce the proof, we leverage a Schnorr threshold signature scheme called FROST, which also provides private quorum accountability [14]. It is worth noting that, in general, the set of block signers can differ from the set of nodes reaching the consensus.

4.1. Ordering blocks with 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. Service operations are invoked by *clients*, which send *requests* to the primary. Then a three-phase protocol begins: (i) in the *pre-prepare* phase, the primary assigns a sequence number to the request and multicasts it to the backups; (ii) in the *prepare* phase, the backups gather a *Byzantine quorum* of $2F_B + 1$ prepare messages in order agree on the sequence number proposed by the primary; (iii) in the *commit* phase, the replicas confirm that an agreement on the request and its sequence number has been reached by a *Byzantine quorum* of replicas. Then, each replica executes the operation and replies to the client. The client waits for a *reply quorum* of $F_B + 1$ replies from different replicas with the same result.

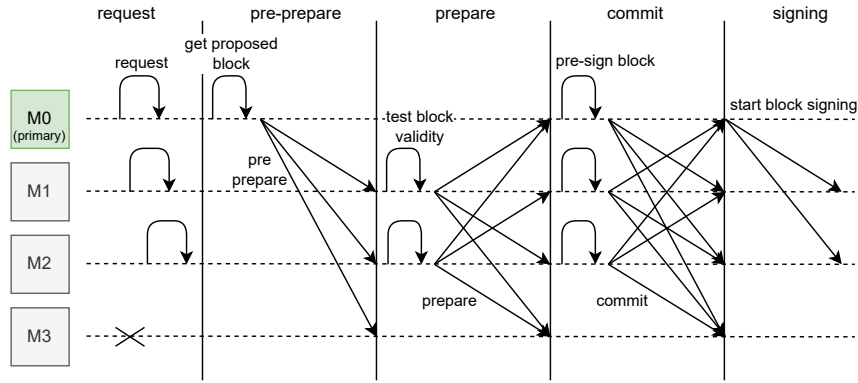


Figure 2: Normal operation (no faulty primary) with 4 nodes, M_0 is primary, M_3 is faulty.

The client. In our setting, the PBFT client is a single virtual entity, i.e., the participants network, and the state machine has a single operation, i.e., append a new block. The participants expect a new valid block to be mined every *target block time*, which in our examples is set to one minute, starting from the *genesis block timestamp*. In light of these considerations, all replicas know in advance all valid request timestamps, that are obtained as *genesis block timestamp* plus multiples of the *target block time*. Requests having such valid timestamps are self-generated locally by each replica. Moreover, given that valid blocks are broadcast to the participants network once a reply quorum of signatures by replicas is achieved, we omit the reply messages.

Normal operations (no faulty primary). The PBFT normal case protocol starts with a request to append a new block. The operation is non-deterministic, as its result depends on the actual content of the block to append (e.g., the set of transactions). We let the primary select and backups verify such content independently [15]. In the pre-prepare phase, when a block is expected at height n , the primary M_0 gathers a set of transactions from its mempool and forms a proposed block to be appended at height n , which corresponds to the sequence number of the operation. Then, the primary includes the block in the pre-prepare message and broadcasts it to backups. A backup (i.e., M_1 , M_2 , or M_3 in the figure) accepts a pre-prepare message *iff* it is valid according to the PBFT rules, its request has been already generated locally by the replica, its timestamp is not in the future according to the local clock of the replica, and the proposed block (checked by the participant node co-located with the replica) is also valid. 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 *iff* all the PBFT conditions are met; no additional checks are present at this stage. When replicas reach an agreement on a block and its height, they proceed to the commit phase. In the commit phase, a replica begins the signing process of the prepared block, by including a so-called *FROST commitments* in the commit message, and broadcasting the commit to other replicas. A replica accepts a commit message *iff* the PBFT conditions are met. A *Byzantine quorum* of commit messages contains a valid *reply quorum* of FROST commitments, which allow the primary to start the FROST signing sessions (described in Sect. 4.2).

