The Latency Price of Threshold Cryptosystem in Blockchains

Zhuolun Xiang[1], Sourav Das[2], Zekun Li[1], Zhoujun Ma[1], and Alexander Spiegelman[1]

[1] Aptos Labs
[2] University of Illinois Urbana-Champaign

Abstract. Threshold cryptography is essential for many blockchain protocols. For example, many protocols rely on threshold common coin to implement asynchronous consensus, leader elections, and provide support for randomized applications. Similarly, threshold signature schemes are frequently used for protocol efficiency and state certification, and threshold decryption and threshold time-lock puzzles are often necessary for privacy.

In this paper, we study the interplay between threshold cryptography and a class of blockchains that use Byzantine-fault tolerant (BFT) consensus protocols with a focus on latency. More specifically, we focus on *blockchain-native threshold cryptosystem*, where the blockchain validators seek to run a threshold cryptographic protocol once for every block with the block contents as an input to the threshold cryptographic protocol. All existing approaches for blockchain-native threshold cryptosystems introduce a latency overhead of at least one message delay for running the threshold cryptographic protocol. In this paper, we first propose a mechanism to eliminate this overhead for blockchain-native threshold cryptosystems with *tight* thresholds, i.e., in threshold cryptographic protocols where the secrecy and reconstruction thresholds are the same. However, many real-world proof-of-stake-based blockchain-native threshold cryptosystems rely on ramp thresholds, where reconstruction thresholds are strictly greater than secrecy thresholds. For these blockchains, we formally demonstrate that the additional delay is unavoidable. We then introduce a mechanism to minimize this delay in the optimistic case. We implement our optimistic protocol for the proof-of-stake distributed randomness scheme on the Aptos blockchain. Our measurements from the Aptos mainnet show that the optimistic approach reduces latency overhead by 71%, from 85.5 ms to 24.7 ms, compared to the existing method.

1 Introduction

Threshold cryptography plays a vital role in modern blockchains, where various applications rely on threshold cryptography primitives such as distributed randomness, threshold signature, and threshold decryption. In threshold cryptography, a secret is shared among a set of parties using a threshold secret sharing [41,17], and parties seek to collaboratively evaluate a function of the shared secret and some public information without revealing the shared secret. For security, we require that the function of the shared secret and the public information is revealed only if a threshold fraction of parties contribute to the function evaluation.

In this paper, we study the interplay between threshold cryptography and a class of blockchains that use Byzantine-fault tolerant (BFT) consensus protocols with a focus on latency. More specifically, we focus on *blockchain-native threshold cryptosystem*, where the blockchain validators seek to run a threshold cryptographic protocol TC once for every block with the block's content as an input to TC protocol. In particular, we focus on schemes where the secret is shared using the Shamir secret sharing scheme [41], and the threshold cryptographic protocol is non-interactive, i.e., parties send a single message during the threshold cryptography protocol.

One concrete example of a blockchain-native threshold cryptosystem is the recent distributed randomness protocol for proof-of-stake blockchains [23], which has been deployed in the Aptos blockchain [10]. In [23], parties collaboratively compute a threshold verifiable random function (VRF) to generate shared randomness for each block, using the cryptographic hash of the block as an input to the threshold VRF. Similarly, Kavousi et al. [33] propose to use threshold decryption to mitigate Maximal Extractable Value (MEV) attacks by the block proposers. Specifically, [33] presents a solution where blockchain validators first order a set of encrypted transactions using a consensus protocol. Next, upon ordering, block validators run a threshold decryption protocol to decrypt the finalized transactions and execute them. Another practical example is the use of threshold signatures (or multi-signatures) by several blockchains [10,47] to certify the blockchain state after each block, and the aggregated threshold signature serves as proof of transaction inclusion for clients. In these blockchains, validators sign and exchange signature shares once each block is finalized and executed. This collective signing also prevents forking of the blockchain state in order-then-execute blockchains, particularly when execution is non-deterministic due to software bugs.

One limitation of existing blockchain-native threshold cryptosystem is that parties participate in the TC protocol only after the block is finalized. Hence, all existing protocol introduces at least one additional message delay before the output of the TC protocol is available, for the parties to exchange their TC shares. As a result, blockchains that seek to use the output of the TC protocol to execute the finalized transactions also incur this additional latency. For blockchains with optimal consensus latency, the additional round of communication adds at least 33% latency overhead, which is significant. More precisely, the optimal consensus latency is known to be three message delays [38,7,36], which is achieved in several protocols and systems [21,8]. For such protocols, the additional round of communication adds at least 33% latency overhead.

In this paper, we investigate whether the additional delay is inherent in supporting threshold cryptography in BFT-based blockchains. More specifically, let TC be a threshold cryptography scheme. Then, the *secrecy threshold* of TC is the upper bound on the number of TC messages an adversary can learn without learning the output of the TC protocol. Alternatively, the *reconstruction threshold* is the number of TC messages an honest party requires to be able to compute the TC output. Committee-based blockchains where the parties have equal weights (stakes), such as Dfinity [31], can use blockchain-native threshold cryptosystem with the same secrecy threshold as the reconstruction threshold. For a wide variety of these blockchains, we present a simple protocol in which the parties can compute the TC output simultaneously with the block finalization time. More specifically, our protocol applies to all BFT consensus protocols that satisfy the property that a value is finalized if and only if a threshold number of parties *prefinalize* the value.

However, many deployed proof-of-stake based blockchains [27,43,10,47] where parties have unequal stakes, will rely on threshold cryptography with *ramp* thresholds [18], i.e., use threshold cryptographic protocols where the reconstruction threshold is strictly larger than the secrecy thresholds. The ramp nature of threshold cryptography in these protocols is because these protocols assign to each party an approximate number of shares proportional to their stake [48]. This approximate assignment of a number of shares to each party introduces a gap between the secrecy and reconstruction threshold, as the assignment process may allocate more shares to the corrupt parties and fewer shares to honest ones. Somewhat surprisingly, we prove a lower bound result illustrating that for blockchain-native threshold cryptosystem with ramp thresholds, the extra latency incurred by existing protocols is inherent for a wide family of consensus protocols.

As a practical solution to circumvent this impossibility result, we propose a mechanism to design blockchain-native threshold cryptosystem protocols with ramp thresholds that achieve small latency overhead under optimistic executions. To demonstrate the effectiveness of our approach, we implement our optimistic solution atop the distributed randomness protocol (based on threshold VRF) used in the Aptos blockchain and evaluate its performance with their prior protocol. Our evaluation with real-world deployment illustrates that our optimistic approach reduces latency overhead by 71%.

To summarize, this paper makes the following contributions:

- We propose a mechanism (Algorithm 2) to remove the latency overhead for blockchain-native threshold cryptosystem with *tight* secrecy and reconstruction thresholds. The result applies to committee-based blockchain systems where parties have *equal* weights.
- We show an impossibility result (Theorem 6) indicating that the latency overhead is inherent for blockchain-native threshold cryptosystem with *ramp* thresholds, and present a solution (Algorithm 3) that can remove the latency overhead under optimistic scenarios. The results apply to proof-of-stake blockchain systems where parties have *unequal* weights.
- We implement our solution of ramp thresholds for distributed randomness and present evaluation numbers from the Aptos mainnet deployment. The evaluation demonstrates that the solution significantly improves the latency overhead by 71%, from 85.5 to 24.7 ms.

2 Related Work

Latency in blockchains. Latency reduction in blockchain has been an important problem for decades. Many works focused on reducing the consensus latency under partial synchrony, for leader-based BFT protocols [38,7,30,29,32,24] and DAG-based BFT protocols [46,45,12,34,14]. Several other works add a fast path to the consensus protocol, allowing certain transactions to be finalized faster [15,16,47]. Research has also focused on improving the latency in consensus protocols in various optimistic scenarios. Optimistic BFT protocols aim to achieve small latencies under certain optimistic conditions [6,25,44,35]. For instance, the optimistic fast paths in [6,42] require more than $3n/4$ parties to be honest in synchrony, while those in [35,30] require all n parties to vote under partial synchrony.

Threshold cryptography in blockchains. In many applications of modern blockchains, threshold cryptography plays a vital role. An important example of blockchain-native threshold cryptosystem is to generate distributed randomness for blockchains [23,31], which samples a fresh randomness beacon for each finalized block, allowing any transaction in the block to directly sample randomness pseudo-randomly using the block-level beacon. For distributed randomness, existing real-world deployments [23,31,10,47,28] follow the threshold VRF-based approach. Certifying the blockchain state using threshold signatures (or multi-signature) is also a common practice [10,47] for order-then-execute blockchains. Numerous blockchain research [40,49,13,51,39,33,37] focuses on MEV countermeasures and privacy enhancement using threshold decryption, where transactions are encrypted with threshold decryption and only revealed and executed by blockchain parties after finalization.

To the best of our knowledge, all existing constructions supporting the blockchain-native threshold cryptosystem mentioned above incur an additional round of latency, as parties participate in the threshold cryptographic protocol only after each block is finalized. We believe this is due to the black-box use of the underlying consensus mechanism. Looking ahead, our proposed constructions reduce this latency by using the underlying consensus protocols in a non-black-box manner. Our techniques are not tied to a specific blockchain but are applicable to a wide range of blockchains that satisfy Definition 5.

3 Preliminaries

Notations. For any integer a, we use $[a]$ to denote the ordered set $\{1, 2, \ldots, a\}$. For any set S, we use $|S|$ to denote the size of set S. We use λ to denote the security parameter. A machine is probabilistic polynomial time (PPT) if it is a probabilistic algorithm that runs in time polynomial in λ. We use $\mathsf{negl}(\lambda)$ to denote functions that are negligible in λ. We summarize the notations we use in the paper in Table 1.

3.1 System Model

We consider a set of n parties labeled $1, 2, ..., n$, where each party executes as a state machine. For brevity, we present the results for parties with equal weights, which can be easily extended to the case with unequal weights. The parties communicate with each other by message passing, via pairwise connected communication channels that are authenticated and reliable. We consider a *static* adversary $\mathcal{A}$ that can corrupt up to t parties before the execution of the system. A corrupted party can behave arbitrarily, and a non-corrupted party behaves according to its state machine. We say that a non-corrupted party is *honest*. We use $\mathcal{C}$ to denote the set of corrupted parties, and $\mathcal{H}$ to denote the set of honest parties. The network is assumed to be *partially synchronous*, where there exists a known message delay upper bound Δ, and a global stabilization time (GST) after which all messages between honest parties are delivered within Δ [26]. The adversary can receive messages from any party instantaneously.

3.2 Blockchain Definitions

We define Multi-shot Byzantine Broadcast under partial synchrony as follows to capture the consensus layer of many real-world blockchains that assume a partial synchronous network. We will use Multi-shot

Symbol	Description
MBB	Multi-shot Byzantine Broadcast (Definition 1)
$\mathsf{MBB}_{\mathsf{FT}}$	Multi-shot Byzantine Broadcast with finalization threshold (Definition 5)
TC	Threshold Cryptosystem (Definition 6)
BTC	blockchain-native threshold cryptosystem (Definition 11)
$\mathsf{BTC}_{\mathsf{FT}}$	blockchain-native threshold cryptosystem with finalization threshold (Definition 13)
t_{sec}	secrecy threshold in TC (Definition 6)
t_{rec}	reconstruction threshold in TC (Definition 6)
t_{fin}	finalization threshold in $\mathsf{MBB}_{\mathsf{FT}}$ (Definition 5)
GFT_r	global finalization time of round r in MBB (Definition 4)
GRT	global reconstruction time in TC (Definition 10)
GRT_r	global reconstruction time of round r in BTC
L_r	latency of round r in BTC (Definition 12)

Table 1: Table of Notations

Byzantine Broadcast and consensus interchangeably throughout the paper. Intuitively, Multi-shot Byzantine Broadcast consists of infinite instances of Byzantine Broadcast with rotating broadcasters and guarantees a total ordering among all instances.

