TrustBoost: **Boosting Trust among Interoperable Blockchains**

Peiyao Sheng*
psheng2@illinois.edu
University of Illinois
Urbana-Champaign
USA

Xuechao Wang*
xuechaowang@hkust-gz.edu.cn
The Hong Kong University of Science
and Technology (Guangzhou)
China

Sreeram Kannan
ksreeram@ece.uw.edu
University of Washington
USA

Kartik Nayak
kartik@cs.duke.edu
Duke University
USA

Pramod Viswanath
pramodv@princeton.edu
Princeton University
USA

ABSTRACT

Currently there exist many blockchains with weak trust guarantees, limiting applications and participation. Existing solutions to boost the trust using a stronger blockchain, e.g., via checkpointing, requires the weaker blockchain to give up sovereignty. In this paper, we propose a family of protocols in which multiple blockchains interact to create a *combined* ledger with *boosted* trust. We show that even if several of the interacting blockchains cease to provide security guarantees, the combined ledger continues to be secure – our TrustBoost protocols achieve the optimal threshold of tolerating the insecure blockchains. This optimality, along with the necessity of blockchain interactions, is formally shown within the classic shared memory model, tackling the long standing open challenge of solving consensus in the presence of both Byzantine objects and processes. Furthermore, our proposed construction of TrustBoost simply operates via smart contracts and require no change to the underlying consensus protocols of the participating blockchains, a form of "consensus on top of consensus". The protocols are lightweight and can be used on specific (e.g., high value) transactions; we demonstrate the practicality by implementing and deploying TrustBoost as cross-chain smart contracts in the Cosmos ecosystem using approximately 3,000 lines of Rust code, made available as open source [52]. Our evaluation shows that using 10 Cosmos chains in a local testnet, TrustBoost has a gas cost of roughly $2 with a latency of 2 minutes per request, which is in line with the cost on a high security chain such as Bitcoin or Ethereum.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**.

KEYWORDS

cross-chain interoperability, smart contracts, consensus

ACM Reference Format:
Peiyao Sheng, Xuechao Wang, Sreeram Kannan, Kartik Nayak, and Pramod Viswanath. 2023. TrustBoost: Boosting Trust among Interoperable Blockchains. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3576915.3623080

1 INTRODUCTION

Motivation. Currently there exist more than a thousand (layer 1) blockchains, each with its own trust/security level. Blockchains with weak trust guarantees tend to support limited applications. A common solution for new/weak blockchains is to "borrow" trust from a secure chain. A standard way of lending such trust is via checkpointing [31, 45, 46, 50] – here checkpoints attest to the hash of well-embedded blocks every so often and newly mined blocks follow the checkpoints. For instance, Bitcoin itself was secured by checkpointing by Nakamoto themselves until as late as 2014. A critical point to note is that this form of trust lending involves the very consensus layer of the weak blockchain – the fork choice rule of the weak chains needs to obey the checkpoints. The asymmetrical nature of this approach constrains its applicability, leading to a one-way transfer of trust from stronger to weaker blockchains. Consequently, participants in the weaker chains are often sidelined, losing their ability to influence consensus decisions entirely. In practice, in the Cosmos ecosystem newer and application-specific chains (called "Cosmos Zones") can use the same validator set as the original Cosmos chain (called "Cosmos Hub") via a governance proposal [48] – in return for the trust of the Hub, the Zones give up their individual sovereignty.

Our goal. This state of affairs begets the following question: how should multiple blockchains interact to create a *combined* ledger whose trust is "boosted"? Ideally, the "trust boost" operations (i.e., deciding which specific transactions or applications need to be in the combined ledger and thus enjoy boosted trust-levels) should be simply offered via smart contract operations without altering the consensus layer (i.e., constituent blockchains do not give up their individual sovereignty while collaboratively contributing to the enhanced consensus). Technically speaking, this means answering the following open question: given m multiple blockchain ledgers, f of which are faulty, i.e., without security guarantees, can we combine them in such a way that there is consensus on the combined ledger?

*The first two authors contributed equally to this work. For correspondence on the paper, please contact Xuechao Wang at xuechaowang@hkust-gz.edu.cn.

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623080

Note that the adversary can collude across the f faulty blockchains. For simplicity's sake, we initially assume all ledgers share the same security level and will discuss the protocol's generality later on. Answering this question comprehensively, from impossibility results on trust boosting to a concrete protocol with optimal trust boosting properties to a full-stack implementation in the Cosmos ecosystem are the goals of this paper.

Blockchain bridges. There are two distinct approaches to boosting trust depending on whether the interaction between the blockchain ledgers is passive or active. In the passive mode, there is no communication between the ledgers and a single combiner has read-access to the ledgers and works to form a combined ledger. In the active mode, cross-chain communication (CCC) is allowed across the ledgers via bridges. This approach has only been made possible recently as blockchains have become more interoperable – recent CCC projects include IBC by Cosmos [15], XCM by Polkadot [44], and CCIP by Chainlink [9]. These bridges allow information to be imported across smart contracts residing on the different programmable blockchains – the trust combiner we are envisioning is a smart contract too, residing on *each* of the blockchains.

Main contributions. We examine the multi-chain framework within the classic shared memory model in distributed computing [30], where blockchain clients act as processes and blockchain ledgers serve as shared objects. Moreover, in the active mode, we extend the model to enable communication between objects, capturing the functionality of CCC. This extension aligns seamlessly with today's multi-chain frameworks, as programmable and interoperable blockchains currently form a network of interconnected global computers. Our study of blockchain interactions brings to sharp focus the nuanced models in terms of the interactions needed to achieve combined trust. As the primary theoretical result, we present a formal proof establishing the necessity of CCC for achieving consensus on top of blockchains. To the best of our knowledge, our work is the first to tackle the long standing open problem of solving consensus in the presence of both Byzantine objects and an infinite number of Byzantine processes in the shared memory model. Our key insight involves utilizing "powerful" shared objects (i.e., programmable and interoperable blockchains), to address this complex problem.

Specifically, in the passive mode, we show that consensus on combined ledgers, for any possible combination, is impossible if $f > 0$. Indeed, one of the earlier efforts in the literature [21], tried to create a combined ledger passively without success. Focusing on a weaker form of consensus that gives up the total ordering property (while still being able to implement the functionality of a cryptocurrency), referred to as Asynchronous Blockchain without Consensus (ABC) [47], we show the following in the passive mode. First, even ABC is impossible, if $m \leq 3f$. Second, we propose a protocol called TrustBoost-Lite that combines different ledgers to achieve ABC whenever $m > 3f$. In the active mode, we show that consensus is impossible in a partial synchronous network if $m \leq 3f$. When $m > 3f$, we propose a protocol called TrustBoost that securely combines the m ledgers together. Both TrustBoost and TrustBoost-Lite protocols can be viewed as BFT consensus protocols: consensus is now amongst the programmable blockchains (whose actions are executed by smart contracts) communicating over pairwise authenticated channels provided via the CCC infrastructure – a form of "consensus on top of consensus".

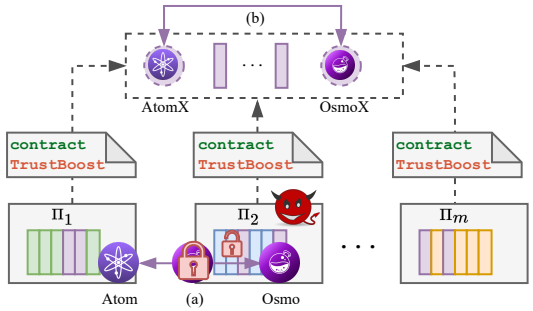

Figure 1: (a) Token exchange across chains are vulnerable to single-chain attacks. Suppose attackers lock 100 Osmos on the Osmosis **chain in exchange for 10 Atoms on the** Cosmos **chain. Once Atoms are received, a double-spend attack on the transaction which locks 100 Osmos on the** Osmosis **chain leads to 10 "free" Atoms, creating a security attack on the** Cosmos **chain. (b)** TrustBoost **secures contract states. Any application contract (e.g., Atom token contract) can be upgraded to a** TrustBoost **cross-chain contract (e.g., AtomX) by creating secure global states. The exchange of** TrustBoost **cross-chain tokens are now secured by the interacting blockchains.**

TrustBoost is a lightweight consensus protocol, executable entirely as a smart contract on each of the blockchains. Further, any specific transactions of any application contract can be upgraded using TrustBoost to avoid single-chain security attacks (an example is depicted in Fig. 1). We demonstrate the practicality by implementing and deploying TrustBoost as cross-chain smart contracts in Cosmos ecosystem using approximately 3,000 lines of Rust code, made available as open source [52].

Several limitations of smart contract programming impose challenges to implementing BFT consensus protocols using them: (a) contracts only behave passively and we need to ensure that every operation in TrustBoost is properly triggered by some IBC message; (b) contracts work only with single-threading, preventing parallelism in operations; (c) Cosmos-SDK allows a smart contract to send IBC messages only when a function returns – a major implementation hurdle, which we deal with by queuing all the IBC messages for each function that need to be sent and send them all when the function returns; (d) finally, special attention should be paid to self-delivered messages. The design principles of our successful implementation of BFT consensus protocols via smart contracts might be of independent and broader interest.