Checkpoints. The checkpoint mechanism is used in PBFT to discard old messages and to

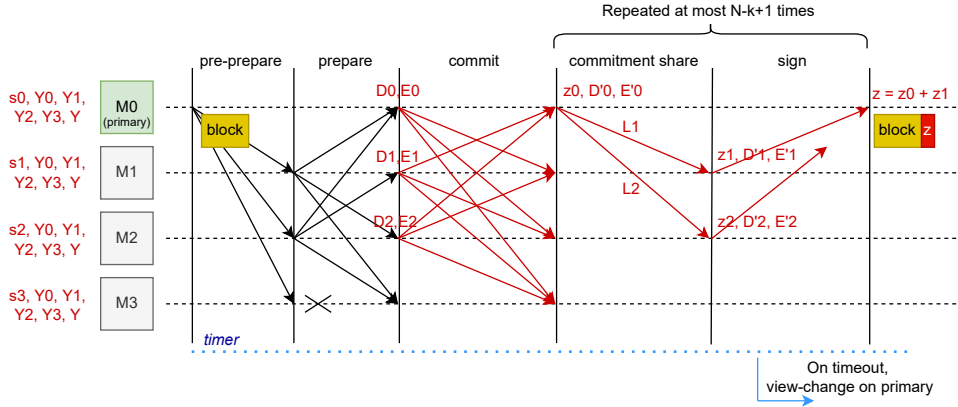


Figure 3: Normal operation of FBFT. Replicas exchange their first commitments in the commit phase. In the commitment share phase, the primary defines two sets of signers, $S_1 = \{0, 1\}$ and $S_2 = \{0, 2\}$, and related L_1 and L_2 parameters. In the sign phase, involved signers send the signature share to the primary who can determine the aggregated signature.

advance in the processing of requests. In our protocol, we rely on the Bitcoin block propagation mechanism of the underlying participants network for propagating checkpoints among replicas: Each block appended to the blockchain represents a PBFT checkpoint and causes the replica to move on to mining the next block.

4.2. FROSTing PBFT

Defining upfront the set of participants S that will collaborate to compute the aggregated signature z is cumbersome in presence of Byzantine nodes, which may arbitrarily refuse to sign blocks. We need rules for exchanging commitments (D_i, E_i) and identifying the set S of signers, two critical information for reconstructing the secret used to sign messages.

Therefore, we design *5-Phase Frosted-BFT* (FBFT, for short), which introduces two main changes to the PBFT protocol. First, it blends the commitment protocol and the 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 is a participant in the signing process, which collaborate to apply the threshold signature on the block agreed upon consensus. As per Fig. 3, the new rounds introduced in FBFT are called commitment-share and sign. When a message is prepared by replica i , 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 commit 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 commit message are considered as part of the initial set of responsive signers and will be candidates for the signing session in the commitment-share phase.

After replicas exchange the commit messages, a set of *commitment-share* phases of 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, a wrapper of the FROST protocol that guarantees

liveness [16]. We choose S with cardinality $k = F_B + 1$ (i.e., a *reply quorum* of signatures), including the primary itself: In such a configuration, at least one honest signer is in, thus preventing forgery of aggregate signatures over blocks that are invalid or not agreed upon. Even though the primary might exclude nodes suspected to be unresponsive, malicious nodes may be unknown and could be included in S . For this reason, the primary can initiate multiple and concurrent commitment-share sessions, maintaining a set of responsive signers. 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 $j \in S$ can compute the signature share z_j on the block.

The *sign* phase of FBFT allows replicas to run the aggregate signature protocol presented in Sect. 3. They create and exchange with the primary the signature shares z_i , with $i \in \{1, \dots, k\}$, 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 $i \in \{1, \dots, k\}$, it can derive the aggregate signature $\sigma = (R, z)$, with $z = \sum_i z_i$ and R the group commitment. If σ is a valid Schnorr signature, the primary appends the signature to the previously proposed block, broadcasts it, and completes FBFT. As demonstrated in ROAST [16], 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 failures F_B is at most $N - k$. The PBFT view-change protocol described in [15] 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.

We now informally argue that the FBFT algorithm satisfies the safety and liveness properties. The safety property relies on the usual cryptographic assumptions and a threshold adversary model with threshold $N > 3f$, while the liveness additionally relies on the partial synchrony of the network (as in PBFT).