Definition 1 (Multi-shot Byzantine Broadcast). *Multi-shot Byzantine Broadcast is defined for a message space $\mathcal{M}$ where $\perp \notin \mathcal{M}$, and rounds $r = 0, 1, 2, ...$ where each round $r \in \mathbb{N}$ has one designated broadcaster B_r who can call* $\mathsf{bcast}(r, m)$ *to broadcast a message $m \in \mathcal{M}$. For any round $r \in \mathbb{N}$, each party can output* $\mathsf{finalize}(r, m)$ *once to finalize a message $m \in \mathcal{M} \cup \{\perp\}$. The Multi-shot Byzantine Broadcast problem satisfies the following properties.*

- Agreement. *For any round $r \in \mathbb{N}$, if an honest party i outputs* $\mathsf{finalize}(r, m)$ *and an honest party j outputs* $\mathsf{finalize}(r, m')$*, then $m = m'$.*
- Termination. *After GST, for any round $r \in \mathbb{N}$ each honest party eventually outputs* $\mathsf{finalize}(r, m)$ *where $m \in \mathcal{M} \cup \{\perp\}$.*
- *Validity. If the designated broadcaster B_r of round r is honest and calls* $\mathsf{bcast}(r, m)$ *for any $m \in \mathcal{M}$ after GST, then all honest parties eventually output* $\mathsf{finalize}(r, m)$.
- Total Order. *If an honest party outputs* $\mathsf{finalize}(r, m)$ *before* $\mathsf{finalize}(r', m')$*, then $r < r'$.*

A Multi-shot Byzantine Broadcast protocol MBB *defines the state machine for each party to solve Multi-shot Byzantine Broadcast.*

The definition of Multi-shot Byzantine Broadcast captures many existing partially synchronous leader-based BFT protocols, or with minor modifications [‡], such as PBFT [20], Tendermint [19], SBFT [30], Hot-Stuff [50], Streamlet [22], Fast-Hotstuff [32], Jolteon [29] Moonshot [24] and many others. Multi-shot Byzantine Broadcast also captures another series of DAG-based BFT protocols, such as Bullshark [46], Shoal [45], Shoal++ [12], Cordial miners [34] and Mysticeti [14].

Now we define an *execution* of a MBB protocol, and when a message is *globally finalized* for a round in an execution.

[‡]Many chained BFT protocols such as HotStuff [50] and Jolteon [29] achieves a weaker Validity property. In these protocols, the finalization of the message proposed by the broadcaster of round r requires multiple consecutive honest broadcasters starting from round r. This weaker Validity does not affect the results we present in this paper.

Definition 2 (Execution). *A configuration of the system consists of the state of each party, together with all the messages in transit. Each execution of a Multi-shot Byzantine Broadcast protocol is uniquely identified by the sequence of configurations.*

Definition 3 (Multivalent and univalent state). *During the execution of a Multi-shot Byzantine Broadcast protocol, the system is in a multivalent state for round r, if there exist two possible executions $\mathcal{E} \neq \mathcal{E}'$ both extending the current configuration, where some honest party output differently in $\mathcal{E}, \mathcal{E}'$; the system is in a univalent state of $m \in \mathcal{M} \cup \{\perp\}$ for round r, if for all execution extending the current configuration, all honest party always outputs* $\mathsf{finalize}(r, m)$.

Definition 4 (Global finalization). *During the execution of a Multi-shot Byzantine Broadcast protocol, a message $m \in \mathcal{M} \cup \{\perp\}$ is globally finalized for round r, if and only if the system is in the univalent state of m for r. The global finalization time* GFT_r *of round r is defined as the earliest physical time when a message is globally finalized for r.*

We say that a party locally finalizes m for round r when it outputs $\mathsf{finalize}(r, m)$.

Intuitively, a message is globally finalized for a round r in Multi-shot Byzantine Broadcast when it is the only message that can be the output of r. Global finalization is a global event that may not be immediately known to any honest party, but must occur no later than the moment that any honest party outputs $\mathsf{finalize}(r, \cdot)$. Compared to local finalization, which occurs when any honest party outputs $\mathsf{finalize}(r, \cdot)$, global finalization is more fundamental. A MBB protocol can easily alter the time of local finalization simply by redefining the conditions for an honest party to local finalize (such as adding artificial delays to local finalization). However, modifying the global finalization time requires a more comprehensive protocol change. Therefore, we use the global finalization time for our impossibility result. For the feasibility results where we present protocol constructions, the time of local finalization will be considered.

Multi-shot Byzantine Broadcast with Finalization Threshold. The paper focuses on a family of MBB protocols that have a finalization threshold defined as follows. We call such a protocol Multi-shot Byzantine Broadcast with finalization threshold, or $\mathsf{MBB}_{\mathsf{FT}}$.

Definition 5 (Finalization Threshold). *For any round $r \in \mathbb{N}$, the Multi-shot Byzantine Broadcast with finalization threshold, or* $\mathsf{MBB}_{\mathsf{FT}}$*, has a state where a party calls* $\mathsf{prefinalize}(r, m)$ *to prefinalize a message $m \in \mathcal{M} \cup \{\perp\}$ for r, and sends a message* $(\texttt{PREFIN}, r, m)$ *to all parties, such that*

- *For any round r, any honest party calls* $\mathsf{prefinalize}(r, m)$ *for at most one $m \in \mathcal{M}$.*
- *For any round r, any honest party can call* $\mathsf{prefinalize}(r, \perp)$ *after calling* $\mathsf{prefinalize}(r, m)$ *for some $m \in \mathcal{M}$, but not the reverse.*
- *$m \in \mathcal{M} \cup \{\perp\}$ is globally finalized for r, if and only if there exist t_{fin} parties (or equivalently $t_{\mathsf{fin}} - |\mathcal{C}|$ honest parties) that have called* $\mathsf{prefinalize}(r, m)$.

Any party outputs $\mathsf{finalize}(r, m)$ *to locally finalize a message $m \in \mathcal{M} \cup \{\perp\}$ for a round r when the party receives* $(\texttt{PREFIN}, r, m)$ *messages from t_{fin} parties, which implies that m is globally finalized for r.*

We say t_{fin} is the finalization threshold of $\mathsf{MBB}_{\mathsf{FT}}$.

Intuitively, prefinalization is a local state of parties that satisfy the property that when enough honest parties prefinalize a message, then the message is globally finalized. A single party prefinalizing a message does not guarantee that the message will be finalized, and the party may finalize another message at the end. In many BFT protocols such as HotStuff [50], prefinalization is also named *lock*.

Examples of Multi-shot Byzantine Broadcast with finalization threshold. A larger number of MBB protocols used in partially synchronous blockchains fall into this family. As a concrete example, Jolteon [29] is a partially synchronous $\mathsf{MBB}_{\mathsf{FT}}$ protocol deployed by blockchains such as Aptos [10] and Flow [28]. We explain in detail how Jolteon satisfies the definition of Multi-shot Byzantine Broadcast with finalization threshold with finalization threshold in Section 7.2. Other than Jolteon, numerous partially synchronous BFT protocols are also part of this family or can be easily adapted to fit into this family, such as PBFT [20], Tendermint [19], SBFT [30], HotStuff [50], Streamlet [22], Fast-Hotstuff [32], Moonshot [24] and many others. Additionally, another series of DAG-based consensus protocols also satisfy the definition of Multi-shot Byzantine Broadcast with finalization threshold with finalization threshold, such as Bullshark [46], Shoal [45], Shoal++ [12], Cordial miners [34] and Mysticeti [14].

3.3 Cryptography Definitions

Next, we describe the syntax and security definitions for threshold cryptosystems. We focus on non-interactive threshold cryptographic protocols.

Definition 6 (Threshold Cryptosystem). *Let $t_{\mathsf{sec}}, t_{\mathsf{rec}}, n$ with $t_{\mathsf{sec}} \le t_{\mathsf{rec}} \le n$ be natural numbers. We refer to t_{sec} and t_{rec} as the secrecy and reconstruction threshold, respectively. Let $\mathcal{X}$ be a input space. Looking ahead, the input space $\mathcal{X}$ denotes the output space of the underlying Multi-shot Byzantine Broadcast protocol.*

A $(n, t_{\mathsf{sec}}, t_{\mathsf{rec}})$-threshold cryptosystem TC *is a tuple of PPT algorithms* $\mathsf{TC} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{Eval}, \mathsf{PEval}, \mathsf{PVer}, \mathsf{Comb})$ *defined as follows:*

1. $\mathsf{Setup}(1^\lambda) \to pp$. *The setup algorithm takes as input a security parameter and outputs public parameters pp (which are given implicitly as input to all other algorithms).*
2. $\mathsf{ShareGen}(s) \to \{\mathsf{pk}_i, [\![s]\!]_i\}_{i\in[n]}$. *The share generation algorithm takes as input a secret $s \in \mathcal{K}$ and outputs a vector of threshold public keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, and a vector of secret shares $([\![s]\!]_1, \ldots, [\![s]\!]_n)$. The j-th party receives $(\{\mathsf{pk}_i\}_{i\in[n]}, [\![s]\!]_j)$.*
3. $\mathsf{Eval}(s, \mathsf{val}) \to \sigma$. *The evaluation algorithm takes as input a secret share s, and a value $\mathsf{val} \in \mathcal{X}$. It outputs a function output σ, which is called the* TC *output in the paper.*
4. $\mathsf{PEval}([\![s]\!]_i, \mathsf{val}) \to \sigma_i$. *The partial evaluation takes as input a secret share $[\![s]\!]_i$, and a value $\mathsf{val} \in \mathcal{X}$. It outputs a function output share σ_i, which is called the* TC *output share in the paper.*
5. $\mathsf{PVer}(\mathsf{pk}_i, \mathsf{val}, \sigma_i) \to 0/1$. *The partial verification algorithm takes as input a public key pk_i, a value val, and a* TC *output share σ_i. It outputs 1 (accept) or 0 (reject).*
6. $\mathsf{Comb}(S, \mathsf{val}, \{(\mathsf{pk}_i, \sigma_i)\}_{i\in S}) \to \sigma/\bot$. *The combine algorithm takes as input a set $S \subseteq [n]$ with $|S| \ge t_{\mathsf{rec}}$, an value val, and a set of tuples $(\mathsf{pk}_i, \sigma_i)$ of public keys and* TC *output shares of parties in S. It outputs either a* TC *output σ or $\bot$.*

Definition 7 (Ramp [18] and tight thresholds). *For any $(n, t_{\mathsf{sec}}, t_{\mathsf{rec}})$-threshold cryptosystem, we call it a* tight *threshold cryptosystem if $t_{\mathsf{sec}} = t_{\mathsf{rec}}$, and a* ramp *threshold cryptosystem if $t_{\mathsf{sec}} < t_{\mathsf{rec}}$.*

We require a threshold cryptosystem to satisfy the following *Robustness* and *Secrecy* properties.

We formalize the robustness property using the $\mathsf{RB\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}$ game in Game 1. Intuitively, the robustness property ensures that the protocol behaves as expected for honest parties, even in the presence of an adversary that corrupts up to t parties. More precisely, it says that: (i) PVer should always accept honestly generated TC output shares and (ii) if we combine t_{rec} valid TC output shares (accepted by PVer) using the Comb algorithm, the output of Comb should be equal to $\mathsf{Eval}(s, \mathsf{val})$, except with a negligible probability. The latter requirement ensures that maliciously generated TC output share cannot prevent honest parties from efficiently computing $\mathsf{Eval}(s, \mathsf{val})$ (except with a negligible probability). Note that we allow $\mathcal{A}$ to generate TC output share arbitrarily. Also, we can achieve robustness even if $\mathcal{A}$ learns shares of all parties.