The performance of TrustBoost, particularly latency and gas usage, depends on both the implemented BFT consensus protocols and IBC efficiency. We implement Information Theoretic HotStuff [2] to avoid expensive operations on signature verification, which however leads to an $O(m^2)$ boost in gas usage and a linear increase in latency. Meanwhile, in Cosmos a single IBC message (e.g., for cross

chain token transfer) would take 2 seconds and cost 350K gas. Concretely, with 10 Cosmos chains in a local testnet, the total gas cost is roughly $2 with a latency of 2 minutes when using TrustBoost to boost the security of a standard contract NameService[33] – here gas fees in fiat are extracted from the exchange rate and the gas price of Osmosis, a popular Cosmos Zone at the time of writing (April 2023) and are in line with the gas fees of a high security chain such as Bitcoin or Ethereum. Improving the efficiency of the implemented BFT protocols and IBC would make TrustBoost more performant.

2 RELATED WORKS

In this section, we survey related works encompassing blockchain protocols that borrow or boost trust, hierarchical consensus frameworks, distributed computing models, and blockchain interoperability.

Borrowing trust. Checkpointing is a method that allows the trust of a highly secure blockchain to be extended to weaker or newer blockchains [31, 45, 46]. Validators of a weaker chain periodically submit block hashes and signatures as checkpoints to a more secure chain, and the finality rule of the weaker chain is modified to respect the checkpoints. Consequently, the weaker chain has a slightly slower finality rule - confirming the chain up to the latest checkpoint, which has the same latency and security level as the secure chain. A concrete and practical instantiation of this idea in the context of bringing Bitcoin trust to Cosmos Zones is [50]. A very recent work [49] generalizes checkpointing approaches and offers new insights that align with the design principles underlying our work. The proposed protocol lets a consumer chain draw additional security from a series of provider chains through sequential checkpointing operations. However, this approach necessitates that all chains remain live to guarantee eventual liveness. In contrast, our setting enhances not only the security but also the liveness of all participating chains.

Boosting trust. An early work on robust ledger combining is [21]; parallel ledgers process a common set of transactions independently, and confirmation in the combined ledger is carried out by observers who can read from all ledgers. Similar to TrustBoost, the combine ledger functions even if a certain fraction of underlying ledgers no longer provide any security guarantees. However, the combined ledger only ensures a notion of relative persistence, which is not sufficient even for a payment system, so its practical use is limited. A detailed exploration of this limitation is discussed in Appendix A.

Hierarchical consensus. To the best of our knowledge, Steward [3] was the pioneering work that proposed a concept of "consensus on top of consensus". Steward employs a BFT protocol within each local site and a benign fault-tolerant protocol among wide area sites. Each local site, consisting of several potentially malicious replicas, is converted into a single logical trusted participant in the global protocol. GeoBFT [27] further improves scalability by introducing parallelization of consensus at the local level, and by minimizing communication between sites. However, in comparison to TrustBoost, both Steward and GeoBFT assume a honest supermajority in each local site, which significantly simplified the problem. Furthermore, alterations in the local consensus are needed, whereas in TrustBoost, the global consensus is lightweight and implemented solely through smart contracts.

The idea most closely related to ours is the "recursive Tendermint" [16] proposed by the Cosmos team, in which Tendermint is run on multiple Cosmos chains using the IBC protocol instead of TCP/IP in a peer-to-peer network. However, this concept was only presented as a preliminary idea, without delving into the scientific and engineering challenges that we addressed in TrustBoost.

Shared memory vs. message passing. The shared memory model and message passing model are two fundamental approaches in distributed computing [30]. In the shared memory model, processes communicate by reading and writing to shared objects, whereas in the message passing model, processes exchange messages with one another to coordinate their actions. Consensus, a critical problem in distributed systems, has also been extensively investigated in both the shared memory model [1, 11, 22, 28] and message passing model [8, 18–20, 36, 37, 37], yielding a variety of positive and negative results. In this work, we expand the shared memory model to enable communication between objects, an adaptation that aligns seamlessly with today's multi-chain framework. Furthermore, we establish theoretical bounds in this refined model and present a clear delineation of the interactions required to achieve combined trust among blockchains.

Blockchain interoperability. [57] presents a general framework to design and evaluate CCC protocols that facilitate blockchain interoperability. The most significant application of blockchain interoperability is atomic cross-chain swaps [29, 51], which enable the exchange of assets across multiple distinct blockchains. However, these protocols necessitate intricate and time-consuming user interactions with the blockchains and their peer-to-peer transaction nature often results in lower liquidity compared to centralized exchanges.

In order to facilitate general cross-chain applications, cross-chain bridges have emerged as a significant building block in today's multi-chain world. There are three primary categories: 1) committee-based bridges; 2) optimistic bridges; 3) light client bridges. Committee-based bridges (PolyNetwork [5], Wormhole [53], LayerZero [38], CCIP [9], etc.) employ a trusted committee of validators to sign off on state transfers, with security relying on the honest majority assumption. Optimistic bridges (like Nomad [43] and Near's Rainbow Bridge [42]) require participants to deposit collateral, and depend on a watchdog service to continuously monitor the blockchain and confiscate offenders' collateral upon detecting invalid updates. However, optimistic protocols fundamentally demand long confirmation latency to ensure high probability of detecting invalid updates. Light client bridges (e.g., Cosmos IBC [15]) are trustless, using on-chain light clients to verify state transitions on the other blockchain. Zk-SNARKs are further leveraged to enhance the efficiency of state verification [34, 54]. While TrustBoost can utilize all these bridge types, light-client bridges are preferred due to their trustlessness and efficiency. Further improvements in the security and performance of bridges represent an interesting and active research area, but it falls outside the scope of this paper.

A very recent work [55] proposes a cross-chain state machine replication protocol in the passive mode, which maintains a consistent state across multiple chains; indeed the security guarantees in

[55] hold only when *each* of the involved blockchains is secure (as expected by one of our theoretical results (cf. Theorem 4.1)).

3 PRELIMINARIES

3.1 System model

Our model is inspired by today's public blockchains, which serve as global state machines shared by blockchain clients. Following the notations of shared memory models in the distributed computing literature [30], we consider a shared memory system consisting of a (possibly infinite) collection of *processes* $P_1, P_2, \cdots$ interacting with a finite collection of m *objects* $O_1, O_2, \cdots, O_m$. Both objects and processes are modeled as I/O automata [40].

A process automaton models a blockchain client such as a full node or light client. Since blockchain clients are untrusted, we assume no trust from the processes. In other words, we allow any number of Byzantine processes, but only honest processes enjoy the guarantees of the protocol. An object automaton models a blockchain ledger (or a smart contract), providing three basic interfaces *submit*, *check* and *read*. A process can access the ledger states via the *read* operation and submit transactions to change the states via the *submit* operation. Once a submitted transaction tx is committed, the *check* interface returns true for tx. We assume that at most f out of m objects are Byzantine, potentially providing arbitrary responses. Byzantine objects represent faulty ledgers run by corrupted nodes, whereas ledgers operated by nodes with an honest (super)majority are modeled as correct or honest objects.

Depending on whether the blockchains in the system are interoperable or not, we consider two different mode: an *active mode* and a *passive mode*. In the active modes, objects are fully connected via authenticated channels and exchange messages through these channels. The channels provide two interfaces, *send* and *deliver*, and they ensure message integrity and reliability, that a message is delivered from p to q if and only if it was previously sent from p to q. We assume this fully connected object network is partially synchronous, meaning there is a global stabilization time (GST) chosen by the adversary, unknown to honest nodes and also to the protocol designer, such that after GST, all messages sent between honest objects are delivered within Δ time. Before GST, the adversary can delay messages arbitrarily. When $GST = 0$, the network becomes synchronous. In practice, authenticated channels are instantiated by cross-chain bridges. In the passive mode, objects do not communicate with each other, and processes have read-only access to objects. Note that granting write access to the processes would effectively transform the passive mode into the active mode, as processes can facilitate message forwarding among objects.

Additionally, to model blockchains capable of producing commitment certificates as proofs of confirmation (for example, the quorum certificate in many BFT-SMR protocols), we further assume the existence of a public-key infrastructure (PKI) for the set of objects. It is important to note that the PKI may or may not be required in our construction.