Theorem 1 (Safety). *If a set of replicas produced a valid signature for block b_1 with sequence number n in view v , then no valid signature will be produced for block b_2 with n and v by another set of replicas.*

Proof. By contradiction, assume that for view v and sequence number n , there are two blocks b_1 and b_2 , with $b_1 \neq b_2$, for which $f + 1$ signature shares have been collected. Consider non-faulty replicas r_1 and r_2 that signed for b_1 and b_2 , respectively. If $r_1 = r_2$, we already get a contradiction: a correct replica signed two different blocks for n and v . If $r_1 \neq r_2$, by the safety property of PBFT, there cannot be two correct replicas that commit to two different blocks for the same sequence number. Similarly, the claim also holds for $v' > v$, since PBFT guarantees that if at least a correct replica locally committed the block b in v , which is the precondition to sign b , then no other request will be considered for the same sequence number n in later views $v' > v$. \square

Theorem 2 (Liveness). *All valid block proposal are eventually committed and signed by all correct replicas.*

Proof. The claim follows from the proof of termination of ROAST (Theorem 4.3 of [16]), and by the liveness property of PBFT, by taking care of triggering view-changes if a replica detects a Byzantine aggregator of signature shares. \square

The requirements of Correctness and Calmness are satisfied by construction, as a pre-prepare message is accepted only if its block is valid according to the protocol rules (Correctness), its request has been generated locally by the replica, and its timestamp is not in the future (Calmness). Finally, the Confidentiality requirement is guaranteed by the confidentiality of the signature created using FROST.

4.3. Amending the Bitcoin protocol

This section describes the main changes we introduce to the Bitcoin codebase.

Block validity. Our blocks are valid *iff* they include the solution to a specific *block challenge*, as in the Bitcoin Signet [17], that can be expressed either as a script or, since the introduction of Taproot [18], as a public key used to validate a Schnorr signature. For each block, the *block solution* to the challenge is stored in a special OP_RETURN output of the coinbase transaction, so it is automatically propagated to the participant network using the standard mechanisms for blocks and transactions. In our case, the solution is an aggregated Schnorr signature, representing a valid but opaque quorum of trusted miners who agreed to append a given block at a specific height. Since different quorums of signers may produce different valid signatures, in order to accommodate for our safety requirement (R2) in the context of a certified Byzantine consensus, it is necessary to exclude the block solution from the computation of the coinbase transaction hash⁶. In addition, it is necessary to include the PoW fields nBits and nNonce in the block signature, in order to prevent a (malicious) miner 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.

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; *the block Merkle root is now finalized.* 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; *the block hash is now finalized.* (2) A quorum of miners signs the block and appends a valid block solution; *the transactions are now finalized:* the Merkle root and block hashes are unaffected.

Block interval. The interval between blocks is fixed to one minute, instead of the 10-minute interval of the public Bitcoin network. We could further 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, transaction scalability is meant to be achieved off-chain.

⁶Different sets of signers for the same block could lead to valid but different block solutions. If the different solutions were included in the computation of the coinbase transaction hash, they would also be included in the Merkle trees, and would result in different block hashes, which would lead to a chain reorganization.

Block subsidy. It plays an incentive role in the public Bitcoin, which is *non-existent* in our setting. We remove the block subsidy checks from the code base of participant nodes.

Coinbase maturity. In Bitcoin, coinbase transaction outputs can only be spent after a certain number of new blocks. In our settings, no forks occur as per our safety requirement, therefore the coinbase maturity is safely set to 0.

5. Evaluation

We evaluate the performance of FBFT in a geographically distributed environment, involving up to 22 mining nodes, placed in 8 different European regions of Amazon Web Services (AWS).⁷ According to data collected by cloudping in 2022⁸, the median latency between these regions ranges between 20 ms and 50 ms, while the intra-region median latency stays below 4 ms.

The main measure of performance we consider is *consensus latency* (or just latency, for short), which represents the time needed by the mining network to reach an agreement and sign a new block. Measured at the primary node, it is the difference between the time at which a new signed block is submitted to the participant network (end of the consensus) and the time at which a new block is proposed to the mining network with a *pre-prepare* message (beginning of the consensus algorithm).