Game 1 (Robustness Under Chosen Message Attack) *For a $(n, t_{\mathsf{sec}}, t_{\mathsf{rec}})$-threshold cryptosystem* TC *we define the game $\mathsf{RB\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}$ in the presence of adversary $\mathcal{A}$ as follows:*

– **Setup.** *$\mathcal{A}$ specifies a set $\mathcal{C} \subset [n]$, with $|\mathcal{C}| < t_{\mathsf{sec}}$ of corrupt parties. Let $\mathcal{H} := [n] \setminus \mathcal{C}$ be the set of honest parties.*
– **Share generation**. *Run $\mathsf{ShareGen}(s)$ to generate the shares of s. $\mathcal{A}$ learns $[\![s]\!]_i$ for each $i \in \mathcal{C}$ and all the public keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$.*

– **Function evaluation shares.** *$\mathcal{A}$ submits a tuple (i, val) for some $i \in \mathcal{H}$ and $\mathsf{val} \in \mathcal{X}$ as input and receives $\sigma_i \leftarrow \mathsf{PEval}([\![s]\!], \mathsf{val})$.*
– **Output determination.** *Output 1 if either of the following happens; otherwise, output 0.*
  1. *$\mathcal{A}$ outputs (i, val) such that $\mathsf{PVer}(\mathsf{pk}_i, \mathsf{PEval}([\![s]\!]_i, \mathsf{val})) = 0$;*
  2. *$\mathcal{A}$ outputs $(S, \{\sigma_i\}_{i \in S}, \mathsf{val})$ where $S \subseteq [n]$ with $|S| \geq t_{\mathsf{rec}}$ and $\mathsf{PVer}(\mathsf{pk}_i, \sigma_i, \mathsf{val}) = 1$ for all $i \in S$, such that $\mathsf{Comb}(S, \{\mathsf{pk}_i, \sigma_i\}, \mathsf{val}) \neq \mathsf{Eval}(s, \mathsf{val})$.*

Definition 8 (Robustness Under Chosen Message Attack). *Let $\mathsf{TC} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{Eval}, \mathsf{PEval}, \mathsf{PVer}, \mathsf{Comb})$ be a $(n, t_{\mathsf{sec}}, t_{\mathsf{rec}})$-threshold cryptosystem. Consider the game $\mathsf{RB\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}$ defined in Game 1. We say that TC is $\mathsf{RB\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}$ secure, if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible, i.e.,*

$$\Pr[\mathsf{RB\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}(\lambda) \Rightarrow 1] = \mathsf{negl}(\lambda) \tag{1}$$

Next, we describe the secrecy property. Intuitively, the secrecy property ensures that $\mathsf{Eval}(s, \mathsf{val})$ remains hidden from an adversary $\mathcal{A}$ that corrupts up to t_{sec} parties, where the precise notion of "hidden" depends on the application. For example, when TC is a threshold signature scheme, i.e., $\mathsf{Eval}(s, \mathsf{val})$ is a signature on the message val using signing key s, the secrecy property is analogous to the standard *unforgeability* property of signature schemes. Similarly, when TC is a threshold decryption scheme, i.e., $\mathsf{Eval}(s, \mathsf{val})$ outputs the decryption of the ciphertext val using secret key s, the Secrecy property requires that TC is semantically secure in the presence of an attacker $\mathcal{A}$ that corrupts up to t parties.

Looking ahead, we will use a distributed randomness beacon as our concrete application (see Section 7), with unpredictability under chosen message attack as our secrecy property. Intuitively, the unpredictability property ensures that an adversary corrupting less than t_{sec} parties can not compute $\mathsf{Eval}(s, \mathsf{val})$ for a value val for which it has seen less than $t_{\mathsf{sec}} - |\mathcal{C}|$ TC output shares. We formalize this with the $\mathsf{UP\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}$ game in Game 2.

Game 2 (Unpredictability Under Chosen Message Attack) *For a $(n, t_{\mathsf{sec}}, t_{\mathsf{rec}})$-threshold cryptosystem TC the game $\mathsf{UP\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}$ in the presence of adversary $\mathcal{A}$ as follows:*

– **Setup.** *$\mathcal{A}$ specifies two sets $\mathcal{C}, \mathcal{S} \subset [n]$, with $|\mathcal{C} \cup \mathcal{S}| < t_{\mathsf{sec}}$. Here, $\mathcal{C}$ is the set of corrupt parties and $\mathcal{S}$ is the set of honest parties that $\mathcal{A}$ queries for TC output shares on the forged input,. Let $\mathcal{H} := [n] \setminus \mathcal{C}$ be the set of honest parties.*
– *The share generation, and function evaluation shares steps are identical to the $\mathsf{RB\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}$ game.*
– **Output determination.** *$\mathcal{A}$ outputs $(\mathsf{val}^*, \mathsf{Eval}(s, \mathsf{val}^*))$. Output 1 if $\mathcal{A}$ has queried for TC output share on val^* from only parties in $\mathcal{S}$. Otherwise, output 0.*

Definition 9 (Unpredictability Under Chosen Message Attack). *Let $\mathsf{TC} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{Eval}, \mathsf{PEval}, \mathsf{PVer}, \mathsf{Comb})$ be a $(n, t_{\mathsf{sec}}, t_{\mathsf{rec}})$-threshold cryptosystem. Consider the game $\mathsf{UP\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}$ in Game 2. We say that TC is $\mathsf{UP\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}$ secure, if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible, i.e.,*

$$\Pr[\mathsf{UP\text{-}CMA}^{\mathcal{A}}_{\mathsf{TC}}(\lambda) \Rightarrow 1] = \mathsf{negl}(\lambda) \tag{2}$$

We define global reconstruction and local reconstruction time as follows.

Definition 10 (Global reconstruction). *For any $(n, t_{\mathsf{sec}}, t_{\mathsf{rec}})$-threshold cryptosystem with secret s and input val, we say that $\mathsf{Eval}(s, \mathsf{val})$ is* globally reconstructed *if and only if the adversary learns $\mathsf{Eval}(s, \mathsf{val})$ (or equivalently $t_{\mathsf{rec}} - |\mathcal{C}|$ honest parties reveal TC output shares to $\mathcal{A}$). The global reconstruction time GRT is defined to be the earliest physical time when $\mathsf{Eval}(s, \mathsf{val})$ is globally reconstructed.*

We say that a party locally reconstructs $\mathsf{Eval}(s, \mathsf{val})$ when it learns $\mathsf{Eval}(s, \mathsf{val})$ (or equivalently receiving t_{rec} valid shares).

Double sharing of the secret. Looking ahead, we require our threshold cryptosystem to support double sharing of the same secret for two sets of thresholds $(t_{\mathsf{sec}}, t_{\mathsf{rec}})$ and $(t'_{\mathsf{sec}}, t'_{\mathsf{rec}})$ where $(t_{\mathsf{sec}}, t_{\mathsf{rec}}) \neq (t'_{\mathsf{sec}}, t'_{\mathsf{rec}})$. It is easy to see that threshold cryptosystems based on Shamir secret sharing [41] easily support double sharing.

4 Blockchain-Native Threshold Cryptosystem

We now formally define the problem of blockchain-native threshold cryptosystem. As mentioned in Section 1, in such a system, a secret is shared among the parties participating in the blockchain protocol and these parties seek to collaboratively run a threshold cryptographic protocol, after every block, using the shared secret and the block as input.

Definition 11 (Blockchain-Native Threshold Cryptosystem). *Let* MBB *be a Multi-shot Byzantine Broadcast protocol as in Definition 1. Let* $\mathsf{TC} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{Eval}, \mathsf{PEval}, \mathsf{PVer}, \mathsf{Comb})$ *be a* $(n, t_{\mathsf{sec}}, t_{\mathsf{rec}})$*-threshold cryptosystem as in Definition 6. A blockchain-native threshold cryptosystem protocol* $\mathsf{BTC} = (\mathsf{MBB}, \mathsf{TC})$ *is defined as follows.*

1. *The parties start with a secret share of a secret key* s *as per* $\mathsf{ShareGen}(s)$.
2. *The parties run* MBB.
3. *Upon* MBB *outputs* $\mathsf{finalize}(r, m)$ *for any round* $r \in \mathbb{N}$*, parties run the* TC *protocol to compute* $\sigma = \mathsf{Eval}(s, (r, m))$ [§] *and outputs* (r, m, σ). [¶]

We require BTC *to satisfy the following properties, except for negligible probabilities.*

- Agreement. *For any round* $r \in \mathbb{N}$*, if an honest party outputs* (r, m, σ) *and another honest party outputs* (r, m', σ')*, then* $m = m'$ *and* $\sigma = \sigma'$.
- Termination. *After GST, for any round* $r \in \mathbb{N}$ *each honest party eventually outputs* (r, m, σ) *where* $m \in \mathcal{M} \cup \{\perp\}$.
- Validity. *For any round* $r \in \mathbb{N}$*, if the designated broadcaster* B_r *is honest and calls* $\mathsf{bcast}(r, m)$ *for* $m \in \mathcal{M}$ *after GST, then all honest parties eventually output* $(r, m, \mathsf{Eval}(s, (r, m)))$.
- Total Order. *If an honest party outputs* (r, m, σ) *before* (r', m', σ')*, then* $r < r'$.
- Secrecy. *If an honest party outputs* (r, m, σ)*, then* $\sigma = \mathsf{Eval}(s, (r, m))$*, and the adversary cannot compute* $\mathsf{Eval}(s, (r, m'))$ *for* $m' \neq m$.

Example of blockchain-native threshold cryptosystem. To facilitate understanding, as a specific example of blockchain-native threshold cryptosystem, consider blockchain-native distributed randomness that generates a shared randomness for every finalized block. TC for this application can be a threshold VRF scheme. Upon MBB (the blockchain consensus layer) outputs m (a block) for a round r, parties run the TC protocol to compute the shared randomness $\mathsf{Eval}(s, (r, m))$.

For $\mathsf{BTC} = (\mathsf{MBB}, \mathsf{TC})$, we use GRT_r to denote the global reconstruction time of round r in TC, where the global reconstruction time in TC is defined in Definition 10.

Below we define the latency of a blockchain-native threshold cryptosystem to be the time from its finalization to reconstruction, which measures the latency overhead added by blockchain-native threshold cryptosystem. We define the latency for the feasibility results of this paper. Intuitively, L_r is the maximum time difference, across all honest parties, between the time a honest party i finalizes (m, r) from MBB and the time the same honest party i outputs $(m, r, \mathsf{Eval}(s, (m, r)))$.

Definition 12 (Latency of Blockchain-Native Threshold Cryptosystem). *During an execution* $\mathcal{E}$ *of a blockchain-native threshold cryptosystem* $\mathsf{BTC} = (\mathsf{MBB}, \mathsf{TC})$*, for any round* r *and party* i*, let* $T^{\mathsf{F}}_{i,r}$ *be the physical time when party* i *outputs* $\mathsf{finalize}(r, m)$ *for some* m *in* MBB*, and* $T^{\mathsf{O}}_{i,r}$ *be the physical time when party* i *outputs* (r, m, σ) *for some* m, σ *in* BTC*. The latency of the blockchain-native threshold cryptosystem for round* r *of execution* $\mathcal{E}$ *is defined to be* $\mathsf{L}_r = \max_{i \in \mathcal{H}}(T^{\mathsf{O}}_{i,r} - T^{\mathsf{F}}_{i,r})$.

[§] More generally, m can be the blockchain state after round r.

[¶] In the case MBB outputs $m = \perp$, computing $\sigma = \mathsf{Eval}(s, (r, m))$ may not be meaningful, and the solutions in the paper can be easily adapted to output $\sigma = \perp$ instead. For the ease of the presentation, we still let parties compute $\sigma = \mathsf{Eval}(s, (r, m))$ for $m = \perp$.

In a blockchain-native threshold cryptosystem, such as distributed randomness, the transaction execution relies on the TC output of the threshold cryptosystem. By definition, a party may have to wait a period of L_r before executing the transactions finalized for round r. Therefore, the latency of blockchain-native threshold cryptosystem is directly added to the blockchain's transaction end-to-end latency.

The following lemma is a direct implication of the blockchain-native threshold cryptosystem definition. Intuitively, a blockchain-native threshold cryptosystem should not reconstruct the TC output before a message is finalized.

Lemma 1. *For any blockchain-native threshold cryptosystem protocol, for any execution, and for all round $r \in \mathbb{N}$, we have $\mathsf{GRT}_r \geq \mathsf{GFT}_r$.*

Proof. Let $\mathsf{BTC} = (\mathsf{MBB}, \mathsf{TC})$ denote the protocol. Suppose that $\mathsf{GRT}_r < \mathsf{GFT}_r$ in some execution. By the definition of GFT_r and GRT_r, the adversary learns the TC output of round r at time GRT_r, before the message is globally finalized for r by MBB at time GFT_r. According to Definition 4, the system is in multivalent state for r at time GRT_r, which means there exist two execution extensions of MBB where the honest parties output different messages for r. Then, the Secrecy (Definition 13) of BTC is violated for at least one of the execution, contradiction.