Contrast with classic shared memory model. In the traditional shared memory model, communication between processes is facilitated through a collection of shared objects, such as registers. These objects store values and offer two basic operations: *read* and *write*. The read operation retrieves the requested values, while the write operation updates the stored values. In our model, an object is extended to store states in a blockchain ledger instead of a single value. Hence, we adapt the interfaces to allow processes to *submit* transactions (analogous to write values) and *read* states (analogous to read values). The additional interface *check* returns the transaction status, indicating whether a *submit* operation has been successfully completed. Our approach aligns with the regular register model, where a read operation can return the value written by either the most recently concluded write operation or a write operation with which it overlaps [35]. Our passive mode represents a simplified version of the classic shared memory model, with processes having read-only access to objects. This mirrors the "lazy" trust-boosting approach, where each blockchain client has read-access to the ledgers and works to form a combined ledger. On the other hand, our active mode is more nuanced and presents a clear distinction from the classic model. In the traditional shared memory model, objects are passive, meaning they undergo state changes only in response to process requests. However, in our active mode, objects can communicate with other objects or observe their states. This advanced shared object capability has been made possible by recent developments in blockchain interoperability. It also improves the system's modularity, as the message-passing operations between objects encapsulate a pair of read and write operations between different objects and processes. Additionally, our active mode can also be considered as a new hybrid model that combines shared memory across processes and message passing across objects, whereas in classic distributed computing, only messages among processes are considered.

3.2 Consensus problems

To study the security guarantee of the "meta" consensus protocol built on top of blockchains, we provide definitions for several relevant problems, ranging from binary consensus to ledger consensus, including their relaxed versions.

Binary consensus. We start with the problem of *binary consensus*, where each object starts with some initial value (0 or 1) and all (honest) processes try to commit the same value by the end of the protocol.

Definition 1 (Binary Consensus).

- ***Agreement****: No conflicting values are committed by honest processes.*
- ***Validity****: If every honest object starts with the same value, this value will be committed by honest processes.*
- ***Termination****: Every honest process commits one of the values.*

We are also interested in a relaxed notion of binary consensus, called *ABC binary consensus*, derived from [47]. The key insight leading to this relaxation is that in numerous applications, termination is only required in optimistic scenarios.

Definition 2 (ABC Binary Consensus).

- ***Agreement****: Same as in Definition 1.*
- ***Validity****: Same as in Definition 1.*
- ***Honest termination****: If every honest object starts with the same value, this value will be committed by honest processes.*

Ledger consensus. It is known that binary consensus protocols can be used to solve the problem of *ledger consensus*, i.e., *state machine replication (SMR)* [41], where all processes maintain a list of transactions that grows in length, called a *public ledger* (cf. Definition 3), with the help of the shared objects. Each process initially starts with the same state and updates the state by executing all transactions in its ledger. We first define the notion of a public ledger.

Definition 3 (Public ledger). *A public ledger $\mathcal{L}$ is a growing list of transactions that provides the following two process interfaces:*

- *Submit: a process can submit a transaction tx to $\mathcal{L}$ by calling* $\mathsf{submit}(tx)$.
- *Check: a process can check whether $tx \in \mathcal{L}$ by calling* $\mathsf{check}(tx)$. *If* $\mathsf{check}(tx)$ *returns true, the process will commit tx.*
- *Read: a process can access the set of states S stored on $\mathcal{L}$ by calling* $\mathsf{read}(S)$.

In the problem of ledger consensus, we allow the following synchronous out-of-band communications among processes: 1) processes can send transactions to objects, but still processes send no other message to objects; 2) processes can communicate with each other only to prove the confirmation of certain transactions (i.e., allowed to sync up state with each other or bootstrap new processes). The guarantees of ledger consensus are defined as follows.

Definition 4 (Ledger Consensus).

- ***Agreement****: If some honest process commits tx, every honest process will also commit tx; Moreover, tx appears at the same place in the ledgers of all honest processes. Equivalently, if $[tx_0, tx_1, \cdots, tx_i]$ and $[tx'_0, tx'_1, \cdots, tx'_{i'}]$ are two ledgers output by two honest processes, then $tx_j = tx'_j$ for all $j \leq \min(i, i')$.*
- ***Termination****: If a process submits tx to all honest objects, tx will be committed by all honest processes.*

Similarly, ABC can also be used to build a public ledger, although without total ordering of the transactions. However, this suffices to implement the functionality of a cryptocurrency like Bitcoin as shown in [4, 25, 47]. Suppose the transaction space is now equipped with a conflict relation (e.g. double spend transactions in Bitcoin), then the problem of ABC ledger consensus is defined as follows.

Definition 5 (ABC Ledger Consensus).

- ***Weak agreement****: No conflicting transactions are committed by honest processes.*
- ***Honest termination****: If a process submits tx to all honest objects and there are no conflicting transactions, tx will be committed by all honest processes.*

Contrast with consensus in traditional models. As mentioned earlier, our passive mode is a simplified version of the classic shared memory model. Surprisingly, as we will demonstrate in §4, ABC remains achievable even within this highly restricted model. This observation leads to the development of the protocol TrustBoost-Lite in §5.2, where no CCC is needed. As for our active mode, it is better to compare it with the classic message passing model, where m validators communicate to reach consensus. It is not difficult to see that any consensus protocol Π in the message passing model can be used to solve consensus in the active mode: Let $3f+1$ objects implement Π; each process commits a value/transaction if it is read from at least $2f+1$ objects (note that solving consensus requires $m > 3f$ in partial synchrony). This reduction allows us to construct TrustBoost in a black-box manner. Conversely, solving consensus in our active mode does not necessarily solve consensus in the classic message passing model. Indeed, in the above example, there are $m-3f-1$ objects unused when solving consensus in our active mode, but all m validators need to participate and reach consensus in the message passing model.

In summary, our active mode offers a more nuanced and flexible approach, combining aspects of shared memory and message passing models, and leveraging advancements in blockchain interoperability. While it can utilize traditional consensus protocols, it can also achieve consensus with fewer objects and additional efficiency, making it a distinct and innovative approach compared to traditional models.

4 THEORETICAL BOUNDS FOR THE SHARED MEMORY MODEL

In a seminal work [39], an impossibility result for the binary consensus problem is shown for an asynchronous shared memory system, where a single process may crash. In this work, we circumvent this impossibility result by (1) adopting a weaker notion of consensus in our passive mode, and (2) permitting partial synchronous communication among objects in the active mode.

Previous works [1, 11, 36] circumvent this impossibility result by employing a leader oracle. The state-of-the-art study [1] proves that 1/3 is the optimal resilience of Byzantine objects for solving consensus; however, it only takes benign processes into account. A recent work [12] considers Byzantine processes but only focus on problems weaker than consensus, such as reliable broadcast, snapshots, and asset transfer (essentially equivalent to ABC). To the best of our knowledge, our work stands as the first attempt to tackle the long standing open problem of solving consensus in the presence of both Byzantine objects and an infinite number of Byzantine processes in the shared memory model.

Specifically, in this section, we show that in the passive mode (i.e., without communication among objects), binary consensus (cf. Definition 1) is impossible, while ABC binary consensus (cf. Definition 2) can be still achieved. Meanwhile, in the active mode, both problems can be solved as long as $m > 3f$. We adapt proof techniques from classic distributed system problems [18, 20] to our nuanced model and show the following tight results and sharp distinctions between consensus and ABC.

4.1 Passive mode

We start with the theoretical bounds in the passive mode.

Theorem 4.1. *In the passive mode, we have*

(1) Consensus can be achieved if and only if $f = 0$.
(2) ABC can be achieved if and only if $m > 3f$.

Proof. Since in our model, the consensus protocols may or may not use PKI, we prove the strongest results: for the negative results (i.e., "only if" parts), we assume PKI is used; and for the positive results (i.e., "if" parts), we avoid using PKI.

(1) Consensus:
Only if: Seeking a contradiction, let us assume there is a protocol that solves consensus in the passive mode. Divide all the processes into two sets: X and Y, each with at least one honest process. We first construct the following $m+1$ worlds. See Fig. 2.
World $1.i$ **(**$0 \le i \le m$**):** In world $1.i$, the first i objects start with value 0 and the rest $m-i$ objects start with value 1. By termination, in all $m+1$ worlds, processes in X and Y must eventually commit some value. By validity, the committed value must be 1 in world 1.0 and 0 in world $1.m$. Then there must exist some integer $0 \le j \le m-1$ such that the committed value is 1 in world $1.j$ and 0 in world $1.(j+1)$. Now consider the following world:
World 2: World 2 will be a hybrid world where the view of processes in X in this world will be indistinguishable to their views in world $1.(j+1)$ and the view of processes in Y in this world will be indistinguishable to their view in world $1.j$. In world 2, the first j objects start with value 0 and the last $m-j-1$ objects start with value 1. The adversary will use its Byzantine power to corrupt the $(j+1)$-th object to perform a split-brain attack and make X and Y each believe that they are in their respective worlds. The $(j+1)$-th object will equivocate and act as if its starting value is 0 when communicating with X and as if its 1 when communicating with Y. And since there is no communication among the objects, their views will be indistinguishable to the views in worlds $1.j$ and $1.(j+1)$. Moreover, the view of X in this world will be indistinguishable to the view of X in world $1.(j+1)$ and the view of Y in this world will be indistinguishable to the view of Y to world $1.j$. Therefore, X will commit 0 and Y will commit 1. This violates the agreement property.