In Fig 4, we compare FBFT with two baseline variants, named PBFT and 3FBFT. The former produces a naïve block solution as the concatenation of signatures by a threshold of the mining nodes. This block solution can be then verified by the participant network using the `OP_CHECKMULTISIG` opcode. The latter is a trivial solution for creating a quorum certificate using FROST: miners run multiple FROST sessions during the consensus, by exchanging, in the commit message, a pair of commitments (D_i, E_i) for all the $\binom{N}{k}$ possible combinations of $k = 2F_B + 1$ signers. As a result, a replica that receives a *Byzantine quorum* of commit messages, can immediately execute the aggregation of the signature shares. 3FBFT optimizes communication because it minimizes the number of rounds required to finalize a block. However, since the number of signature shares grows exponentially, this protocol is practical only with small mining networks. For the three variants, Fig 4 shows the size of the block solution (i.e., the block signature), which represents a witness of the mining network agreement and is broadcast to the participant network together with the block itself. PBFT produces a block solution whose size increases with the mining network size (from 222 bytes with 4 replicas to 657 bytes with 13 replicas). Notably, the `OP_CHECKMULTISIG` opcode, used by participants to verify the block solution, allows checking at most 15 public keys. Both 3FBFT and FBFT represent an improvement over the PBFT baseline because they use FROST for creating a quorum certificate and produce a single Schnorr signature, which can be verified with an ad-hoc Taproot output. Therefore, the block solution size is 67 bytes, no matter the number of miners. Fig 4 also compares the three algorithms in terms of latency for different sizes of the mining network, here in absence of load. The experiments confirm that FBFT shows higher latency than PBFT, which is motivated by the presence of additional rounds needed for the FROST signature aggregation. Moreover, 3FBFT

⁷Namely: eu-west-1 (Ireland), eu-central-1 (Frankfurt), eu-south-1 (Milan), eu-north-1 (Stockholm), eu-west-2 (London), eu-central-2 (Zurich), eu-south-2 (Spain), eu-west-3 (Paris). We assign a sequential identifier to each node and determine the AWS region where to deploy it using the modulo function.

⁸https://www.cloudping.co/grid/p_50/timeframe/1Y

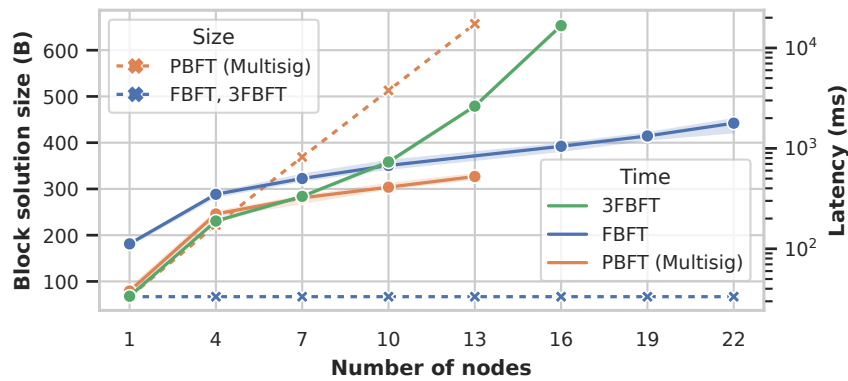


Figure 4: Solution size and average consensus latency achievable with naïve PBFT, FBFT, and 3FBFT.