4.1 Blockchain-Native Threshold Cryptosystem with Finalization Threshold

This paper focuses on a family of blockchain-native threshold cryptosystem protocols defined as follows.

Definition 13 (Blockchain-Native Threshold Cryptosystem with Finalization Threshold). *A blockchain-native threshold cryptosystem with finalization threshold, or $\mathsf{BTC}_{\mathsf{FT}} = (\mathsf{MBB}_{\mathsf{FT}}, \mathsf{TC})$, is a blockchain-native threshold cryptosystem (Definition 11) that uses a $\mathsf{MBB}_{\mathsf{FT}}$ protocol (Definition 5).*

For brevity, we will henceforth use BTC to refer to *blockchain-native threshold cryptosystem*, and $\mathsf{BTC}_{\mathsf{FT}}$ to refer to *blockchain-native threshold cryptosystem with finalization threshold.*

As some of the results in the paper hold under optimistic conditions, we formally define these optimistic conditions next.

Definition 14 (Error-free Execution). *We define an execution of a BTC protocol as error-free if all parties are honest.*

Definition 15 (Synchronous Execution). *We define an execution of a protocol $\mathsf{BTC}_{\mathsf{FT}} = (\mathsf{MBB}_{\mathsf{FT}}, \mathsf{TC})$ as synchronous for a round r if all honest parties prefinalize the same message $m \in \mathcal{M}$ for r at the same physical time in $\mathsf{MBB}_{\mathsf{FT}}$.*

Definition 16 (Optimistic Execution). *We define an execution of a $\mathsf{BTC}_{\mathsf{FT}}$ protocol as optimistic if the execution is error-free and synchronous for any round r, and all messages have the same delay.*

4.2 A Strawman Protocol

As a warm-up, we first describe a strawman protocol for any blockchain-native threshold cryptosystem $\mathsf{BTC} = (\mathsf{MBB}, \mathsf{TC})$. The protocol has a latency $\mathsf{L}_r \geq \delta$ for any round $r \in \mathbb{N}$ even in executions that are error-free and have the same message delay δ between honest parties. To the best of our knowledge, all existing blockchain-native threshold cryptosystem follow this approach. For example, Dfinity [31] blockchain adopts this approach for generating distributed randomness, and thus incurs a latency of at least one message delay. We present the strawman protocol in Algorithm 1, which works for both tight and ramp thresholds. For brevity, we refer to this protocol as the *slow path*.

As part of the setup phase, each party i receives $(\{\mathsf{pk}\}_{i\in[n]}, [\![s]\!]_i)$, where $[\![s]\!]_i$ is the secret share of party i and $\{\mathsf{pk}\}_{i\in[n]}$ is the vector of threshold public keys of all parties. Each party maintains a FIFO *queue* to record the finalized rounds awaiting the TC output. These rounds are pushed into the FIFO *queue* in the order they are finalized, and only the top of the FIFO *queue* will pop and be output. Looking ahead, this

Algorithm 1 Slow Path for Blockchain-Native Threshold Cryptosystem

SETUP:

let $\boldsymbol{m} \leftarrow \{\}$, $\boldsymbol{\sigma} \leftarrow \{\}$ ▷ Maps that store outputs for rounds
let $queue \leftarrow \{\}$ ▷ A FIFO queue that stores the finalized rounds
let $(\{\mathsf{pk}_j\}_{j\in[n]}, [\![s]\!]_i) \leftarrow \mathsf{ShareGen}(s)$ for thresholds $t+1 \le t_{\mathsf{sec}} \le t_{\mathsf{rec}} \le n-t$

SLOW PATH:

upon $\mathsf{finalize}(r, m)$ **do**
  let $\boldsymbol{m}[r] \leftarrow m$
  $queue.push(r)$
  let $\sigma_i \leftarrow \mathsf{PEval}([\![s]\!]_i, (r, m))$
  send $(\texttt{SHARE}, r, m, \sigma_i)$ to all parties

RECONSTRUCTION:

upon receiving $(\texttt{SHARE}, r, m, \sigma_j)$ from party j **do**
  if $\mathsf{PVer}(\mathsf{pk}_j, (r, m), \sigma_j) = 1$ **then**
    $S_{r,m} \leftarrow S_{r,m} \cup \{j\}$
    if $|S_{r,m}| \ge t_{\mathsf{rec}}$ and $\boldsymbol{\sigma}[r] = \{\}$ **then**
      let $\boldsymbol{\sigma}[r] \leftarrow \mathsf{Comb}(S_{r,m}, (r, m), \{(\mathsf{pk}_i, \sigma_i)\}_{i\in S_{r,m}})$

OUTPUT:

upon $\boldsymbol{\sigma}[queue.top()] \ne \{\}$ **do** ▷ Always running in the background
  let $r \leftarrow queue.pop()$
  output $(r, \boldsymbol{m}[r], \boldsymbol{\sigma}[r])$

ensures Total Ordering, even when parties reconstruct TC outputs of different rounds in out of order. Each party additionally maintains two maps $\boldsymbol{m}$ and $\boldsymbol{\sigma}$ to store the finalized message and TC output shares of each parties of each round, respectively.

In the strawman protocol, for any given round r, each party i waits until a message m is finalized by the MBB protocol in round r. Upon finalization, each party i computes its TC output share $\sigma_i := \mathsf{PEval}([\![s]\!]_i, (r, m))$ and sends the $\texttt{SHARE}$ message $(\texttt{SHARE}, r, m, \sigma_i)$ to all parties. Party i also adds round r to $queue$ and updates $\boldsymbol{m}$ as $\boldsymbol{m}[r] := m$. Next, upon receiving $(\texttt{SHARE}, r, m, \sigma_j)$ from party j, party i first validates σ_j using PVer algorithm and adds σ_j to the set $S_{r,m}$ upon successful validation. Finally, upon receiving t_{rec} valid $\texttt{SHARE}$ messages for (m, r), party i computes the TC output σ using Comb algorithm and updates $\boldsymbol{\sigma}$ as $\boldsymbol{\sigma}[r] = \sigma$. Whenever party i has the TC output of round r that is the top of $queue$, party i pops the queue and outputs the result $(r, \boldsymbol{m}[r], \boldsymbol{\sigma}[r])$ for round r.

To ensure the Termination property in the strawman protocol, the reconstruction threshold must be greater than $n - t$, i.e., $t_{\mathsf{rec}} \le n - t$. Intuitively, this ensures that once the MBB outputs in a round, every honest party receives a sufficient number of TC output shares to reconstruct the TC output. Additionally, for Secrecy for the strawman protocol, the secrecy threshold must be greater than the number of TC shares controlled by the adversary, i.e., $t_{\mathsf{sec}} \ge t + 1$. Intuitively, this prevents the adversary from reconstructing the TC output on its own.

Analysis of the strawman protocol. The correctness of the protocol is straightforward and is omitted here for brevity.

We will now argue that the slow path has a latency overhead of at least δ even in executions that are error-free and have the same message delay δ between honest parties. More specifically, for any round r, any party needs to receive at least $t_{\mathsf{rec}} - |\mathcal{C}| \ge 1$ shares from the honest parties to compute σ. Consider the first honest party i that outputs $\mathsf{finalize}(r, m)$. In the strawman protocol, party i needs to wait for at least one message delay starting from finalization to receive the shares from the honest parties to compute σ, since other honest parties only send shares after finalization. This means $\mathsf{L}_r \ge \delta$. As mentioned in Section 1, the

Algorithm 2 Blockchain-Native Threshold Cryptosystem with Tight Thresholds

SETUP:

let $\boldsymbol{m} \leftarrow \{\}$, $\boldsymbol{\sigma} \leftarrow \{\}$ ▷ Maps that store outputs for rounds
let $queue \leftarrow \{\}$ ▷ A FIFO queue that stores the finalized rounds
let $(\{\mathsf{pk}_j\}_{j\in[n]}, [\![s]\!]_i) \leftarrow \mathsf{ShareGen}(s)$ for thresholds $t_{\mathsf{sec}} = t_{\mathsf{rec}} = t_{\mathsf{fin}}$ ▷ t_{fin} is the finalization threshold of $\mathsf{MBB}_{\mathsf{FT}}$

PREFINALIZATION:

upon $\mathsf{prefinalize}(r, m)$ **do**
  let $\sigma_i \leftarrow \mathsf{PEval}([\![s]\!]_i, (r, m))$
  send $(\texttt{PREFIN}, r, m)$ and $(\texttt{SHARE}, r, m, \sigma_i)$ to all parties
upon receiving $(\texttt{PREFIN}, r, m)$ from party j **do**
  $T_{r,m} \leftarrow T_{r,m} \cup \{j\}$
  if $|T_{r,m}| \geq t_{\mathsf{fin}}$ and $\boldsymbol{m}[r] = \{\}$ **then**
    call $\mathsf{finalize}(r, m)$
upon $\mathsf{finalize}(r, m)$ **do**
  let $\boldsymbol{m}[r] \leftarrow m$
  $queue.push(r)$
  if $(\texttt{SHARE}, r, *, *)$ not sent **then**
    let $\sigma_i \leftarrow \mathsf{PEval}([\![s]\!]_i, (r, m))$
    send $(\texttt{SHARE}, r, m, \sigma_i)$ to each other party

RECONSTRUCTION and OUTPUT are same as Algorithm 1.

latency of MBB can be as short as three message delays, so adding an additional message delay to the system represents a significant overhead.

5 Blockchain-Native Threshold Cryptosystem with Tight Thresholds

In this section, we present a protocol for $\mathsf{BTC}_{\mathsf{FT}} = (\mathsf{MBB}_{\mathsf{FT}}, \mathsf{TC})$ (Definition 13) for tight thresholds that has low latency. In any given round $r \in \mathbb{N}$ and in any execution, the global finalization time of our protocol is same as the global reconstruction time, i.e., $\mathsf{GFT}_r = \mathsf{GRT}_r$. Moreover, in error-free executions, honest parties in our protocol learns the TC output simultaneously with the $\mathsf{MBB}_{\mathsf{FT}}$ output, i.e., $\mathsf{L}_r = 0$. We summarize our construction in Algorithm 2 and describe it next.

5.1 Design

The setup phase is identical to that of Algorithm 1, except that the secrecy and reconstruction thresholds are set to be equal to the finalization threshold of $\mathsf{MBB}_{\mathsf{FT}}$. Note that, the Termination property of $\mathsf{MBB}_{\mathsf{FT}}$ requires $t_{\mathsf{fin}} \leq n - t$, as honest parties needs to finalize a message even when corrupted parties do not send any throughout the protocol. Next, unlike Algorithm 1, parties reveal their TC output shares when they prefinalize a message. More specifically, for every round r, each party i computes $\sigma_i := \mathsf{PEval}([\![s]\!]_i, (r, m))$ upon prefinalizing the value (r, m), and sends the $\texttt{SHARE}$ message $(\texttt{SHARE}, r, m, \sigma_i)$ to all parties in addition to sending the $\texttt{PREFIN}$ message $(\texttt{PREFIN}, r, m)$. When a party receives t_{fin} $\texttt{PREFIN}$ messages $(\texttt{PREFIN}, r, m)$, it finalizes the message m for round r, by adding round r to *queue* and recording m in $\boldsymbol{m}[r]$. The party also computes and sends σ_i if it has not done so. The reconstruction and output phases are also identical to Algorithm 1, where parties collect and combine shares to generate TC output, and output the result round-by-round.

5.2 Analysis of Algorithm 2

Theorem 3. *Algorithm 2 implements a blockchain-native threshold cryptosystem and guarantees the Agreement, Termination, Validity, Total Order, and Secrecy properties.*

Proof. Let $\mathsf{BTC}_{\mathsf{FT}} = (\mathsf{MBB}_{\mathsf{FT}}, \mathsf{TC})$ denote the protocol.