If: With $f=0$, we solve the problem of consensus in the passive mode. The protocol is simple: each process queries the value from all m objects; and the process commits a value if it receives the same value from all m objects; otherwise if the process receives both 0 and 1, it commits 0 by default. It is easy to check that this protocol satisfies agreement, validity and termination; Moreover, there is no communication among the objects.

(2) ABC:
Only if: Seeking a contradiction, let us assume there is a protocol that claims to solve ABC with $f \ge m/3$ Byzantine objects. Divide the m objects into three sets: A, B, and C, each with at least one object and at most f objects. Divide all the processes into two sets: X and Y, each with at least one process. We consider the following three worlds and explain the worlds from the view of A, B, C, X and Y.
World 1: In World 1, objects in A and B start with the value 1. Objects in C are Byzantine but pretend to be honest with initial value 0. Since C has at most f objects, the processes in X must eventually commit a value by honest termination. For validity to hold, all the processes in X will output 1.
World 2: In World 2, objects in B and C start with the value 0. Objects in A are Byzantine but pretend to be honest with initial value 1. Since A has at most f objects, the processes in Y must eventually commit a value by honest termination. For validity to hold, all the processes in Y will output 1.
World 3: World 3 will be a hybrid world where the view of X in this world will be indistinguishable to the view of X in world 1 and the view of Y in this world will be indistinguishable to the view of Y in world 2. A will start with value 1 and C will start with value 0. The adversary will use its Byzantine power to corrupt B to perform a split-brain attack and make X and Y each believe that they are in their respective worlds. B will equivocate but act honestly as if its starting value is 1 when communicating with X and as if its 0 when communicating with Y. Then by an indistinguishability argument, X will commit 1 and Y will commit 0. This violates the agreement property.

If: Assume $m > 3f$, now we solve the problem of ABC in the passive mode. The protocol is simple: each process queries the value from all m objects; and the process commits a value if it receives the same value from at least $2f+1$ objects. It is easy to check that this protocol satisfies agreement, validity and honest termination; Moreover, there is no communication among the objects. □

4.2 Active mode

Active mode. For the completeness of our results, we prove the following theorem in the active mode, which is similar to the well-known impossibility result in the partially synchronous network model [18].

Theorem 4.2. *In the active mode, we have:*

(1) Consensus can be achieved if and only if $m > 3f$.
(2) ABC can be achieved if and only if $m > 3f$.

Proof. Since in our model, the consensus protocols may or may not use PKI, we prove the strongest results: for the negative results (i.e., "only if" parts), we assume PKI is used; and for the positive results (i.e., "if" parts), we avoid using PKI.
(1) Consensus:
Only if: Seeking a contradiction, let us assume there is a protocol that claims to solve consensus with $f \ge m/3$ Byzantine objects. Divide the m objects into three sets: A, B, and C, each with at least one object and at most f objects. Divide all the processes into two sets: X and Y, each with at least one process. We consider the following three worlds and explain the worlds from the view of A, B, C, X and Y. In all three worlds, we will assume that all messages between $A \longleftrightarrow B$ and $B \longleftrightarrow C$ arrive immediately. See Fig. 3.
World 1: In World 1, objects in A and B start with the value 1. Objects in C and processes in Y have crashed. Since C has at most f objects, the processes in X must eventually commit a value by termination. For validity to hold, all the processes in X will output 1. From the perspective of A, B and X, they cannot distinguish between a crashed (or Byzantine) C vs. an honest C whose messages are delayed.
World 2: World 2 will be a world similar to world 1 where the roles of A and C and the roles of X and Y are interchanged. The objects in B and C start with the value 0. Objects in A and processes in X have crashed. So, all the processes in Y will output 0 by termination and validity. Again, from the perspective of B, C and Y, they cannot distinguish between a crashed A vs. an honest A whose messages are delayed.

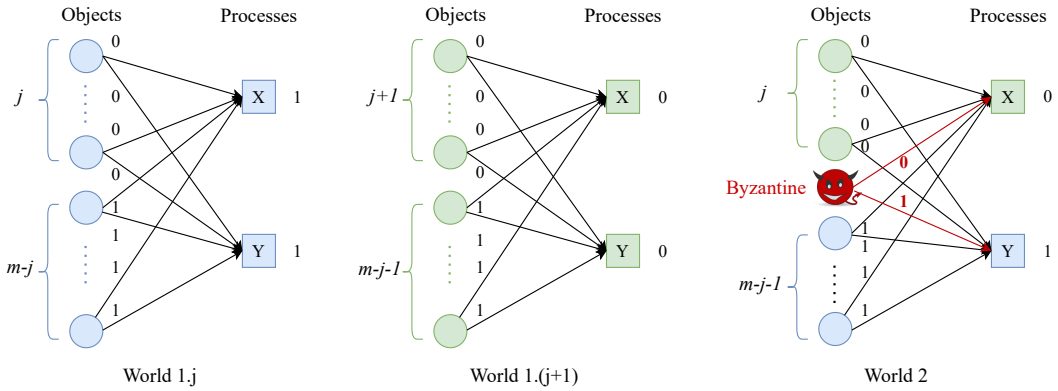

Figure 2: Different worlds in the proof of Theorem 4.1.

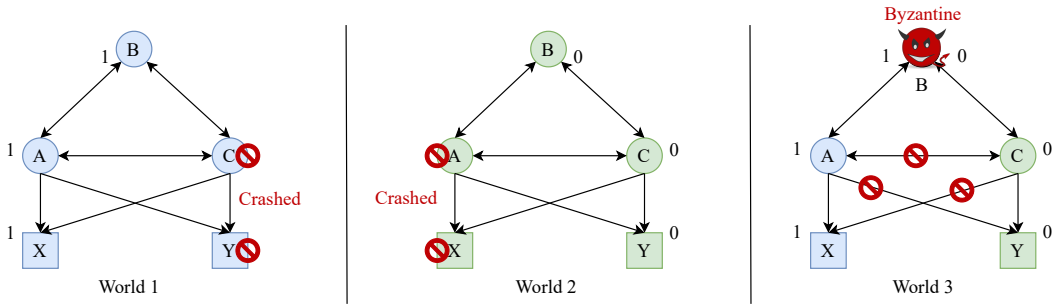

Figure 3: Different worlds in the proof of Theorem 4.2.

World 3: World 3 will be a hybrid world where the view of A and X in this world will be indistinguishable to the view of A and X in world 1 and the view of C and Y in this world will be indistinguishable to the view of C and Y in world 2. A will start with value 1 and C will start with value 0. The adversary will use its Byzantine power to corrupt B to perform a split-brain attack and make A (or X) and C (or Y) each believe that they are in their respective worlds. B will equivocate and act as if its starting value is 1 when communicating with A and X and as if its 0 when communicating with C and Y. If the adversary delays messages between $A \longleftrightarrow C$, $A \longrightarrow Y$ and $C \longrightarrow X$ for longer than the time it takes for X and Y to decide in their respective worlds, then by an indistinguishability argument, X will commit to 1 and Y will commit to 0. This violates the agreement property.

If: Assume $m > 3f$, now we solve the problem of consensus in active mode. The objects run any partial synchronous consensus protocol (in the classic message passing model) and send the committed value to the clients. The client commits a value if it receives the same value from at least $2f + 1$ objects. It is easy to check that this protocol satisfies agreement, validity and termination.

(2) ABC:

Only if: Same as the proof for consensus as above. Note that in world 1&2 honest objects start with the same value, hence honest termination suffices for the proof.

If: Same as the algorithm that solves ABC in the passive mode (in the proof of Theorem 4.1). □

Consensus with reduced communication and connectivity. So far, we have observed that ABC (cf. Definition 2) can be solved without any communication among the objects, while consensus (cf. Definition 1) cannot. One natural question arises: Can we design new consensus protocols to make TrustBoost more efficient in terms of number of IBC connections and messages? Theoretically, it would also be interesting to study the lower bounds on number of messages and network connectivity for consensus in our active mode. For instance, it is evident that when solving consensus with $m > 3f + 1$ objects, we only need $3f + 1$ objects. The extra $m - 3f - 1$ objects do not even to establish connections with other objects. Meanwhile, in the classic message passing model, all honest objects must form a connected component in order to solve consensus. However, identifying the minimum requirement on network connectivity for consensus in the active mode remains an interesting and challenging problem.