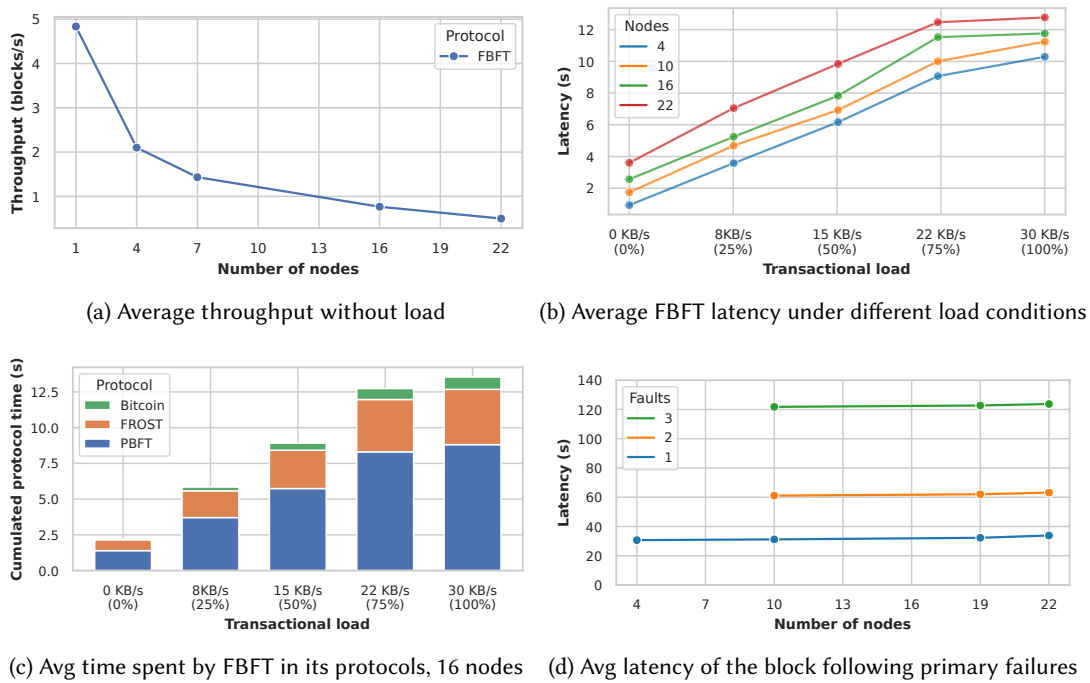


Figure 5: Evaluation of FBFT: throughput, time spent in its protocols, and consensus latency.

shows lower latency than FBFT in small mining networks, but its performances degrade very quickly when the number of nodes is greater than 10. This is motivated by the calculation of all possible combinations of a *Byzantine quorum* of signatures out of all the possible signers, leading to a prohibitively high latency of 19.2 s with 16 nodes. With 22 mining nodes, FBFT registers an average latency of 1.7 s in a setting spread across 8 AWS regions. This value of consensus latency is largely below our requirement of a new block every 60 s.

Fig. 5a reports the maximum throughput achievable with FBFT across the AWS European

regions. The throughput is the number of blocks that could be produced by the mining network per unit of time. It is slightly lower than the inverse of the consensus latency, because it also takes into account the time for block propagation in the mining network. We configure the experiments so that the network produces blocks as fast as possible (i.e., we “disable” the calmness, forcing miners to recover a blockchain with a genesis block time in the past). As expected, the throughput decreases as the mining network size increases. This is mainly due to the quadratic communication complexity of FBFT, which builds on PBFT.

In Fig 5b we investigate the impact of incoming transactions on consensus latency. We configure the mining network to generate a new block every minute in the steady state, during which we evaluate the performances. However, since we set the genesis block timestamp 30 minutes in the past, there is a warm-up period in which the miners will try to mine the first 30 blocks at the highest rate achievable with the given network conditions. In order to generate load for the mining network, we set up additional 8 participant nodes that submit transactions to the mining network. The warm-up period described above allows the clients to fill their wallets with a number of coins sufficient to generate a transaction load at the desired rate. The client rate is set to target a given block size that we describe as 0%, 25%, 50%, 75%, 100% of maximum block size (1.8 MB). As Fig 5b shows, when the load increases, the consensus latency increases as well. Indeed, the increase of the block size slows down the exchange of the *pre-prepare* message by the primary, and the block validity check by backups. We experience that the impact of transaction load is linear for all the mining network configurations. Nevertheless, even with blocks at full load and 22 mining nodes, the maximum latency we experienced is around 13 s, below our requirement to mine a block every 60 s.

To agree and sign the next block, FBFT uses different protocols: PBFT for consensus, FROST for signature aggregation, and Bitcoin for block validation. Fig. 5c details the time spent in these different stages, under different load conditions, with 16 mining nodes. When the load is 0 KB/s, FBFT closes empty blocks in 2170 ms: it spends 64% of time to complete the PBFT phases; 35.1% of time to exchange and aggregate the signature shares, whereas the Bitcoin block validity and submission checks take less than 1% of the time. If the load is at its maximum, then PBFT takes 65% of time, the signature aggregation takes 29%, and the block validity check takes 6% of time.