Secrecy. We first prove that, for any round $r \in \mathbb{N}$, if an honest party outputs (r, m, σ) then $\sigma = \mathsf{Eval}(s, (r, m))$. For the sake of contradiction, suppose that for some $r \in \mathbb{N}$, an honest party outputs (r, m, σ') where $\sigma' \neq \mathsf{Eval}(s, (r, m))$. By the protocol, the party outputs $\mathsf{finalize}(r, m)$ in $\mathsf{MBB}_{\mathsf{FT}}$. Also, according to the protocol and the Robustness property of TC, $\sigma' = \mathsf{Eval}(s, (r, m'))$ for some $m' \neq m$. This implies, by the Unpredictability property of TC, that at least $t_{\mathsf{sec}} - |\mathcal{C}|$ honest parties have revealed their TC output shares $\mathsf{PEval}([\![s]\!], (r, m'))$. Let $T_{m'}$ be set indicating these honest parties. Note that in Algorithm 2 an honest party reveals its TC output share for any (r, m') only upon prefinalizing (r, m') or finalizing (r, m'). Consider the latter case: if any party $i \in T_{m'}$ finalizes a message (r, m') for $m' \neq m$, then this violates the Agreement property of $\mathsf{MBB}_{\mathsf{FT}}$, and hence a contradiction. This implies that all parties in $T_{m'}$ has called $\mathsf{prefinalize}(r, m')$ in $\mathsf{MBB}_{\mathsf{FT}}$. However, since $\mathsf{MBB}_{\mathsf{FT}}$ has the finalization threshold t_{fin}, $t_{\mathsf{fin}} = t_{\mathsf{sec}}$ and $|T_{m'}| \geq t_{\mathsf{sec}} - |\mathcal{C}| = t_{\mathsf{fin}} - |\mathcal{C}|$ implies that the message m' is globally finalized. This again violates the Agreement property of $\mathsf{MBB}_{\mathsf{FT}}$, hence a contradiction. Therefore, for any round $r \in \mathbb{N}$, if an honest party outputs (r, m, σ) then $\sigma = \mathsf{Eval}(s, (r, m))$.

For any corrupted party, the same argument above applies; thus the adversary cannot learn $\mathsf{Eval}(s, (r, m'))$ where $m' \neq m$

Agreement. Suppose an honest party outputs (r, m, σ) and another honest party outputs (r, m', σ'). The Agreement property of $\mathsf{MBB}_{\mathsf{FT}}$ ensures that $m = m'$. Therefore, by the Secrecy property of $\mathsf{BTC}_{\mathsf{FT}}$ above, we get $\sigma = \sigma'$.

Termination. The Termination property of $\mathsf{MBB}_{\mathsf{FT}}$ requires that $t_{\mathsf{fin}} \leq n - t$, since the honest party needs to finalize the message even when the corrupted parties all remain slient. After GST, by the Agreement and Termination property of $\mathsf{MBB}_{\mathsf{FT}}$, for any round $r \in \mathbb{N}$ all honest parties eventually output the same $\mathsf{finalize}(r, m)$ where $m \in \mathcal{M} \cup \{\perp\}$. Then, eventually, all $n - t$ honest parties send their $\mathsf{PEval}([\![s]\!], (r, m))$ to all parties according to the prefinalization step of the protocol. By the Robustness property of TC, all honest parties can eventually reconstruct $\sigma = \mathsf{Eval}(s, (r, m))$ since $t_{\mathsf{rec}} = t_{\mathsf{fin}} \leq n - t$. Therefore, after GST, each honest party eventually outputs (r, m, σ) for some σ.

Validity. By the Validity property of $\mathsf{MBB}_{\mathsf{FT}}$, all honest parties eventually output $\mathsf{finalize}(r, m)$ in $\mathsf{MBB}_{\mathsf{FT}}$, and will eventually output (r, m, σ') by Termination of BTC. Since if any honest party outputs (r, m, σ') then $\sigma' = \mathsf{Eval}(s, (r, m))$ by the Secrecy property of BTC, we conclude that all honest parties eventually output $(r, m, \mathsf{Eval}(s, (r, m)))$.

Total Order. In the protocol, an honest party always outputs (r, m, σ) according to the order of r in the FIFO *queue*. Suppose that an honest party outputs (r, m, σ) before (r', m', σ') in BTC, then the party enqueues r before r'. This implies that the honest party outputs $\mathsf{finalize}(r, m)$ before $\mathsf{finalize}(m', r')$ in $\mathsf{MBB}_{\mathsf{FT}}$. By the Total Order property of $\mathsf{MBB}_{\mathsf{FT}}$, we conclude $r < r'$ since an honest party outputs $\mathsf{finalize}(r, m)$ before $\mathsf{finalize}(m', r')$. Therefore, the protocol satisfies Total Order: if an honest party outputs (r, m, σ) before (r', m', σ'), then $r < r'$.

□

Theorem 4. *Algorithm 2 achieves* $\mathsf{GFT}_r = \mathsf{GRT}_r$ *for any round* $r \in \mathbb{N}$ *in any execution.*

Proof. Recall from Section 3, for any round r, GFT_r and GRT_r are the global finalization time and global reconstruction time of round r, respectively. Consider any execution. By the definition of GFT_r, $t_{\mathsf{fin}} - |\mathcal{C}|$ honest parties have called $\mathsf{prefinalize}(r, m)$ at time GFT_r. According to Algorithm 2, these honest parties have also revealed their TC output shares for (r, m) at time GFT_r. Since, $t_{\mathsf{rec}} = t_{\mathsf{fin}}$, this implies that the adversary can adversary to reconstruct the TC output at time GFT_r, and hence $\mathsf{GRT}_r \leq \mathsf{GFT}_r$. From Lemma 1, we have that $\mathsf{GRT}_r \geq \mathsf{GFT}_r$. Therefore, we conclude that $\mathsf{GFT}_r = \mathsf{GRT}_r$ for any r in any execution.

□

Theorem 5. *Algorithm 2 achieves* $\mathsf{L}_r = 0$ *for any round* $r \in \mathbb{N}$ *in any error-free execution.*

Proof. Recall from Section 3, in any round r, L_r is the maximum time difference, across all honest parties, between the time a honest party i finalizes (m,r) from MBB and the time the same honest party i outputs $(m,r,\mathsf{Eval}(s,(m,r))$. For any fixed round r, consider any error-free execution. By the definition of $\mathsf{MBB}_{\mathsf{FT}}$, an honest party finalizes a message once it receives PREFIN messages from t_{fin} parties. Since all parties in an error-free execution are honest, these parties also send their TC output share $\mathsf{PEval}([\![s]\!]_i,(r,m))$ upon prefinalizing (r,m). Since $t_{\mathsf{fin}} = t_{\mathsf{rec}}$, any honest party simultaneously receives t_{rec} valid shares of the form $\mathsf{PEval}([\![s]\!],(r,m))$ and t_{fin} PREFIN messages for $\mathsf{MBB}_{\mathsf{FT}}$. Therefore, every honest party finalizes (m,r) and computes $\boldsymbol{\sigma}[r] = \mathsf{Eval}([\![s]\!],(r,m))$ for any round r, simultaneously.

Now we prove the theorem by induction on the round number r. For the base case, consider $r = 0$. We have shown that any honest party computes $\boldsymbol{\sigma}[r]$ and finalizes (r,m) at the same time, it also outputs $(r,\boldsymbol{m}[r],\boldsymbol{\sigma}[r])$ at the same time since $r = 0$ is at the top of *queue* by the Total Ordering property of $\mathsf{MBB}_{\mathsf{FT}}$. Thus, $\mathsf{L}_0 = 0$. For the induction steps, assume that the theorem is true up to round $r = k-1$, that is, $\mathsf{L}_r = 0$ for $r = 0,...,k-1$. Now consider round $r = k$. Similarly, any honest party computes $\boldsymbol{\sigma}[r]$ and finalizes (r,m) at the same time. Due to the Total Ordering property of $\mathsf{MBB}_{\mathsf{FT}}$, any rounds pushed in *queue* before $r = k$ must be smaller than k, and they have been popped when (r,m) is finalized since $\mathsf{L}_r = 0$ for $r = 0,...,k-1$. Therefore, $r = k$ is at the top of *queue* and the party can output immediately, thus $\mathsf{L}_k = 0$. Therefore, by induction, the theorem holds. ◻

6 Blockchain-Native Threshold Cryptography with Ramp Thresholds

In this section, we present the results for $\mathsf{BTC}_{\mathsf{FT}} = (\mathsf{MBB}_{\mathsf{FT}}, \mathsf{TC})$ (Definition 13) with ramp thresholds $t_{\mathsf{sec}} < t_{\mathsf{rec}}$. We first show in Section 6.1 an impossibility result that, for any round r, for any $\mathsf{BTC}_{\mathsf{FT}}$ with ramp thresholds, there must exist an execution where $\mathsf{GRT}_r > \mathsf{GFT}_r$. To circumvent the impossibility result, in Section 6.2 we propose an optimistic protocol that achieves $\mathsf{GRT}_r = \mathsf{GFT}_r$ in synchronous executions and $\mathsf{L}_r = 0$ in optimistic executions.

6.1 Impossibility

First, we demonstrate the impossibility result, which says that no $\mathsf{BTC}_{\mathsf{FT}}$ protocol with ramp thresholds $t_{\mathsf{sec}} < t_{\mathsf{rec}}$ can always guarantee that global finalization and reconstruction occur simultaneously.

Theorem 6. *For any* $\mathsf{BTC}_{\mathsf{FT}} = (\mathsf{MBB}_{\mathsf{FT}}, \mathsf{TC})$ *with ramp thresholds, there always exists some execution where* $\mathsf{GRT}_r > \mathsf{GFT}_r$ *for each round* $r \in \mathbb{N}$.

Proof. From Lemma 1, we have that in all executions, in each round $r \in \mathbb{N}$, $\mathsf{GRT}_r \geq \mathsf{GFT}_r$.

For the sake of contradiction, suppose that there exists a protocol $\mathsf{BTC}_{\mathsf{FT}} = (\mathsf{MBB}_{\mathsf{FT}}, \mathsf{TC})$ with ramp thresholds, such that for any given round r, in all executions we have $\mathsf{GRT}_r = \mathsf{GFT}_r$. Let $\mathcal{E}$ be one such execution and let $\tau_{\mathcal{E}} = \mathsf{GRT}_r = \mathsf{GFT}_r$. Let s denote the secret shared between the parties. Since $\mathsf{MBB}_{\mathsf{FT}}$ has the finalization threshold t_{fin}, at least $t_{\mathsf{fin}} - |\mathcal{C}|$ honest parties have called $\mathsf{prefinalize}(r,m)$ at $\tau_{\mathcal{E}}$. Similarly, by the definition of GRT_r, at least $t_{\mathsf{rec}} - |\mathcal{C}|$ honest parties have revealed their TC output shares $\mathsf{PEval}([\![s]\!],(r,m))$ at $\tau_{\mathcal{E}}$. Without loss of generality, we can assume that exactly $t_{\mathsf{fin}} - |\mathcal{C}|$ honest parties have called $\mathsf{prefinalize}(r,m)$ and $t_{\mathsf{rec}} - |\mathcal{C}|$ honest parties have revealed their TC output shares at $\tau_{\mathcal{E}}$. Let h be any honest party that called $\mathsf{prefinalize}(r,m)$ at time $\tau_{\mathcal{E}}$ in the execution $\mathcal{E}$. There are two cases, which we denote with (1) and (2) below.

(1) h also reveals its TC output share $\mathsf{PEval}([\![s]\!]_h,(r,m))$ at time $\tau_{\mathcal{E}}$. Consider another execution $\mathcal{E}'$ identical to $\mathcal{E}$ up to time $\tau_{\mathcal{E}}$ with the only difference that, h calls $\mathsf{prefinalize}(r,m)$ and reveals $\mathsf{PEval}([\![s]\!]_h,(r,m))$ at time time $\tau_{\mathcal{E}} + \epsilon$ for some $\epsilon > 0$ (say due to asynchrony in computation or communication). Therefore, $\tau_{\mathcal{E}} + \epsilon$ is the global finalization time of execution $\mathcal{E}'$.
Now, consider the time $\tau_{\mathcal{E}}$ in the new execution $\mathcal{E}'$. The message m is not globally finalized at $\tau_{\mathcal{E}}$ for r since only $t_{\mathsf{fin}} - |\mathcal{C}| - 1$ honest parties have called $\mathsf{prefinalize}(r,m)$. However, since $t_{\mathsf{rec}} > t_{\mathsf{sec}}$, there are $t_{\mathsf{rec}} - |\mathcal{C}| - 1 \geq t_{\mathsf{sec}} - |\mathcal{C}|$ honest parties that have revealed their TC output shares $\mathsf{PEval}([\![s]\!],(r,m))$. This implies that in execution $\mathcal{E}'$ the adversary $\mathcal{A}$ can compute the TC output $\mathsf{Eval}(s,(m,r))$ at $\tau_{\mathcal{E}}$, which is earlier than its global finalization time $\tau_{\mathcal{E}} + \epsilon$. This contradicts Lemma 1.

