TrustBoost: Boosting Trust among Interoperable Blockchains

Xuechao Wang[1], Peiyao Sheng[1], Sreeram Kannan[2], Kartik Nayak[3], and Pramod Viswanath[4]

[1] University of Illinois, Urbana-Champaign, IL
xuechao2,psheng2@illinois.edu
[2] University of Washington, Seattle, WA
ksreeram@ece.uw.edu
[3] Duke University, Durham, NC
kartik@cs.duke.edu
[4] Princeton University, Princeton, NJ
pramodv@princeton.edu

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. Furthermore, the protocol 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 [31]. 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.

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 [19,26,27,30] – 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. 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 [29] – 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 (i.e., constituent blockchains do not give up their individual sovereignty and there is no change to the consensus layer). 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? Note that the adversary can collude across the f faulty blockchains. 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 different 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 [9], XCM by Polkadot [25], and CCIP by Chainlink [5]. 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. 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 [14], tried to create a combined ledger passively without success. Focusing on a weaker form of consensus that gives up the total ordering property (but can still implement the functionality of a cryptocurrency), known as ABC consensus [28], we show the following in the passive mode. First, even ABC consensus is impossible, if $m \leq 3f$. Second, we propose a protocol called TrustBoost-Lite that combines different ledgers to achieve ABC consensus 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 authen-

ticated channels provided via the CCC infrastructure – a form of "consensus on top of consensus", a first in the literature, to the best of our knowledge. Finally, we observe that our study of blockchain interactions has a side effect of bringing to sharp focus the nuanced definitions of traditional and the weakened ABC consensus in terms of the interactions needed to achieve combined trust.

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 [31].

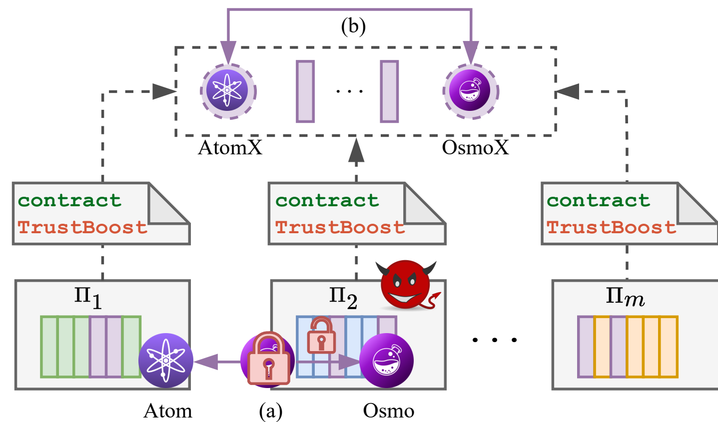

Fig. 1: (a) Token exchange across chains are vulnerable to single-chain attacks. Suppose attackers lock 100 Osmos on the Osmosis chain in exchange for 10 Atoms on the Cosmos chain. Once Atoms are received, a double-spend attack on the transaction which locks 100 Osmos on the Osmosis chain leads to 10 "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.

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 [1] 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[21] – 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 (October 2022) 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.

Related works. An early work on robust ledger combining is [14]; parallel ledgers process a common set of transactions independently, and confirmation in the combined ledger is done by observers who can read from all ledgers. However, the combined ledger fails to guarantee agreement (termed absolute persistence in [14]), so its practical use is limited. A very recent work [33] 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 [33] hold only when *each* of the involved blockchains is secure (as expected by one of our theoretical results (cf. Theorem 1)). As mentioned earlier, checkpointing is a natural way to transfer the trust of one (very secure) blockchain to weaker/newer blockchains [27,19,26]. A concrete and practical instantiation of this idea in the context of bringing Bitcoin trust to Cosmos Zones is [30].

2 Preliminaries

Since our goal is to achieve consensus on top of consensus (blockchains), it is natural to model blockchains as *validators* in the "meta" consensus protocol, and blockchain users as *clients* that can query states of blockchains via RPCs. Hence, we have the following participation and network model.

2.1 Model

Participants. In this paper, the distributed protocols involve two types of participants: (i) validators, and (ii) clients. We assume Byzantine faults that can behave arbitrarily. Let m be the number of validators and f be the number of Byzantine validators. We assume no trust from the clients, i.e., we allow any

number of Byzantine clients, but only honest clients enjoy the guarantees of the protocol. We suppose one of the following two prerequisites is met: (1) all participants are connected via authenticated channels (see [23]); (2) there exists a public-key infrastructure (PKI) for the set of validators, i.e., the public keys of all validators are known to all validators and clients.