Finally, we evaluate the block latency in presence of failures. When the primary appears as faulty, FBFT uses the view change protocol to elect a new primary and recovers from failure. Fig. 5d shows the impact of failures in the worst-case scenario, where the primaries of subsequent views fail consecutively, and multiple view changes are triggered before finding the agreement on the next block. After 60 s, we forcefully terminate the primary of the mining network in the initial view, and possibly up to two other primaries expected for the next views. We set the initial view change timeout to 30 s for an expected block time of 60 s. Almost for every mining network size, the consensus latency is strongly delayed by the view-change protocol, which doubles subsequent timeouts with the number of subsequent views. It increases from less than 2 s to ≈ 30 s, with 1 failure, to ≈ 60 s, with 2 concurrent failures, to ≈ 120 s, with 3 concurrent failures (123.7 s with 22 nodes). When the view change is completed, the consensus protocol recovers the delayed blocks at the maximum throughput, and continues to mine with calmness at a consensus latency that is less than 2 s.

Overall, it appears that FBFT can provide Byzantine fault tolerance, network confidentiality, and efficient usage of block solution space, for just a reasonable increment in consensus latency.

6. Related work

There exist previous examples of Bitcoin-derived ledgers meant for private networks. In particular, Elements⁹, whose production deployment (the “Liquid” sidechain [19]) uses a consensus algorithm within a permissioned mining network consisting of cryptocurrency businesses. Elements is the closest work to ours in terms of technologies and Bitcoin reuse goals. However, to the best of our knowledge, no public specification for its BFT approach exists, and the open-sourced components¹⁰ do not include its implementation.

The second largest DLT born public and then adapted to permissioned settings is Ethereum [20]. An example of an Ethereum-like ledger designed for private networks is Hyperledger Besu¹¹, which supports a PoA consensus based on Istanbul BFT [21]. Another example is Concord¹². This is possibly the closest work to ours, in spirit, but (i) it implements SBFT [22] instead of PBFT as a consensus algorithm; (ii) it works with BLS signatures instead of Schnorr signatures; and (iii) it has Ethereum instead of Bitcoin as a foundation.

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 from major public blockchains. E.g.: Hyperledger Fabric¹³ is a general-purpose DLT that enables the development of enterprise applications, not necessarily financial. Among its components, there is a BFT consensus module, but its development appears to have ceased¹⁴. Corda¹⁵ is a DLT designed for the financial industry; it has a token-based data model, like Bitcoin, and a Turing-complete programming language, like Ethereum. Its *notaries* can run either a crash fault-tolerant (CFT) consensus or a BFT consensus. Neither the BFT specification nor its implementation is available in the open-source repository and, apparently, no aggregated signature scheme is included. Hyperledger Sawtooth¹⁶ allows deploying private DLT networks with a variety of consensus algorithms, including PBFT. Sawtooth has an open source implementation¹⁷, with an incomplete PBFT implementation and no Schnorr signature aggregation. Diem¹⁸ (formerly Libra) implements a Turing-complete programming language designed for safe and verifiable transaction-oriented computation. It employs a custom BFT algorithm called DiemBFT [23], based on Hotstuff [24]. Hamilton [25] is a DLT designed to support payments in a permissioned network (a use case similar to ours). It inherits certain elements from Bitcoin (e.g., UTXO and cryptographic primitives). Relevant differences: its ledger is not a blockchain and is meant to stay private; its consensus protocol is not BFT but CFT; its main focus is on obtaining