(2) h does not reveal its TC output share at time $\tau_{\mathcal{E}}$. Similarly, consider another execution $\mathcal{E}'$ identical to $\mathcal{E}$ up to time $\tau_{\mathcal{E}}$ with the only difference that, h calls $\mathsf{prefinalize}(r, m)$ at time $\tau_{\mathcal{E}} + \epsilon$ for some $\epsilon > 0$ (say due to asynchrony in computation or communication). Therefore, for the new execution $\mathcal{E}'$, $\tau_{\mathcal{E}} + \epsilon$ is the global finalization time, and the global reconstruction time remains $\tau_{\mathcal{E}}$. Then, for the new execution $\mathcal{E}'$, the global reconstruction comes before the global finalization, contradicting Lemma 1. □

Theorem 6 states that for any blockchain-native threshold cryptosystem with finalization threshold and ramp thresholds, there always exists an execution where global reconstruction occurs after global finalization for each round. In fact, existing solutions for BTC with ramp thresholds all have a latency of at least one message delay, such as Das et al. [23].

6.2 Fast Path

Theorem 6 claims that no $\mathsf{BTC}_{\mathsf{FT}}$ protocol with ramp thresholds can *always* guarantee that the global finalization and reconstruction occur simultaneously, implying that the latency of $\mathsf{BTC}_{\mathsf{FT}}$ may be unavoidable. Fortunately, we can circumvent this impossibility result in optimistic executions. In this section, we describe a simple protocol named *fast path* that, for any round r, achieves $\mathsf{GRT}_r = \mathsf{GFT}_r$ under synchronous executions (Definition 15), and $\mathsf{L}_r = 0$ under optimistic executions (Definition 16). As we illustrate in Section 7.3, in practice, our new protocol achieves significantly lower latency compared to the strawman protocol.

The key observation from Theorem 6 is that to ensure the same global finalization and reconstruction time, the secrecy threshold cannot be lower than the finalization threshold; otherwise, the TC output could be revealed before $\mathsf{MBB}_{\mathsf{FT}}$ finalizes a message. A naive way to address this is to increase the secrecy threshold to match the finalization threshold, i.e., $t_{\mathsf{sec}} = t_{\mathsf{fin}}$. However, the problem with this approach is that, since the threshold is ramped, $t_{\mathsf{rec}} > t_{\mathsf{sec}} = t_{\mathsf{fin}} = n - t$ ‖, there may not be enough honest shares for parties to reconstruct TC output upon finalizing the $\mathsf{MBB}_{\mathsf{FT}}$ output, violating the Termination property.

We address this issue as follows: First, we share the TC secret s among the parties twice, using independent randomness, with two different pairs of thresholds $(t_{\mathsf{sec}}, t_{\mathsf{rec}})$ and $(t'_{\mathsf{rec}}, t'_{\mathsf{sec}})$. In particular, we choose the thresholds $(t_{\mathsf{sec}}, t_{\mathsf{rec}})$ with the constraint that $t_{\mathsf{sec}} \geq t + 1$ and $t_{\mathsf{rec}} \leq n - t$. Similarly, we choose $(t'_{\mathsf{sec}}, t'_{\mathsf{rec}})$ with the constraint that $t'_{\mathsf{sec}} \geq t_{\mathsf{fin}}$ and $t'_{\mathsf{rec}} \leq n$. Let $\{[\![s]\!]\}_{i \in [n]}$ and $\{[\![s]\!]'\}_{i \in [n]}$ be the secret shares of s with thresholds $(t_{\mathsf{sec}}, t_{\mathsf{rec}})$ and $(t'_{\mathsf{sec}}, t'_{\mathsf{rec}})$, respectively. Second, we add an *fast path*, where parties reveal their TC output shares they compute with $\{[\![s]\!]'\}_{i \in [n]}$ immediately upon prefinalizing a message.

We present our final protocol in Algorithm 3 and discuss it next. The setup phase is similar to Algorithm 1, except that the same secret s is shared twice, using independent randomness, for the slow path and fast path, respectively. For any round r, each party does the following:

- *Fast path:* For the fast path, when a party prefinalizes a message m for round r, it reveals its TC output share $\mathsf{PEval}([\![s]\!]'_i, (r, m))$. Once a party receives t'_{rec} verified shares of the fast path, it reconstructs the TC output.
- *Slow path:* For the slow path, upon $\mathsf{MBB}_{\mathsf{FT}}$ finalization for message m and round r, each party i reveals its TC output share $\mathsf{PEval}([\![s]\!]_i, (r, m))$. Next, any party who has not received t'_{rec} verified TC output shares from the fast path waits to receive t_{rec} verified TC output shares from the slow path. Once the party receives t_{rec} verified shares of the slow path, it reconstructs the TC output.

Lastly, similarly to the output phase of Algorithm 1, to guarantee Total Order, the parties push the finalized rounds into the First-in-first-out (FIFO) *queue* and output the result round-by-round once either the fast path or slow path has reconstructed the TC output. So the latency of the protocol is the minimum latency of the two paths.

Note that, a party reveals its share of the slow path even if it has already revealed its share of the fast path or reconstructed the TC output from the fast path. This is crucial for ensuring Termination, because with corrupted parties sending their TC output shares to only a subset $\mathcal{S}$ of honest parties, it is possible that only parties in $\mathcal{S}$ can reconstruct the TC output from the fast path. If the honest parties in $\mathcal{S}$ do not

‖ $\mathsf{MBB}_{\mathsf{FT}}$ protocols with optimal resilience typically have $t_{\mathsf{fin}} = n - t$ to ensure quorum intersection for safety.

reveal their TC output shares of the slow path, the remaining honest parties would be unable to reconstruct the TC output, thereby losing the Termination guarantee.

Algorithm 3 Fast Path for Blockchain-Native Threshold Cryptosystem with Ramp Thresholds

SETUP:

let $\boldsymbol{m} \leftarrow \{\}$, $\boldsymbol{\sigma} \leftarrow \{\}$ ▷ Maps that store outputs for rounds
let $queue \leftarrow \{\}$ ▷ A FIFO queue that stores the finalized rounds
let $(\{\mathsf{pk}_j\}_{j\in[n]}, [\![s]\!]_i) \leftarrow \mathsf{ShareGen}(s)$ for thresholds $t+1 \leq t_{\mathsf{sec}} < t_{\mathsf{rec}} \leq n-t$ ▷ For slow path
let $(\{\mathsf{pk}'_j\}_{j\in[n]}, [\![s]\!]'_i) \leftarrow \mathsf{ShareGen}(s)$ for thresholds $t'_{\mathsf{sec}} = t_{\mathsf{fin}} < t'_{\mathsf{rec}} \leq n$ ▷ For fast path

FAST PATH:

upon $\mathsf{prefinalize}(r, m)$ **do**
  let $\sigma'_i \leftarrow \mathsf{PEval}([\![s]\!]'_i, (r, m))$
  send $(\texttt{PREFIN}, r, m)$ and $(\texttt{FAST-SHARE}, r, m, \sigma'_i)$ to each other party

SLOW PATH:

upon receiving $(\texttt{PREFIN}, r, m)$ from party j **do**
  $T_{r,m} \leftarrow T_{r,m} \cup \{j\}$
  if $|T_{r,m}| \geq t_{\mathsf{fin}}$ and $\boldsymbol{m}[r] = \{\}$ **then**
    call $\mathsf{finalize}(r, m)$
upon $\mathsf{finalize}(r, m)$ **do**
  let $\boldsymbol{m}[r] \leftarrow m$
  $queue.push(r)$
  let $\sigma_i \leftarrow \mathsf{PEval}([\![s]\!]_i, (r, m))$
  send $(\texttt{SLOW-SHARE}, r, m, \sigma_i)$ to each other party

RECONSTRUCTION:

upon receiving $(\texttt{FAST-SHARE}, r, m, \sigma'_j)$ from party j **do** ▷ Fast path reconstruction
  if $\mathsf{PVer}(\mathsf{pk}'_j, (r, m), \sigma'_j) = 1$ **then**
    $S'_{r,m} \leftarrow S'_{r,m} \cup \{j\}$
    if $|S'_{r,m}| \geq t'_{\mathsf{rec}}$ and $\boldsymbol{\sigma}[r] = \{\}$ **then**
      let $\boldsymbol{\sigma}[r] \leftarrow \mathsf{Comb}(S'_{r,m}, (r, m), \{(\mathsf{pk}'_i, \sigma'_i)\}_{i\in S'_{r,m}})$
upon receiving $(\texttt{SLOW-SHARE}, r, m, \sigma_j)$ from party j **do** ▷ Slow path reconstruction
  if $\mathsf{PVer}(\mathsf{pk}_j, (r, m), \sigma_j) = 1$ **then**
    $S_{r,m} \leftarrow S_{r,m} \cup \{j\}$
    if $|S_{r,m}| \geq t_{\mathsf{rec}}$ and $\boldsymbol{\sigma}[r] = \{\}$ **then**
      let $\boldsymbol{\sigma}[r] \leftarrow \mathsf{Comb}(S_{r,m}, (r, m), \{(\mathsf{pk}_i, \sigma_i)\}_{i\in S_{r,m}})$

OUTPUT is same as Algorithm 1

6.3 Proof of Correctness

Theorem 7. *Algorithm 3 implements a blockchain-native threshold cryptosystem and guarantees the Agreement, Termination, Validity, Total Order, and Secrecy properties.*

Proof. Let $\mathsf{BTC}_{\mathsf{FT}} = (\mathsf{MBB}_{\mathsf{FT}}, \mathsf{TC})$ denote the protocol. The proofs of Agreement, Termination, Validity and Total Order properties are identical to that of Theorem 3. Thus, we focus on the Secrecy property.

Secrecy. We first prove that, for any round $r \in \mathbb{N}$, if an honest party outputs (r, m, σ) then $\sigma = \mathsf{Eval}(s, (r, m))$. For the sake of contradiction, suppose that for some $r \in \mathbb{N}$, an honest party outputs (r, m, σ') where $\sigma' \neq$

$\mathsf{Eval}(s,(r,m))$. By the protocol, the honest party outputs $\mathsf{finalize}(r,m)$ in $\mathsf{MBB_{FT}}$. Also, as per the protocol and the Robustness property of TC, $\sigma' = \mathsf{Eval}(s,(r,m'))$ for some $m' \neq m$. There are two cases:

First, if σ' is obtained from the slow path, then by the Unpredictability property of TC, at least $t_{\mathsf{sec}} - |\mathcal{C}| \geq 1$ honest parties have revealed their $\mathsf{PEval}([\![s]\!],(r,m'))$ for the slow path. According to the protocol, these honest parties output $\mathsf{finalize}(r,m')$, which violates the Agreement property of $\mathsf{MBB_{FT}}$ since an honest party outputs $\mathsf{finalize}(r,m)$, contradiction.

Second, if σ' is obtained from the fast path, then by the Unpredictability property of TC, at least $t'_{\mathsf{sec}} - |\mathcal{C}|$ honest parties reveal their $\mathsf{PEval}([\![s]\!]',(r,m'))$ for the fast path. According to the protocol, these honest parties call $\mathsf{prefinalize}(r,m')$ in $\mathsf{MBB_{FT}}$. Since $t_{\mathsf{fin}} = t'_{\mathsf{sec}}$, the message m' is globally finalized since $\mathsf{MBB_{FT}}$ has the finalization threshold t_{fin}. Again, the Agreement property of $\mathsf{MBB_{FT}}$ is violated, contradiction. Therefore, for any round $r \in \mathbb{N}$, if an honest party outputs (r,m,σ) then $\sigma = \mathsf{Eval}(s,(r,m))$.