Network model. Validators are in a fully connected network among themselves. We say the validators are in a *passive mode* if they do not communicate with each other; otherwise they are in an *active mode*. Each client is also connected to all the validators – clients only receive messages from the validators, but do not transmit any messages to the validators. Note that allowing bidirectional communications between validators and clients would turn the passive mode into the active mode as clients can help forward messages. We assume the network is partially synchronous, i.e., there is a global stabilization time (GST) chosen by the adversary, unknown to the honest parties and also to the protocol designer, such that after GST, all messages sent between honest parties are delivered within Δ time. Before GST, the adversary can delay messages arbitrarily. When $GST = 0$, the network becomes synchronous.

2.2 Distributed protocols

To study the security guarantee of the "meta" consensus protocol, we define the following client consensus problems.

Client consensus. We start with the problem of (binary) *client consensus* where each validator starts with some initial value (0 or 1) and all (honest) clients try to commit the same value by the end of the protocol.

Definition 1 (Client Consensus)**.**

- ***Agreement****: No conflicting values are committed by honest clients.*
- ***Validity****: If every honest validator starts with the same value, this value will be committed by honest clients.*
- ***Termination****: Every honest client commits one of the values. If messages are delivered quickly, the consensus protocol terminates quickly.*

We are also interested in a relaxed notion of client consensus, called *ABC client consensus*, derived from [28]. The key insight leading to this relaxation is that Byzantine parties do not need to enjoy any guarantees.

Definition 2 (ABC Client Consensus)**.**

- ***Agreement****: Same as in Definition 1.*
- ***Validity****: Same as in Definition 1.*
- ***Honest termination****: If every honest validator starts with the same value, this value will be committed by honest clients. If messages are delivered quickly, the ABC consensus protocol terminates quickly.*

Contrast with traditional consensus protocols. In the traditional (ABC) consensus problem [11,28], there is no client and the guarantees are enjoyed by

the validators. It is not hard to see that any traditional (ABC) consensus protocol can be used to solve (ABC) client consensus in our setting: Let $3f+1$ validators run the traditional (ABC) consensus consensus protocol and send the committed value to all clients; each client commits the majority value received from those $3f+1$ validators (note that solving both consensus and ABC consensus would need $m > 3f$ in partial synchrony). Therefore, any impossibility result (necessary condition) for the problem of (ABC) client consensus consensus in our model also applies to traditional (ABC) consensus. On the other hand, solving the problem of client (ABC) consensus in our model does not necessarily solve traditional (ABC) consensus. Indeed, in the above example, the other $m-3f-1$ validators do not even participate in the protocol so that they cannot commit any value. In §3, we will also see that ABC consensus can be solved in our model without any communication among validators. However, this is not possible in the traditional model because, without communication, basically nothing can be done in a distributed system.

Ledger consensus. It is known that consensus protocols can be used to solve the problem of *ledger consensus*, i.e., *state machine replication (SMR)* [24]. In our model, we also slightly modify the definition of ledger consensus such that it now requires all clients maintain a list of transactions that grows in length, called a *public ledger* (cf. Definition 3), with the help of the validators. Validators participate in the protocol to decide a total ordering of all transactions. Each client initially starts with the same state and updates the state by executing all transactions in its ledger. The definition of ledger consensus in our model and a comparison with the traditional definition can be found in Appendix A.

3 Necessary and Sufficient Conditions

In this section, we show that in the passive mode (i.e., without communication among validators), client consensus (cf. Definition 1) is impossible, while ABC client 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 [12,13] to our nuanced model/definitions and show the following tight results and sharp distinction between consensus and ABC consensus. The full proof of Theorem 1 can be found in Appendix B. We provide a proof sketch below.

Theorem 1. *In passive mode, we have*

1. *Client consensus can be achieved if and only if $f = 0$.*
2. *ABC client consensus can be achieved if and only if $m > 3f$.*

Proof sketch. (1) Client consensus: We apply the proof idea from [13] to find two adjacent initial configurations such that: they differ only in the initial value x_p of a single validator p, but reach different decision values. Then if p equivocates, the agreement property will be violated.

(2) ABC client consensus: The proof is similar as for client consensus. The only difference is that now the ABC client consensus protocol will terminate only if

all $m - f$ honest validators start with the same value. When $f \geq m/3$, we can divide the m validators into three sets, each with at most f validators, and the rest of the proof remains the same.