⁹<https://blockstream.com/elements>

¹⁰<https://github.com/ElementsProject>

¹¹<https://www.hyperledger.org/use/besu>

¹²<https://blogs.vmware.com/opensource/2018/08/28/meet-project-concord>

¹³<https://www.hyperledger.org/use/fabric>

¹⁴<https://github.com/bft-smart/fabric-orderingservice>

¹⁵<https://www.corda.net> and <https://github.com/corda>

¹⁶<https://www.hyperledger.org/use/sawtooth>

¹⁷<https://github.com/hyperledger/sawtooth-pbft>

¹⁸<https://developers.diem.com/docs/welcome-to-diem>

Table 1
Comparison of major permissioned distributed ledgers

	Byzantine Fault Tolerance	Confidential Quorum Certificate	Distributed Ledger Technology	Schnorr Quorum Certificate	Open Source
Elements	Strong Federations [29]	Unclear	Bitcoin	Unclear	Partially
Hyperledger Besu	IstanbulBFT[21]	No	Ethereum	No	Yes
Concord	SBFT [22]	Yes, BLS[30]	Ethereum	No	Yes
Hyperledger Fabric	No, Raft [31]	No	Fabric	No	Yes
Corda	Unclear	Unclear	Corda	Unclear	Partially
Hyperledger Sawtooth	PBFT [15]	No	Sawtooth	No	Yes
Diem	DiemBFT[23]	No	Diem	No	Yes
Project Hamilton	No, Raft [31]	No	Hamilton	No	Yes
Our solution	FBFT	Yes	Bitcoin	Yes, FROST [7]	Yes

transactional scalability on-ledger. Table 1 summarizes the main differences between the major permissioned ledgers.

We analyzed several BFT consensus algorithms (e.g., [22, 23, 24, 26, 27]), and we decided to design our block creation process around PBFT [15]. PBFT sacrifices linear communication (the number of exchanged messages is quadratic in the cluster size) in return for a simpler implementation; however, our requirements call for the mining network to produce no more than a block per minute, and our cluster is (by design) small enough to make the superlinear communication complexity a minor drawback, whereas simplicity in the implementation helps a lot the cohabitation with the Bitcoin platform. We implemented the BFT consensus algorithm from scratch instead of relying on already existing blockchain implementations, such as Tendermint [28], in order to maximize the reuse of and compatibility with the Bitcoin protocol, including its consensus engine.

Finally, note how most BFT protocols use threshold BLS signatures (e.g., [22, 24]), which rely on pairing-based cryptography. However, this may be challenging to implement in practice in our platform, since BLS signatures are not supported in Bitcoin. Conversely, Schnorr signatures received increased interest recently, and they have already been included in the Bitcoin protocol. Komlo and Goldberg [7] propose FROST (see Sect. 3), which is currently considered the most efficient scheme for generating threshold Schnorr signatures. Recently, Ruffing et al. [16] propose ROAST, a wrapper protocol around FROST, that provides liveness guarantees in presence of malicious nodes and asynchronous networks: Our FBFT protocol guarantees liveness like ROAST, in a peer to peer network which aggregates signature shares even in the case of Byzantine coordinator.

7. Conclusion

We presented and evaluated a Bitcoin-derived permissioned blockchain, where blocks are signed by a federation of trusted actors and transactions enjoy deterministic finality. Using a variant of PBFT, such a federation can correctly operate also in presence of limited Byzantine failures. Block signatures are aggregated via FROST, which preserves the confidentiality of the mining network configuration and quorum. As future work, we plan to evolve our architecture towards different research directions: dynamic federations (which enable network reconfiguration), fairness (which would prevent a Byzantine primary to censor transactions), privacy, and scalability.