For any corrupted party, the same argument above applies; thus the adversary cannot learn $\mathsf{Eval}(s,(r,m'))$ where $m' \neq m$ □

Theorem 8. *Algorithm 3 achieves $\mathsf{GFT}_r = \mathsf{GRT}_r$ for round r if the execution is synchronous for r.*

Proof. Consider any synchronous execution. By Definition 15, all honest parties prefinalize the same message m for round r at the same physical time GFT_r. Thus, as per Algorithm 3, all honest parties reveal their TC output shares for (r,m) at time GFT_r in the fast path, which allows the adversary to reconstruct the TC output at time GFT_r, which implies $\mathsf{GRT}_r \leq \mathsf{GFT}_r$. By Lemma 1, $\mathsf{GRT}_r \geq \mathsf{GFT}_r$. Therefore, we conclude that $\mathsf{GFT}_r = \mathsf{GRT}_r$ for any r in any synchronous execution. □

Theorem 9. *Algorithm 3 achieves $\mathsf{L}_r = 0$ for any round $r \in \mathbb{N}$ in any optimistic execution.*

Proof. Consider any optimistic execution and any round $r \in \mathbb{N}$. As per Algorithm 3, honest parties prefinalize a message and send its TC output share for the fast path simultaneously. Recall from Definition 16, in optimistic executions, all messages have the same delay, and all parties are honest and finalize the same message at the same physical time. Therefore, all honest parties locally receive t_{fin} `PREFIN` messages and t_{rec} TC output shares for (r,m) simultaneously. This implies that the local finalization and reconstruction occur simultaneously. Then, by a similar induction argument as in Theorem 5, $\mathsf{L}_r = 0$ for any round $r \in \mathbb{N}$ in any optimistic execution. □

In practice, different honest parties may prefinalize at different physical times due to a lack of synchrony. Moreover, honest parties may need to wait slightly longer after local finalization to receive additional shares from the fast path since the reconstruction threshold of the fast path is higher than the finalization threshold. Despite this, the latency of Algorithm 3 remains significantly lower than one message delay. The evaluation in Section 7.3 demonstrates that the fast path reduces the latency overhead by 71% compared to the slow path (Algorithm 1), which has a latency of one message delay.

7 Distributed Randomness: A Case Study

In this section, we implement and evaluate distributed randomness as a concrete example of blockchain-native threshold cryptosystem, to demonstrate the effectiveness of our solution in reducing the latency for real-world blockchains. We implement the fast path (Algorithm 3) for Das et al. [23], which is a distributed randomness scheme designed for proof-of-stake blockchains and is deployed in the Aptos blockchain [10]. We then compare our latency (using both micro-benchmarks and end-to-end evaluation) with the [23], that implements the strawman protocol (Algorithm 1).

In the rest of the section, we first provide the brief overview of [23] necessary to describe the implementation of our fast-path protocol atop their scheme.

7.1 Overview of Das et al. [23]

Das et al. [23] is a distributed randomness protocol for proof-of-stake blockchains where each party has a stake, and the blockchain is secure as long as the adversary corrupts parties with combined stake less than 1/3-th of the total stake.

Rounding. Note that the total stake in practice can be very large. For example, as of July 2024, the total stake in the Aptos blockchain [3] exceeds 8.7×10^8. Since a distributed randomness protocol with such a large total stake will be prohibitively expensive,[23] assigns approximate stakes of parties to a much smaller value called *weights*, and this process is called *rounding* **. Briefly, the rounding algorithm in [23] defines a function $\mathsf{Round}(\boldsymbol{S}, t_{\mathsf{sec}}, t_{\mathsf{rec}}) \to (\boldsymbol{W}, w)$ that inputs the stake distribution $\boldsymbol{S}$ of the parties before rounding, the secrecy and reconstruction threshold $t_{\mathsf{sec}}, t_{\mathsf{rec}}$ (in stakes), and outputs the weight distribution $\boldsymbol{W}$ of the parties after rounding, and the weight threshold w. The rounding algorithm guarantees that any subset of parties with a combined stake $< t_{\mathsf{sec}}$ will always have a combined weight $< w$, thus preventing them from reconstructing the TC output; and any subset of parties with a combined stake $\geq t_{\mathsf{rec}}$ will always have a combined weight $\geq w$, allowing them to reconstruct the TC output. We refer the reader to [23] for the concrete implementation of the Round algorithm.

Parties in [23] then participates in a publicly verifiable secret sharing (PVSS) based distributed key generation (DKG) protocol to receive secret shares of a TC secret s.

Randomness generation. Das et al. [23] implements the weighted extension †† of slow path (Algorithm 1) for blockchain-native threshold cryptosystem (Definition 13), where they use a distributed verifiable unpredictable function (VUF) as the TC protocol. More precisely, for each finalized block, each party computes and reveals its VUF shares. Next, once a party receives verified VUF shares from parties with combined weights greater than or equal to w, it reconstructs the VUF output as its TC output.

7.2 Implementation

Now we describe our implementation of fast path (Algorithm 3) atop [23].

Consensus. Here we explain how the consensus in [23] satisfies the definition of $\mathsf{MBB}_{\mathsf{FT}}$ (Definition 5). Das et al. [23] is built on the Aptos blockchain, which uses an improved version ‡‡ of Jolteon [29] as its consensus protocol. Jolteon tolerates an adversary capable of corrupting parties holding up to t stakes out of a total of $n = 3t + 1$ stakes under partial synchrony. In Jolteon, each message m in $\mathsf{MBB}_{\mathsf{FT}}$ is a block containing transactions, and only one party (called leader) can propose a block for each round.

- *Prefinalization.* A party at local round $\leq r$ calls $\mathsf{prefinalize}(r, m)$ upon receiving a quorum certificate (QC) for m of round r, where a QC consists of votes from parties with combined stakes of 2/3-rd of the total stakes. When a party prefinalizes a block m of round r, it also implicitly prefinalizes all previous rounds that are not yet finalized (either parent blocks or $\perp$). In Jolteon, each party votes for at most one block per round, so that at most one QC can be formed for each round by quorum intersection. This means that for each round, all honest parties can only prefinalize at most one block.
- *Finalization.* A block m is globally finalized for round r if and only if parties with combined $t_{\mathsf{fin}} = 2t + 1$ stakes (or $t_{\mathsf{fin}} - t = t + 1$ honest stakes) calls $\mathsf{prefinalize}(r, m)$. A party locally finalizes m for round r once it receives $(\texttt{PREFIN}, r, m)$ messages from parties with t_{fin} stakes, which also finalizes all previous rounds with parent blocks or $\perp$.

Rounding. Recall that the fast path (Algorithm 3) requires sharing the same secret two sets of thresholds, i.e., $t_{\mathsf{sec}} < t_{\mathsf{rec}}$ for the slow path and $t'_{\mathsf{sec}} < t'_{\mathsf{rec}}$ for the fast path. Consequently, we augment the rounding

**The rounding is only used for the TC protocol. The MBB (consensus) protocol is still based on accurate stakes.

††The weighted extension of Algorithm 1 is where each party has a stake and the threshold check (line 4 of the reconstruction phase) is based on the stake sum instead of the number of parties. However, the implementation can check the combined weight against the weight threshold w, instead of checking the stakes.

‡‡At of June 2024, the consensus latency of Jolteon on the Aptos mainnet is 4 message delays, improved from 5 message delays [29]. Further improvements to 3 message delay are currently in progress [9].

algorithm of [23] to additionally take $(t'_{\mathsf{sec}}, t'_{\mathsf{rec}})$ as input, and output the weight threshold w' for the fast path. That is, $\mathsf{Round}'(\boldsymbol{S}, t_{\mathsf{sec}}, t_{\mathsf{rec}}, t'_{\mathsf{sec}}, t'_{\mathsf{rec}}) \to (\boldsymbol{W}, w, w')$ where w, w' are the weight thresholds of the slow path and fast path, respectively. Round' also provides the same guarantees for the thresholds of fast path, namely any subset of parties with a combined stake $< t'_{\mathsf{sec}}$ will always have a combined weight $< w'$, and any subset of parties with a combined stake $\geq t'_{\mathsf{rec}}$ will always have a combined weight $\geq w'$. As in Algorithm 3, t'_{sec} is set to be equal to the consensus finalization threshold t_{fin}, and t'_{rec} is set to be n (total stake).

Distributed key generation. To setup the secret-shares of the TC secret, we use the DKG protocol of [23] with the following minor modifications. Each party starts by sharing the same secret independently using two weight thresholds w and w'. The rest of the DKG protocol is identical to [23], except parties agree on two different aggregated PVSS transcript instead of one. Note that, these doubles the computation and communication cost of DKG.

Randomness generation. As described in Algorithm 3, parties reveal their VUF shares (TC output shares) for the fast path upon prefinalizing a block, and for the slow path upon finalizing a block. For both paths, the parties collect the VUF shares and are ready to execute the block as soon as the randomness (TC output) is reconstructed from either path. As in [23], parties use the VUF output evaluated on the secret s and the round (and epoch) number of the finalized block as the TC output [§§]. Since the parties are weighted, similar to Section 7.1, we implement the weighted extension of Algorithm 3.

Implementation. We implement fast-path in Rust, atop the open-source Das et al. [23] implementation [10] on the Aptos blockchain. Our implementation [11] is also deployed in Aptos blockchain. For communication, we use the `Tokio` library [2]. For cryptography, we use the `blstrs` library [1], which implements efficient finite field and elliptic curve arithmetic. Similar to Das et al. [23], our implementation runs the share verification step of different parties in parallel and parallelizes the VUF derivation using multi-threading.

7.3 Evaluation Setup

As of July 2024, the Aptos blockchain is run by 140 validators, distributed 50 cities across 22 countries with the stake distributed described in [4]. The 50-th, 70-th and 90-th percentile (average) of round-trip latency between the blockchain validators is approximately 150ms, 230ms, and 400ms, respectively.

Let n denote the total stake before rounding, which is approximately 8.7×10^8. The secrecy and reconstruction thresholds (in stakes) for the slow path are $t_{\mathsf{sec}} = 0.5n$ and $t_{\mathsf{rec}} = 0.660n$, respectively. The secrecy and reconstruction thresholds (in stakes) for the fast path are $t'_{\mathsf{sec}} = 0.667n$ and $t'_{\mathsf{rec}} = 0.830n$, respectively. The total weight of the mainnet validators after rounding is 244. The weight threshold for the slow path is $w = 143$, and that for the fast path is $w' = 184$.

Most of the Aptos validators use the following recommended hardware specs [5].

- CPU: 32 cores, 2.8GHz or faster, AMD Milan EPYC or Intel Xeon Platinum.
- Memory: 64GB RAM.
- Storage: 2T SSD with at least 60K IOPS and 200MiB/s bandwidth.
- Network bandwidth: 1Gbps.

Evaluation metrics. We measure the *randomness latency* as the duration required to generate randomness for each block, as in Definition 12. It measures the duration from the moment the block is finalized by consensus to the when the randomness for that block becomes available. We report the average randomness latency (measured over a period of 12 hours). We also measure and compare the setup overhead for Das et al. [23] and fast-path, using microbenchmarks on machines of the same hardware specs as the Aptos mainnet. As mentioned in Section 7.2, fast-path requires the DKG to share the same secret in two different thresholds, resulting in approaximately twice the cost compared to Das et al. [23]. We also measure the end-to-end DKG latency of fast-path on the Aptos mainnet.

[§§]In the implementation of Algorithm 3 with Jolteon as the MBB protocol, it is safe to omit m in the evaluation function, since all honest parties always prefinalize the same block for each round as mentioned in section *consensus*.

Randomness latency	
Das et al. [23]	fast-path
85.5	24.7

Table 2: Latencies of Das et al. [23] and our fast-path on Aptos mainnet. The latency unit is millisecond.