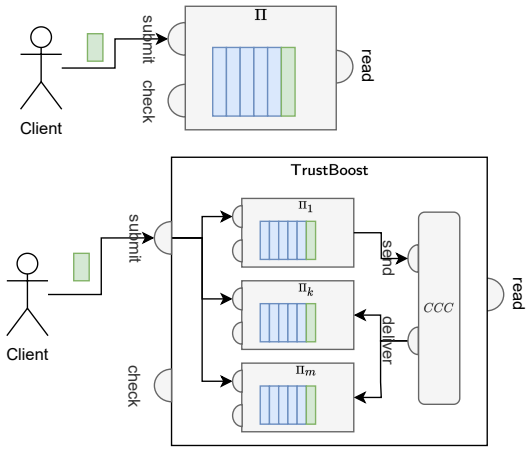

Figure 4: Clients see the same interface of submitting a transaction to TrustBoost **as submitting a transaction to a single blockchain.**

5 THE TRUSTBOOST PROTOCOL

5.1 TrustBoost: blockchains as objects

TrustBoost is run by m blockchains, which support smart contracts, simulating a group of objects written by processes (blockchain clients) to maintain global states. The i-th blockchain ($1 \le i \le m$) is run by a group of nodes with β_i fraction of which are Byzantine. Blockchains can be permissioned or permissionless. Communications between blockchains are made possible by cross-chain communication (CCC), which is a protocol that manages bidirectional ledger-to-ledger links. There are two primitives used by the TrustBoost protocol.

- **Local consensus protocol.** Π_k is a public verifiable ledger consensus protocol (cf. Definition 4), it provides an interface Π_k.submit(tx) (analogous to the write operation of objects), allowing clients to submit and commit transactions to the blockchain. Clients can verify if a transaction tx has been committed to the ledger through Π_k.check(tx) and read the current ledger state S through Π_k.read(S), which is analogous to the read operation of objects. Optionally, Π_k.check(tx) may return a commitment certificate, such as a quorum certificate in many BFT-SMR protocols, as proof of commitment. Π_k can be instantiated by a black-box partial synchronous consensus protocol.
- **Cross-chain communication (CCC).** CCC is a routing protocol that enables independent blockchains to actively communicate with each other. It consists of two primitives, CCC.send(src, dst, tx) and CCC.deliver(src, dst, tx), which transmit transaction tx, with src and dst representing the source and destination blockchain identifiers, respectively. CCC ensures both reliability and authenticity. Reliability guarantees that any transaction sent by the source chain will be delivered to the destination chain eventually, while authenticity ensures that any node on the destination chain can verify that the transactions delivered by the protocol correspond to committed states on the source chain.

Algorithm 1 Protocol TrustBoost (CCC, k, Π_k)

1: **Init:**
2: $txVotes \leftarrow$ an empty map
3: $m \leftarrow$ number of participating chains
4: **function** submit(tx)
5: **for** $dst = 1, \cdots, m$ **do**
6: invoke CCC.send($k, dst, \langle$Propose, $tx\rangle$)
7: **upon event** CCC.deliver($src, dst, \langle$Propose, $tx\rangle$) **do**
8: $txVotes[tx] = \emptyset$
9: **for** $dst = 1, \cdots, m$ **do**
10: invoke CCC.send($k, dst, \langle$Vote, $tx\rangle$)
11: **upon event** CCC.deliver($src, k, \langle$Vote, $tx\rangle$) **do**
12: $txVotes[tx] = txVotes[tx] \cup \{src\}$
13: **if** $|txVotes[tx]| = \lfloor 2m/3 \rfloor + 1$ **then**
14: invoke Π_k.submit(tx)
15: **function** check(tx)
16: $cnt \leftarrow 0$
17: **for** $k = 1, \cdots, m$ **do** ▷ These m queries are executed concurrently
18: **if** Π_k.check(tx) returns true **then**
19: $cnt \leftarrow cnt + 1$
20: **if** $cnt \ge \lfloor m/3 \rfloor + 1$ **then**
21: return true
22: return false
23: **function** read(S)
24: $tx \leftarrow$ the latest transaction which has $check(tx) = true$
25: $C \leftarrow$ the set of chains whose latest committed transaction is tx
26: **for** $k \in C$ **do**
27: $V_k = \Pi_k$.read(S)
28: return V_k if majority of C agree on V_k

TrustBoost **protocol.** With the above primitives, the TrustBoost protocol can be built up by running a public verifiable ledger consensus protocol to issue global states on m local blockchains. It also provides an interface TrustBoost.submit(tx) to accept requests from clients, an interface TrustBoost.check(tx) for clients to check the commitment of transactions (see Fig. 4) and an interface TrustBoost.read(S) to fetch the current ledger states S. By using CCC as transmission channels, and invoking $\Pi_k (1 \le k \le m)$ to commit transactions, any partial synchronous consensus protocol can be used to instantiate TrustBoost. While local consensus protocols can be either permissioned or permissionless, TrustBoost protocol is a permissioned BFT protocol.

For simplicity, we demonstrate how to instantiate TrustBoost using a majority voting protocol in Algorithm 1. Although not a complete consensus protocol, it contains essential building blocks (propose, vote, and commit phases) of most consensus protocols and the usage of all primitives. Initially, the protocol sets up data structures to store votes for transactions. When a client wants to post a transaction tx on TrustBoost blockchains, it calls the TrustBoost.submit function (line 4), which prompts CCC to broadcast a proposal to all blockchains. Lines 7-14 detail how to handle cross-chain transactions such as Propose and Vote. Once the commitment condition is met, for example, in this case, a supermajority of votes

are collected, the TrustBoost protocol commits the transaction by requesting the local blockchain to commit the transaction (line 14). The protocol also offers a check function (line 15) to derive the global states of a transaction from local states. In this example, the transaction is considered committed by TrustBoost once it is committed by more than 2/3 of the local blockchains.

More complex rules can be designed according to the specifications of various consensus protocols. For example, when instantiating TrustBoost with BFT protocols that use signatures, we require Π_k to provide commitment certificates to enable authentication after being forwarded to a third chain. Conversely, local blockchains can also be permissionless (like public proof-of-stake blockchains) if the consensus protocol used for TrustBoost does not rely on a PKI.

Security guarantee. In TrustBoost, each blockchain operates similarly to a single object, with read and write operations used to access and apply changes to stored states. A blockchain that adheres to the security assumption ($\beta_i < \theta_i$, where θ_i is the security threshold of Π_i) is considered an honest blockchain and behaves like an honest object (e.g., no equivocation). Otherwise, it may behave arbitrarily, akin to a Byzantine object, when the security assumption is not met.

Let $f = |\{i : \beta_i \geq \theta_i, 1 \leq i \leq m\}|$ represent the number of faulty blockchains. We aim to prove that TrustBoost securely constructs a combined ledger with total ordering as long as $f < t$, where t is the security threshold of TrustBoost .

To formally demonstrate that TrustBoost solves ledger consensus, we must establish that it satisfies the two properties defined in Definition 4.

First, agreement necessitates that if an honest client determines a transaction tx has been committed on TrustBoost , every honest client will also commit it. Since TrustBoost employs a publicly verifiable ledger consensus protocol, when $f < t$, it ensures agreement as long as the underlying blockchains can simulate an honest object. In the SMR literature, an honest object is consistently defined as an object that abides by the protocol's rules. To better understand the descriptions of an honest object, we can think of it as a server that reads inputs from a communication channel and performs actions (such as updating stored states or broadcasting messages through communication channels) according to the protocol specifications. In this context, an honest blockchain with accessible CCC can also read inputs or broadcast messages by invoking CCC's deliver, send interface and executing actions through smart contracts to apply changes to its states. Moreover, since an honest blockchain runs a publicly verifiable ledger consensus, it guarantees that the blockchain will never reach two conflicting states concerning TrustBoost execution (agreement).

To prove termination, if a client submits a transaction tx to all honest blockchains, the TrustBoost protocol ensures termination, which means it will prompt all honest blockchains to commit tx. Subsequently, each honest blockchain will process the tx execution through a local transaction, which will terminate due to the termination property of the local consensus protocol. As a result, the TrustBoost.check(tx) will eventually return true.

5.2 TrustBoost-Lite

Here we propose a lightweight protocol called TrustBoost-Lite that solves the problem of ABC ledger consensus as defined in Definition 5. In TrustBoost-Lite, m blockchains are independently run by each group C_i using local consensus protocols Π_i ($1 \leq i \leq m$). The transactions in TrustBoost-Lite use the unspent transaction output (UTXO) model, where a transaction consists of a set of inputs and outputs and can be denoted as $tx = (in, out)$. The inputs are pointers to some outputs in previously committed transactions, we use *input.from* to represent the transaction that contains the output which the *input* points to. TrustBoost-Lite only provides two interfaces, submit and check, which are defined in Algorithm 2. The read interface is not available in TrustBoost-Lite because there is no consensus on the ledger states.

TrustBoost-Lite **protocol.** Different from TrustBoost, no cross-chain communication is needed in TrustBoost-Lite. Thus, the requests from clients will trigger the submit of each local blockchain directly (line 5). And to check whether a transaction is committed by TrustBoost-Lite, the client will observe the states of m individual blockchains and make sure (1) the transaction is committed on at least 2/3 of local blockchains and (2) all its inputs are outputs from globally committed transactions (line 11).

Security guarantee. Let $f = |\{i : \beta_i \geq \theta_i, 1 \leq i \leq m\}|$ represent the number of faulty blockchains. We aim to prove that TrustBoost-Lite solves ABC ledger consensus. To demonstrate this, we must show that it satisfies the two properties defined in Definition 5.