References

- [1] The Diem Team, DiemBFT v4: State Machine Replication in the Diem Blockchain, techreport, Facebook, Inc., 2021. URL: <https://developers.diem.com/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2021-08-17.pdf>.
- [2] L. Cai, Y. Sun, Z. Zheng, J. Xiao, W. Qiu, Blockchain in china, *Communications of the ACM* 64 (2021) 88–93.
- [3] D. Chaum, C. Grothoff, T. Moser, How to issue a central bank digital currency, *arXiv:2103.00254* (2021).
- [4] European Central Bank, Report on a digital euro, ECB publications (2020).
- [5] Federal Reserve, Money and payments: The US Dollar in the age of digital transformation, Federal Reserve publications (2022).
- [6] M. Castro, B. Liskov, Practical Byzantine Fault Tolerance and Proactive Recovery, *ACM Trans. Comput. Syst.* 20 (2002) 398–461.
- [7] C. Komlo, I. Goldberg, FROST: Flexible Round-optimized Schnorr Threshold Signatures, in: *SAC 2020*, volume 12804, Springer, 2021, pp. 34–65.
- [8] M. Benedetti, F. De Sclavis, M. Favorito, G. Galano, S. Giammusso, A. Muci, M. Nardelli, PoW-less Bitcoin with Confidential Byzantine PoA, in: *Proc. 2023 IEEE Int. Conf. Blockchain and Cryptocurrency (ICBC)*, 2023. (to appear).
- [9] M. Benedetti, F. De Sclavis, M. Favorito, G. Galano, S. Giammusso, A. Muci, M. Nardelli, A PoW-less Bitcoin with Certified Byzantine Consensus, *arXiv:2207.0687* (2022).
- [10] M. J. Fischer, N. A. Lynch, M. S. Paterson, Impossibility of distributed consensus with one faulty process, *Journal of the ACM (JACM)* 32 (1985) 374–382.
- [11] C. Dwork, N. Lynch, L. Stockmeyer, Consensus in the Presence of Partial Synchrony, *Journal of the ACM* 35 (1988) 288–323. doi:10.1145/42282.42283.
- [12] J. A. Garay, A. Kiayias, N. Leonardos, The Bitcoin backbone protocol: Analysis and applications, in: *EUROCRYPT 2015*, Springer, 2015, pp. 281–310.
- [13] J. A. Garay, A. Kiayias, N. Leonardos, Full analysis of Nakamoto consensus in bounded-delay networks, *IACR Cryptol. ePrint Arch.* (2020).
- [14] D. Boneh, C. Komlo, Threshold signatures with private accountability, *Cryptology ePrint Archive* (2022).
- [15] M. Castro, B. Liskov, Practical Byzantine Fault Tolerance, in: *Proc. of OSDI'99, USENIX Association, USA*, 1999, pp. 173–186.
- [16] T. Ruffing, V. Ronge, E. Jin, J. Schneider-Bensch, D. Schröder, ROAST: Robust asynchronous Schnorr threshold signatures, *Cryptology ePrint Archive* (2022).
- [17] K.-J. Alm, A. Towns, Signet, BIP 325 (2019).
- [18] P. Wuille, J. Nick, A. Towns, Taproot: SegWit version 1 spending rules, BIP 341 (2020).
- [19] J. Nick, A. Poelstra, G. Sanders, Liquid: A Bitcoin Sidechain, Technical Report, Liquid, 2020.
- [20] G. Wood, et al., Ethereum: A secure decentralised generalised transaction ledger, *Ethereum project yellow paper* 151 (2014) 1–32.
- [21] H. Moniz, The Istanbul BFT Consensus Algorithm, *arXiv:2002.03613* (2020).
- [22] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, A. Tomescu, SBFT: A Scalable and Decentralized Trust Infrastructure, in: *Proc.*

- of IEEE/IFIP DSN'19, 2019, pp. 568–580.
- [23] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, A. Sonnino, State machine replication in the Libra blockchain, Technical Report, The Libra Association, 2019.
 - [24] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, I. Abraham, HotStuff: BFT Consensus with Linearity and Responsiveness, in: Proc. ACM PODC '19, ACM, 2019, pp. 347–356.
 - [25] J. Lovejoy, C. Fields, M. Virza, T. Frederick, D. Urness, K. Karwaski, A. Brownworth, N. Narula, A high performance payment processing system designed for central bank digital currencies, Cryptology ePrint Archive (2022).
 - [26] Y. Buchnik, R. Friedman, Fireledger: A high throughput blockchain consensus protocol, Proc. VLDB Endow. 13 (2020) 1525–1539.
 - [27] T. Distler, Byzantine fault-tolerant state-machine replication from a systems perspective, ACM Comput. Surv. 54 (2021).
 - [28] E. Buchman, Tendermint: Byzantine Fault Tolerance in the Age of Blockchains, Ph.D. thesis, University of Guelph, 2016.
 - [29] J. Dilley, A. Poelstra, J. Wilkins, M. Piekarska, B. Gorlick, M. Friedenbach, Strong federations: An interoperable blockchain solution to centralized third-party risks, arXiv:1612.05491 (2016).
 - [30] D. Boneh, B. Lynn, H. Shacham, Short Signatures from the Weil Pairing, J. Cryptol. 17 (2004) 297–319.
 - [31] D. Ongaro, J. Ousterhout, In search of an understandable consensus algorithm, in: Proc. of USENIX ATC'14, USENIX Association, 2014, p. 305–320.