Scheme	DKG latency			Transcript size
	deal	verify	aggregate	
Das et al. [23]	190.2	171.8	1.6	80,021 bytes
fast-path	377.4	351.6	3.1	160,041 bytes

Table 3: Setup overhead of Das et al. [23] and our fast-path in microbenchmarks. The latency unit is millisecond.

7.4 Evaluation Results.

We report the latency measurements observed on Aptos mainnet.

Randomness latency. Table 2 summarizes the latency comparison of our fast-path and Das et al. [23]. As observed, fast-path significantly reduces the randomness latency of Das et al., by 71%. As mentioned in Section 6.2, the small latency overhead of fast-path comes from the fact that honest parties may need to wait slightly longer after local finalization to receive additional shares from the fast path, since the reconstruction threshold of the fast path is higher than the finalization threshold.

Setup overhead. We report the computation costs and the transcript sizes of the setup in Table 3. The end-to-end DKG latency for setting up fast-path on Aptos mainnet is 16.8 seconds. This includes 14.6 seconds for transcript dealing, dissemination, and aggregation, and 2.2 seconds for agreeing on the aggregated transcript. As observed in Table 3, the computation overhead of DKG is a relatively small proportion of the end-to-end latency. As each party verifies the transcripts of other parties in parallel, the main bottleneck is the dissemination of the transcripts.

Note that the setup overhead occurs only during the initial setup or when the set of parties changes (which happens every few hours or days), and does not affect the blockchain's end-to-end latency as the setup is performed asynchronously to blockchain transaction processing. In contrast, the randomness latency increases the latency of every transaction. Thus, we believe that the significantly improved randomness latency at the cost of higher setup overhead is a reasonable trade off.

8 Discussions

As we discussed in §1, our results applies to many other threshold cryptosystems that are natively integrated into blockchains. Next, we provide a few specific examples.

Threshold signature for state certification. Modern blockchains [10,47] typically require validators to sign the cryptographic digest of the blockchain state (e.g., the root of a Merkle tree representing the state) after executing transactions in each block to certify the state changes to external clients, and prevent forks caused by nondeterminism due to software bugs. Existing blockchains achieve this using either a (weighted) threshold signature or a multi-signature. However, this introduces an additional message delay in every transaction's latency. For the (weighted) threshold signature or the multi-signature, our solution for blockchain-native threshold cryptosystem with tight thresholds (Algorithm 2) can be used to reduce this additional message delay as follows.

During the blockchain consensus protocol, validators start executing the transactions in a block immediately upon receiving the block, i.e., before the block is finalized. Once the block is prefinalized, each validator sends its partial signature on the blockchain state to all other validators. Algorithm 2 guarantees that any state certified by the aggregated signature will be finalized by the blockchain consensus protocol. We want to note that this approach may lead to wasted computation in executing blocks that do not get finalized by the consensus protocol (called orphan blocks). Nevertheless, in modern high-performance blockchains [10,47], where orphan blocks are rare, we believe that this trade-off is justified.

Threshold decryption for privacy. To counteract MEV and enhance privacy guarantees, numerous recent papers propose to use threshold decryption to hide transaction contents in a block until the block is finalized by the consensus protocol [40,49,13,51,39,33,37]. These proposals work as follows. The blockchain validators start by secret sharing a decryption key using threshold secret sharing. Clients submit encrypted transactions to the blockchain, and the validators collectively decrypt the transactions in a block once the block is finalized. Our solutions (Algorithm 2 and Algorithm 3) can also be used to improve the latency of such proposals that rely on non-interactive threshold decryption. However, there is a subtle issue that needs to be addressed first. Recall that in the definition of blockchain-native threshold cryptosystem, the TC output is evaluated with respect to (r, m) where the message m is finalized by round r in Multi-shot Byzantine Broadcast. Therefore, each transaction must be encrypted for a specific round and finalized for that round in Multi-shot Byzantine Broadcast. We leave addressing this issue as an interesting open research direction.

9 Conclusion

In this paper, we studied the problem of blockchain-native threshold cryptosystem and its associated latency overhead. Unlike prior works that introduces at least one additional message delay atop the underlying consensus protocol, we designed a simple protocol that eliminates this overhead for blockchain-native threshold cryptosystem with tight thresholds for a wide class of blockchain protocols. For blockchain-native threshold cryptosystem with ramp thresholds, we formally proved that the latency overhead incurred by prior works is unavoidable. To address this issue, we proposed a protocol that can significantly reduce this overhead in optimistic scenarios, such as when the network is synchronous and the adversary corrupts parties owning a very small fraction of the total shares. We demonstrated the performance improvement of our protocol by deploying it atop the distributed randomness scheme of the Aptos blockchain.

Acknowledgments

We would like to thank Alin Tomescu, Andrei Tonkikh, and Benny Pinkas for helpful discussions.

References

1. blstrs library, `https://docs.rs/blstrs/latest/blstrs/`
2. tokio library, `https://docs.rs/tokio/latest/tokio/`
3. Aptos explorer (2024), `https://explorer.aptoslabs.com/?network=mainnet`
4. Aptos node distribution (2024), `https://explorer.aptoslabs.com/validators?network=mainnet`
5. Aptos node requirements (2024), `https://preview.aptos.dev/en/network/nodes/validator-node/node-requirements`
6. Abraham, I., Malkhi, D., Nayak, K., Ren, L., Yin, M.: Sync hotstuff: Simple and practical synchronous state machine replication. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 106–118. IEEE (2020)
7. Abraham, I., Nayak, K., Ren, L., Xiang, Z.: Good-case latency of byzantine broadcast: A complete categorization. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. pp. 331–341 (2021)
8. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the thirteenth EuroSys conference. pp. 1–15 (2018)
9. Aptos: Aip-89 consensus latency reduction using order votes. (2024), `https://github.com/aptos-foundation/AIPs/blob/main/aips/aip-89.md`
10. Aptos: Official implementation in rust. (2024), `https://github.com/aptos-labs/aptos-core`
11. Aptos: Optimistic path implementation for aptos on-chain randomness. (2024), `https://github.com/aptos-labs/aptos-core/pull/12395`
12. Arun, B., Li, Z., Suri-Payer, F., Das, S., Spiegelman, A.: Shoal++: High throughput dag bft can be fast! arXiv preprint arXiv:2405.20488 (2024)
13. Asayag, A., Cohen, G., Grayevsky, I., Leshkowitz, M., Rottenstreich, O., Tamari, R., Yakira, D.: A fair consensus protocol for transaction ordering. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP). pp. 55–65. IEEE (2018)

14. Babel, K., Chursin, A., Danezis, G., Kokoris-Kogias, L., Sonnino, A.: Mysticeti: Low-latency dag consensus with fast commit path. arXiv preprint arXiv:2310.14821 (2023)
15. Baudet, M., Danezis, G., Sonnino, A.: Fastpay: High-performance byzantine fault tolerant settlement. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 163–177 (2020)
16. Baudet, M., Sonnino, A., Kelkar, M., Danezis, G.: Zef: low-latency, scalable, private payments. In: Proceedings of the 22nd Workshop on Privacy in the Electronic Society. pp. 1–16 (2023)
17. Blakley, G.R.: Safeguarding cryptographic keys. In: 1979 International Workshop on Managing Requirements Knowledge (MARK). pp. 313–318. IEEE (1979)
18. Blakley, G., Meadows, C.: Security of ramp schemes. In: Proceedings of CRYPTO'84 on Advances in cryptology. pp. 242–268 (1985)
19. Buchman, E.: Tendermint: Byzantine fault tolerance in the age of blockchains. Ph.D. thesis (2016)
20. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the third Symposium on Operating Systems Design and Implementation (OSDI). pp. 173–186. USENIX Association (1999)
21. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems (TOCS) **20**(4), 398–461 (2002)
22. Chan, B.Y., Shi, E.: Streamlet: Textbook streamlined blockchains. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT) (2020)
23. Das, S., Pinkas, B., Tomescu, A., Xiang, Z.: Distributed randomness using weighted vrfs. Cryptology ePrint Archive (2024)
24. Doidge, I., Ramesh, R., Shrestha, N., Tobkin, J.: Moonshot: Optimizing chain-based rotating leader bft via optimistic proposals. arXiv preprint arXiv:2401.01791 (2024)
25. Dutta, P., Guerraoui, R., Vukolic, M.: Best-case complexity of asynchronous byzantine consensus. Tech. rep. (2005)
26. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM (JACM) **35**(2), 288–323 (1988)
27. Ethereum: Whitepaper. (2024), `https://ethereum.org/en/whitepaper/`
28. Flow: Jolteon: Advancing flow's consensus algorithm. (2023), `https://flow.com/engineering-blogs/jolteon-advancing-flows-consensus-algorithm`
29. Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.: Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In: International conference on financial cryptography and data security. pp. 296–315. Springer (2022)
30. Gueta, G.G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M., Seredinschi, D.A., Tamir, O., Tomescu, A.: Sbft: a scalable and decentralized trust infrastructure. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 568–580. IEEE (2019)
31. Hanke, T., Movahedi, M., Williams, D.: Dfinity technology overview series, consensus system. arXiv preprint arXiv:1805.04548 (2018)
32. Jalalzai, M.M., Niu, J., Feng, C., Gai, F.: Fast-hotstuff: A fast and resilient hotstuff protocol. arXiv preprint arXiv:2010.11454 (2020)
33. Kavousi, A., Le, D.V., Jovanovic, P., Danezis, G.: Blindperm: Efficient mev mitigation with an encrypted mempool and permutation. Cryptology ePrint Archive (2023)
34. Keidar, I., Naor, O., Poupko, O., Shapiro, E.: Cordial miners: Fast and efficient consensus for every eventuality. arXiv preprint arXiv:2205.09174 (2022)
35. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative byzantine fault tolerance. In: Proceedings of twenty-first ACM Symposium on Operating Systems Principles (SOSP). pp. 45–58 (2007)
36. Kuznetsov, P., Tonkikh, A., Zhang, Y.X.: Revisiting optimal resilience of fast byzantine consensus. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. pp. 343–353 (2021)
37. Malkhi, D., Szalachowski, P.: Maximal extractable value (mev) protection on a dag. arXiv preprint arXiv:2208.00940 (2022)
38. Martin, J.P., Alvisi, L.: Fast byzantine consensus. IEEE Transactions on Dependable and Secure Computing **3**(3), 202–215 (2006)
39. Momeni, P., Gorbunov, S., Zhang, B.: Fairblock: Preventing blockchain front-running with minimal overheads. In: International Conference on Security and Privacy in Communication Systems. pp. 250–271. Springer (2022)
40. Reiter, M.K., Birman, K.P.: How to securely replicate services. ACM Transactions on Programming Languages and Systems (TOPLAS) **16**(3), 986–1009 (1994)
41. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
42. Shrestha, N., Abraham, I., Ren, L., Nayak, K.: On the optimality of optimistic responsiveness. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 839–857 (2020)

43. Solana: Whitepaper. (2024), https://solana.com/solana-whitepaper.pdf
44. Song, Y.J., van Renesse, R.: Bosco: One-step byzantine asynchronous consensus. In: International Symposium on Distributed Computing (DISC). pp. 438–450. Springer (2008)
45. Spiegelman, A., Aurn, B., Gelashvili, R., Li, Z.: Shoal: Improving dag-bft latency and robustness. In: International Conference on Financial Cryptography and Data Security. Springer (2023)
46. Spiegelman, A., Giridharan, N., Sonnino, A., Kokoris-Kogias, L.: Bullshark: Dag bft protocols made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2705–2718 (2022)
47. Sui: Official implementation in rust. (2024), https://github.com/MystenLabs/sui
48. Tonkikh, A., Freitas, L.: Swiper: a new paradigm for efficient weighted distributed protocols. In: Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing. pp. 283–294 (2024)
49. Yang, S., Zhang, F., Huang, K., Chen, X., Yang, Y., Zhu, F.: Sok: Mev countermeasures: Theory and practice. arXiv preprint arXiv:2212.05111 (2022)
50. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 347–356. ACM (2019)
51. Zhang, H., Merino, L.H., Qu, Z., Bastankhah, M., Estrada-Galiñanes, V., Ford, B.: F3b: A low-overhead blockchain architecture with per-transaction front-running protection. In: 5th Conference on Advances in Financial Technologies (2023)