To prove weak agreement, we assume that an honest client commits tx, which means the check function of TrustBoost-Lite returns true when the honest client called it. In other words, at least $\lfloor 2m/3 \rfloor + 1$ of the blockchains have committed tx, and there does not exist a committed double-spending input. Among these chains, at least $\lfloor 2m/3 \rfloor + 1 - f$ of the blockchains are honest, and they will commit tx and terminate within a bounded time. Since honest blockchains will never commit a conflicting transaction (agreement), faulty blockchains, together with the remaining honest blockchains, are not enough to commit a conflicting transaction. Hence, if an honest client commits tx, no conflicting transactions can be committed.

Termination can be guaranteed when clients are honest and do not submit conflicting transactions. Honest clients will submit tx to all honest blockchains, and the tx will be eventually included in each of them. Given the termination of the ledger consensus running on local blockchains, the check function will return true eventually.

Algorithm 2 Protocol TrustBoost-Lite (Π_k)

Init:
  $m \leftarrow$ number of participating chains
function submit(tx)
  for $dst = 1, \cdots, m$ **do**
    invoke Π_k.submit(tx)
function check(tx)
  $cnt \leftarrow 0$
  for $k = 1, \cdots, m$ **do** ▷ These m queries are executed concurrently
    if Π_k.check(tx) returns true **then**
      $cnt \leftarrow cnt + 1$
  if $cnt \geq \lfloor 2m/3 \rfloor + 1$ and valid(tx) **then**
    return true
  return false
function valid(tx)
  $(in, out) \leftarrow tx$
  $valid \leftarrow$ true
  for $input \in in$ **do**
    if check($input.from$) returns false **then**
      $valid \leftarrow$ false
  return $valid$

6 IMPLEMENTATION

We implement TrustBoost in the Cosmos ecosystem [32], which is a decentralized network of parallel and interoperable blockchains, each powered by BFT consensus protocols like Tendermint [6], where the CCC is enabled by an inter-blockchain communication (IBC) protocol [24]. In this section, we first give a brief overview of the Cosmos ecosystem and then highlight the key challenges to implement TrustBoost.

6.1 Cosmos **overview**

Cosmos **SDK.** Cosmos-SDK [13] provides tools for building permissioned or proof-of-stake (PoS) blockchains. Cosmos-SDK allows developers to easily create custom programmable and interoperable blockchain applications within the Cosmos network without having to recreate common blockchain functionality. Cosmos-SDK has several pre-built modules to serve different functionalities such as defining transactions, handling application state and the state transition logic, etc. The most important modules related to TrustBoost are the CosmWasm module [14] and the IBC module [15] (see below).

CosmWasm. CosmWasm adds smart contract support to the Cosmos chains, where Rust is currently the most used programming language for contracts. The basic function calls of a CosmWasm contract are executed through an entry point (or a handler) shown in Fig. 5 (Left), by processing two given parameters[1]. The `info` contains contract information about function executions such as the address of the transaction sender, while the `executeMsg` encapsulates the name and parameters of the target function, which will be processed by the handler using a pattern-matching statement.

IBC protocol. IBC is an interoperability protocol for communicating arbitrary data between Cosmos blockchains. The protocol consists of two distinct layers: the transport layer which provides the necessary infrastructure to establish secure connections and authenticate data packets between chains, and the application layer, which defines exactly how these data packets should be packaged and interpreted by the sending and receiving chains. In the transport layer, blockchains are not directly sending messages to each other over networking infrastructure, but rather are creating messages to be sent which are then physically relayed from one blockchain to another by monitoring "relayer processes". These relayers [17] continuously scan the state of chains that implement the IBC protocol and relay packets when these packets are present. This enables transaction execution on connected chains when outgoing packets relayed over have been committed. Relayers cannot modify IBC packets, as each IBC packet is verified using light-clients by the receiving chain before being committed.

Cosmos **ecosystem.** Currently, over twenty CosmWasm-enabled blockchains are connected in the Cosmos ecosystem by the IBC protocol. Therefore, Cosmos provides the ideal environment for us to build and deploy TrustBoost.

6.2 TrustBoost **implementation.**

We implement and deploy TrustBoost as cross-chain smart contracts on $3f + 1$ Cosmos chains. It consists of two major parts, the TrustBoost contract and cross-chain application contracts (denoted as AppX). A complete flow chart is shown in Fig. 6. To use application with TrustBoost, a client first calls the TrustBoost contract to initiate a request to a specific application (①), which is logged on the local blockchain (②) and trigger a consensus protocol among all the blockchains (③). Once the request is committed by TrustBoost, it calls the corresponding application contract to execute the request (④). Clients can extract global states from each local contract (⑤). In our implementation, the example application is the NameService contract, where users can buy and transfer domain names. We also observe that the changes to turn a single-chain application contract into a cross-chain one are minor[2].

Next, we discuss key challenges of implementing TrustBoost.

Use PKI or not? In the relatively recent blockchain era, partially synchronous BFT protocols have received renewed attention; performant and efficient BFT protocols have been constructed (e.g., SBFT [26], Tendermint [6] and HotStuff [56]). However, all these protocols require a public key infrastructure (PKI) or even a threshold signature scheme. Unfortunately, we found that in Cosmos-SDK, once an IBC packet is verified using light-clients by the receiving chain, the signatures (i.e., the quorum certificate from the sending chain) are then removed and only the plaintext of the IBC message is passed to the receiving chain. Therefore, the IBC message verification in Cosmos current lacks of transferability, that is a third chain won't be able to verify that an IBC message is indeed sent from its sending chain.

Due to this limitation, we have to implement BFT protocols without using PKI and the state-of-the-art one is Information Theoretic HotStuff (IT-HS) [2]. IT-HS has a quadratic message complexity in each view; and it is optimistically responsive: with a honest leader, parties decide as quickly as the network allows them to do so, without regard for the known upper bound on network delay.

[1] We omit some semantic details for simplicity, check full descriptions in [14].

[2] See changes to upgrade a contract to have TrustBoost support in github link.

```
fn execute(info, executeMsg) {
    let funcName = executeMsg.funcName;
    let param = executeMsg.param;




    match executeMsg {
        funcA -> funcA(info, param),
        funcB -> funcB(info, param),
        ...
    }
}
// definition of funcA, funcB...
```

```
fn execute(info, executeMsg ) {
    let funcName = executeMsg.funcName;
    let param = executeMsg.param;

    assert_eq!(info.sender, addressOfTB);
    let TBInfo = executeMsg.TBInfo;

    match executeMsg {
        funcA -> funcA( TBInfo , param),
        funcB -> funcB( TBInfo , param),
        ...
    }
}
// definition of funcA, funcB...
```

Figure 5: Left: Handler of CosmWasm contract. Right: Handler of AppX contract with TrustBoost proxy.

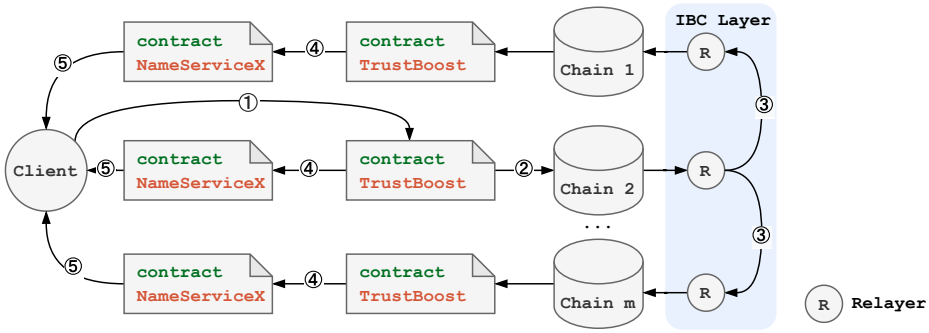

Figure 6: The flow chart of executing a TrustBoost transaction.

Although IT-HS has slightly higher round and message complexity than Tendermint/HotStuff, it avoids many expensive operations, such as signature verification and aggregation. In addition, IT-HS only requires constant persistent storage. Therefore, we believe IT-HS is a good candidate to be implemented as the consensus protocol for our TrustBoost smart contract.

We also note that if we make a few minor changes to the Cosmos-SDK (i.e., passing the signatures to the smart contract layer), then TrustBoost can implement any BFT protocol. However, to make sure that TrustBoost is directly deployable in the real world, we keep the Cosmos-SDK untouched and stick to IT-Hotstuff in this paper.

Consensus as smart contracts. To the best of our knowledge, this is the first work to build consensus protocols using smart contracts. The inherent limitations to smart contract programming (e.g., single-threading) pose challenges to consensus protocol implementation. A prominent challenge is to ensure that every operation in the BFT protocol is properly triggered by some IBC message. Fortunately, this is the case in IT-HS (as it is optimistically responsive). The only exception is the timeout for the view change, which occurs when the leader is malicious or the network is poor, thus this is not in the optimistic path. Particularly, in IT-HS, a party will enter the view change phase if the leader of current view does not make any progress for a certain period of time (a pre-defined timeout value). However, this is not triggered by any IBC message. To address this issue, we set external bots/scripts to regularly ping the blockchains to trigger the timeout in time. Note that we make no trust assumptions on these bots (they can also be replaced by "keepers", a recent proposal from Chainlink [10]).

Another limitation of the IBC protocol is that a blockchain can not send IBC messages to itself. However, in many BFT protocols including IT-HS, there are operations triggered by self-delivered messages. Therefore, we have to pay special attention to the self-delivered messages when implementing IT-HS in TrustBoost. In addition, Cosmos-SDK allows the smart contract to send IBC messages only when a function returns; once that is complete, the function call is over and no more operations can be conducted. To deal with these constraints in Cosmos, we use a *message queue* in each function, queue all the IBC messages that need to be sent, and send them all when the function returns. For example, in IT-HS, when a node receives a propose message from the current leader, it will send a echo message to all the other nodes; and when a node receives $2f + 1$ echo messages, it will send a key1 message to all the other nodes. Extra caution is warranted when writing the send_proposal function in TrustBoost: in the send_proposal function, the current leader blockchain would like to broadcast the propose message, but can only do this when the function returns. So, we need to first put the propose message into the message queue. Considering that the leader blockchain should send an echo message when the propose message is self-delivered, we also put the echo message into the message queue. And because the echo message will also be self-delivered, we increase the counter for echo

messages by one. Since there are not enough echo messages yet, the next step (sending the key1 message) will not be triggered, the send_proposal function can finally return and the leader blockchain can send the IBC messages in the message queue (propose and echo). Several such subtle aspects abound in the TrustBoost implementation, making the design and implementation a challenging and rewarding endeavor; as such, we believe how to implement complex communication/distributed protocols using smart contracts is of independent interest.

TrustBoost **as a proxy for application contracts.** In order to boost the security promises, any single-chain application contract App can be equipped with a TrustBoost proxy to become a cross-chain application contract AppX without touching functional codes. Specifically, a TrustBoost proxy will issue a function call to AppX contract after committing a client's request. To support this, AppX contract only modifies a few lines (highlighted in Fig. 5 (Right)) in the handler function of App contract to (1) check the function call is initiated by a certified TrustBoost contract; and (2) add contract information `info` of App contract as an extra parameter of AppX contract (stored in `executeMsg.TBInfo`) to reproduce single-chain executions.

Contract state. Though an application contract App and its cross-chain variant AppX provide the same functionality, they have to maintain isolated states. Regardless of the states owned by an App contract already existing on any single blockchain, the deployment of a new AppX contract initializes all related states independently from scratch, which are secured by more stringent security rules.

7 EVALUATION

Our experimental evaluation answers the following questions:

- What is overhead of TrustBoost in terms of gas usage and confirmation latency? This is the price paid for security.
- How well does TrustBoost scale when the number of chains increases?
- How does TrustBoost perform under Byzantine attacks?

Testbed setup. We deploy TrustBoost on an AWS m5.4xlarge instance with 16 vCPU and 64 GB memory. There are three steps in the setup phase: 1) start $m = 3f + 1$ Cosmos chain instances in our local testnet; 2) deploy TrustBoost and NameService contracts on each of the m chains; 3) connect each pair of the m chains with an IBC channel. In our experiments, the block rate of each Cosmos chain is set to be about one block per second.

Performance. In this experiment, we evaluate the performance of TrustBoost with $m = 4, 7, 10$. We measure the number of IBC messages, the total gas usage and the confirmation latency per request. For the confirmation latency, we record the duration between the submission and the execution of the request. We repeat the experiment on the single-chain NameService itself without using TrustBoost (i.e., $m = 1$) for comparison (results in Table 1).

From Table 1, we see that the number of IBC messages and the gas usage scale quadratically in the number of chains. The overhead in gas comes from two parts: 1) the communications and computations in the TrustBoost contract cost gas; 2) the NameService contract needs to be executed on all m chains. Note that for fixed m, the former gas usage is a constant, independent of the application contracts. And in our experiments, the NameService contract uses very little gas, so the overhead caused by TrustBoost is dominating. Further, by batching the requests, this overhead can be amortized. Also note that the performance of TrustBoost is also limited by IBC efficiency: sending one single IBC message to transfer tokens cross chains would need 2s and 350K gas. Hence, making the transmission and execution of IBC messages more efficient could greatly improve the performance of TrustBoost. Moreover, 600M gas only costs $2-$10, for example based on the gas price and the exchange rate on the Osmosis chain, at the time of writing (April 2023). The latency is independent of the application contract. However, we can see that it scales almost linearly when m increases, and it is pretty acceptable compared with other high security chains, such as Ethereum.

m	1	4	7	10
# IBC	0	102	348	738
Gas usage	202K	74M	261M	586M
Latency	2.5s	67.2s	105.0s	138.2s

Table 1: Performance of TrustBoost with different number of chains.

Attacks	I	II	III	IV
Latency	137.2s	138.6s	66.8s	66.0s

Table 2: Performance of TrustBoost under different attacks.

Security. In this experiment, TrustBoost is evaluated under active attacks, specifically within a four-chain scenario where one blockchain is compromised by an attacker and may behave arbitrarily. The malicious behaviors of the compromised blockchain are triggered by external calls. Just like many other BFT protocols, IT-HS also has a view-change sub-protocol to ensure liveness: when no progress is made in one view, all nodes will enter the next view by broadcasting abort messages; each view is assigned with a predefined primary node. We test the following attacks.

- **Attack I - Primary blockchain crashes.** In this attack, the primary blockchain of the first view crashes at the beginning, therefore progress can only be made in the second view.
- **Attack II- Primary blockchain equivocates.** In this attack, the primary blockchain of the first view sends different proposals to different non-primary blockchains in the network.
- **Attack III - Non-primary blockchain equivocates.** In this attack, a non-primary blockchain of the first view sends different votes to different blockchains in the network.
- **Attack IV - Non-primary blockchain aborts.** In this attack, a non-primary blockchain of the first view keeps sending abort messages to enter the view change phase.

We measure the confirmation latency under these attacks. The results are shown in Table 2. We can see that TrustBoost still works under these attacks: the confirmation latency doubles under the first two attacks as it takes two views to terminate; and attacks III and IV hardly have any impact on the latency.

8 DISCUSSION

Unequal weights. Just like many BFT-style PoS protocols [23], blockchains in TrustBoost can also have different weights, i.e., the

voting powers in the BFT protocol. For example, if all constituent chains are PoS chains, we can set the weight of a chain to be proportional to the total market cap of its native token. This strategy is also adopted in recent research [49], which interprets the stake required for an adversary to compromise safety as a measure of economic security. This enables the comparison of security levels among various PoS chains, aiming to maximize the expense required to attack the TrustBoost ledger. Another example is that we can set the weight of all chains except one strong chain to be zero so that TrustBoost can directly borrow trust from the strong chain. How to dynamically adjust the weights is also an interesting question. Moreover, today many blockchains are heavily intertwined economically (e.g., Osmosis and Axelar Network in Cosmos), so the idea of asymmetric trust [7] can also be brought to TrustBoost. We defer these topics to the future work.

Share security via checkpointing. An important application of TrustBoost is that we can use it to checkpoint the m constituent chains or other weak chains. The validators of each chain just need to regularly submit block hashes and signatures as checkpoints to the TrustBoost ledger, and the finality rule of each chain will be altered to respect the checkpoints. In this way, each chain will also have a slightly slower finality rule - confirming the chain up to the latest checkpoint, which has the same latency and security level as TrustBoost. For high value transactions on a constituent chain, the users can apply the slow finality rule to enjoy stronger security guarantees. In the context of Cosmos, with this approach each constituent Cosmos chain in TrustBoost will have slashable safety and much shorter withdrawal delays as long as at most 1/3 of the chains are faulty, following the results shown in [50].

Cross chain applications. We note that an important application of cross chain bridges today is cross chain token transfer. However, it has a fundamental security limit, where the attacker transfers some tokens on chain 1 to chain 2 and then reverts the state on chain 1 (e.g., by doing 51% attack) to get his tokens back. We point out that this issue can be alleviated by TrustBoost using the checkpointing idea discussed above. Particularly, when a user wants to move 100 token ABC from chain 1 to get 100 token XYZ on chain 2, first the 100 token ABC will be locked on chain 1 and then token XYZ will be sent to the user only when the lock transaction on chain 1 is confirmed by the slow finality rule (i.e., in the checkpointed chain). In this way, the state of the token ABC contract on chain 1 is implicitly upgraded into the global state of the TrustBoost ledger, which has stronger security guarantees.

Heterogeneous chains. Although our TrustBoost implementation is in the Cosmos ecosystem, the idea can be extended to a heterogeneous blockchain network. We just need all the heterogeneous chains to be programmable and interoperable. Different blockchains may have different virtual machines, therefore the TrustBoost smart contract will need to be written in multiple programming languages. On the other hand, there are quite a few ongoing projects using zk-SNARKs to build trustless and efficient cross chain bridges [34, 54]. We believe TrustBoost will have broader applications in the near future when the heterogeneous blockchain network becomes mature.

9 ACKNOWLEDGEMENTS

We thank Dion Hiananto and Luhao Wang for their help with the implementation and experiments. We thank Jack Zampolin and a few other Cosmos core developers for valuable suggestions on this project. This research is supported in part by the US National Science Foundation under grants CCF-1705007 and CNS-1718270, the US Army Research Office under grant W911NF1810332 and W911NF2310147 and a gift from XinFin Private Limited.

REFERENCES

[1] Ittai Abraham, Gregory V Chockler, Idit Keidar, and Dahlia Malkhi. Byzantine disk paxos: optimal resilience with byzantine shared memory. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 226–235, 2004.
[2] Ittai Abraham and Gilad Stern. Information theoretic hotstuff. *arXiv preprint arXiv:2009.12828*, 2020.
[3] Yair Amir, Claudiu Danilov, Jonathan Kirsch, John Lane, Danny Dolev, Cristina Nita-Rotaru, Josh Olsen, and David Zage. Scaling byzantine fault-tolerant replication to wide area networks. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 105–114. IEEE, 2006.
[4] Mathieu Baudet, George Danezis, and Alberto Sonnino. Fastpay: High-performance byzantine fault tolerant settlement. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 163–177, 2020.
[5] PolyNetwork Bridge. Polynetwork. https://poly.network/.
[6] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.
[7] Christian Cachin. Asymmetric distributed trust. In *International Conference on Distributed Computing and Networking 2021*, pages 3–3, 2021.
[8] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.
[9] Chainlink. Ccip. https://chain.link/cross-chain/.
[10] Chainlink. Chainlink keepers. https://docs.chain.link/docs/chainlink-keepers/introduction.
[11] Gregory Chockler and Dahlia Malkhi. Active disk paxos with infinitely many processes. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 78–87, 2002.
[12] Shir Cohen and Idit Keidar. Tame the wild with byzantine linearizability: Reliable broadcast, snapshots, and asset transfer. *arXiv preprint arXiv:2102.10597*, 2021.
[13] Cosmos. Cosmos-sdk. https://github.com/cosmos/cosmos-sdk.
[14] Cosmos. Cosmwasm. https://github.com/CosmWasm/cosmwasm.
[15] Cosmos. Ibc. https://github.com/cosmos/ibc.
[16] Cosmos. Ics ?: Recursive tendermint. https://github.com/cosmos/ibc/issues/547.
[17] Cosmos. Relayer. https://github.com/cosmos/relayer.
[18] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.
[19] Michael J Fischer, Nancy A Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.
[20] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.
[21] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ledger combiners for fast settlement. In *Theory of Cryptography Conference*, pages 322–352. Springer, 2020.
[22] Eli Gafni and Leslie Lamport. Disk paxos. In *Distributed Computing: 14th International Conference, DISC 2000, Maurice Herlihy, editor. Lecture Notes in Computer Science number 1914, Springer-Verlag,(2000) 330-344.*, 2003.
[23] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.
[24] Christopher Goes. The interblockchain communication protocol: An overview. *arXiv preprint arXiv:2006.15918*, 2020.
[25] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 307–316, 2019.
[26] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019.
[27] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. Resilientdb: Global scale resilient blockchain fabric. *arXiv preprint arXiv:2002.00160*, 2020.

[28] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(1):124–149, 1991.
[29] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*, pages 245–254, 2018.
[30] Maurice Herlihy, Nir Shavit, Victor Luchangco, and Michael Spear. *The art of multiprocessor programming*. Newnes, 2020.
[31] Dimitris Karakostas and Aggelos Kiayias. Securing proof-of-work ledgers via checkpointing. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5. IEEE, 2021.
[32] Jae Kwon and Ethan Buchman. Cosmos whitepaper. *A Netw. Distrib. Ledgers*, 2019.
[33] Deus Labs. Nameservice contract in rust using cosmwasm. https://github.com/deus-labs/cw-contracts/tree/main/contracts/nameservice.
[34] Succinct Labs. Towards the endgame of blockchain interoperability with proof of consensus. https://blog.succinct.xyz/blog/endgame.
[35] Leslie Lamport. *On interprocess communication*. Digital Equipment Corporation Systems Research Center, 1985.
[36] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.
[37] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.
[38] layerzero Bridge. Layerzero. https://layerzero.network/.
[39] Michael C Loui and Hosame H Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing research*, 4(163-183):31, 1987.
[40] Nancy A Lynch and Mark R Tuttle. *An introduction to input/output automata*. Laboratory for Computer Science, Massachusetts Institute of Technology, 1988.
[41] Kartik Nayak and Ittai Abraham. Consensus for state machine replication. https://decentralizedthoughts.github.io/2019-10-15-consensus-for-state-machine-replication/.
[42] NEAR. Near rainbow bridge. https://near.org/bridge/.
[43] Nomad. Nomad bridge. https://docs.nomad.xyz/nomad-101/introduction.
[44] Polkadot. Xcm. https://polkadot.network/cross-chain-communication/.
[45] Ranvir Rana, Dimitris Karakostas, Sreeram Kannan, Aggelos Kiayias, and Pramod Viswanath. Optimal bootstrapping of pow blockchains. In *Proceedings of the Twenty-Third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pages 231–240, 2022.
[46] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. Blockchain cap theorem allows user-dependent adaptivity and finality. In *International Conference on Financial Cryptography and Data Security*, pages 84–103. Springer, 2021.
[47] Jakub Sliwinski and Roger Wattenhofer. Abc: Proof-of-stake without consensus. *arXiv preprint arXiv:1909.10926*, 2019.
[48] Informal Systems. An overview of interchain security v1. https://informal.systems/2022/02/02/interchain-security-v1/.
[49] Ertem Nusret Tas, Runchao Han, David Tse, Fisher Yu, and Kamilla Nazirkhanova. Interchain timestamping for mesh security. *arXiv preprint arXiv:2305.07830*, 2023.
[50] Ertem Nusret Tas, David Tse, Fisher Yu, Sreeram Kannan, and Mohammad Ali Maddah-Ali. Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. *arXiv preprint arXiv:2207.08392*, 2022.
[51] Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. Universal atomic swaps: Secure exchange of coins across all blockchains. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1299–1316. IEEE, 2022.
[52] TrustBoost. Trustboost contract in rust. https://github.com/trustboost/TrustBoost.
[53] Wormhole. Wormhole bridge. https://wormhole.com/.
[54] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. *arXiv preprint arXiv:2210.00264*, 2022.
[55] Yingjie Xue and Maurice Herlihy. Cross-chain state machine replication. *arXiv preprint arXiv:2206.07042*, 2022.
[56] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.
[57] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. Sok: Communication across distributed ledgers. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 3–36. Springer, 2021.

A THE LIMITATION OF LEDGER COMBINER

The previous work on robust ledger combiner [21] proposes that in passive mode, when $m \geq 2f + 1$, a combined ledger with relative settlement can be constructed. This relative settlement definition asserts that if a transaction is relatively settled, it will remain reliably settled in the ledger from that point forward, and no conflicting transactions will appear in the ledger unconsciously. However, this definition is notably less robust than our definition (see Definition 2) of ABC. Specifically, ABC includes the assurance of honest termination, meaning that if every honest object receives the same transaction (and no conflicting ones), that transaction will be committed by all honest processes. Unfortunately, relative settlement falls short of guaranteeing this essential property.

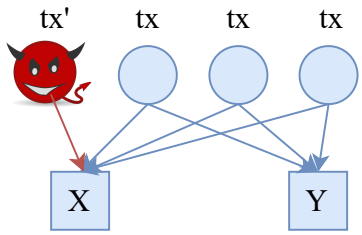

Figure 7: An example with $m = 4$ objects and $f = 1$ Byzantine object. All honest objects receive and commit the same transaction tx; only clients in X see the conflicting transaction tx' committed on the Byzantine object.

Consider the example illustrated in Figure 7, where $m = 4$ and $f = 1$. Here, three honest objects receive the identical transaction tx, and each commits tx. Processes in groups X and Y both observe this. However, suppose a Byzantine object commits a conflicting transaction tx' but only reveals it to group X. In our TrustBoost-Lite protocol, both X and Y will commit tx, as honest objects received the same transaction and no conflicting transactions. In contrast, under the ledger combiner model of [21], only group Y will commit tx, while group X will not commit tx because they observed a conflicting transaction tx'. This divergence violates the honest termination property.

Furthermore, Theorem 4.1 in our work presents a profound implication: ABC is unachievable when $m \leq 3f$. Since the model in [21] assumes $m \geq 2f + 1$, it follows that the protocol they present cannot successfully solve ABC according to our theoretical lower bound.