Remark. Note that in the proof of only if parts in Theorem 1, the messages are never delayed by the adversary. Therefore, the same proofs apply to a synchronous network (i.e., $GST = 0$). The robust ledger combiner in [14] is proposed precisely in this model; thus we can conclude that the robust ledger combiner (with $m = 2f+1$) cannot solve ABC client consensus, let alone client consensus.
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 [12]. The proof of Theorem 2 can be found in Appendix C.

Theorem 2. *In the active mode, we have:*

1. *Client consensus can be achieved if and only if $m > 3f$.*
2. *ABC client consensus can be achieved if and only if $m > 3f$.*

4 The TrustBoost Protocol

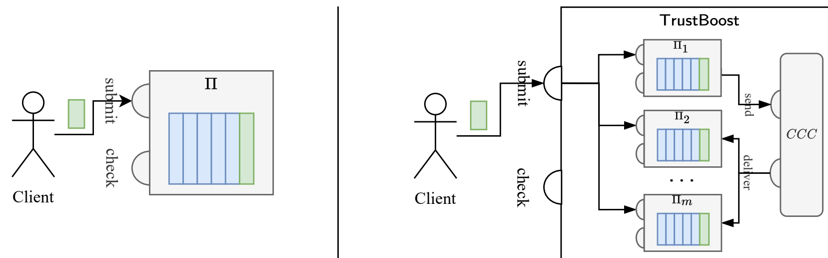

Fig. 2: Clients see the same interface of submitting a transaction to TrustBoost as submitting a transaction to a single blockchain.

4.1 TrustBoost: Blockchains as Validators

TrustBoost is run by m blockchains, simulating a group of validators interacting with each other to maintain global states. The i-th blockchain ($1 \leq i \leq m$) is run by a permissioned committee C_i with $|C_i| = n_i$ of which at most f_i nodes are Byzantine. 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) run by nodes in the committee C_k, it provides an interface $\Pi_k.\mathsf{submit}(tx)$ for any client to submit and commit transactions on the blockchain. And clients can check whether a transaction tx is committed on the ledger through $\Pi_k.\mathsf{check}(tx)$. Optionally, $\Pi_k.\mathsf{check}(tx)$ will return a commitment certificate generated by the protocol as a proof of commitment, e.g., the quorum certificate in many BFT-SMR protocols. Π_k can be instantiated by a black-box partial synchronous consensus protocol.
- **Cross-chain communication (CCC).** CCC is a routing protocol that allows independent blockchains to communicate with each other, it has two primitives, $CCC.\mathsf{send}(src, dst, tx)$ and $CCC.\mathsf{deliver}(src, dst, tx)$ to transmit transaction tx, where src and dst are identifiers of the source and destination blockchains. CCC guarantees reliability and authenticity. Reliability implies that any transaction sent by the source chain will get delivered in the destination chain. Authenticity means that any validator on the destination chain can verify the transactions delivered by the protocol represent some states committed on the source chain.

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 and an interface TrustBoost.check(tx) for clients to check the commitment of transactions (see Fig. 2). By using CCC as transmission channels, and invoking $\Pi_k(1 \leq k \leq m)$ to commit transactions, any partial synchronous protocol consensus can be used to instantiate TrustBoost.

For simplicity, we show how to instantiate TrustBoost with a majority voting protocol in Algorithm 1, which is not a complete consensus protocol but contains basic building blocks (propose, vote and commit phases) of most consensus protocols and the usage of all primitives. In the beginning, the protocol initializes some data structures to store votes for transactions. Whenever a client wants to post a transaction tx on TrustBoost blockchains, it calls TrustBoost.submit function (line 4), which triggers CCC to broadcast a proposal to all blockchains. Lines 7-14 describe how to handle cross-chain transactions such as Propose and Vote. Once the commitment condition is met, e.g., in this example, a super majority of votes are collected, the TrustBoost protocol commits the transaction by asking the local blockchain to commit the transaction (line 14). The protocol also provides a check function (line 15) to extract 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 1/3 of the local blockchains. More complicated rules can be designed in accordance to the specifications of various consensus protocols. For instance, when instantiating TrustBoost with BFT protocols that use signatures, we need π_k that can provide commitment certificates to allow message authentication after being forwarded to some third chain.

Security guarantee. We see that each blockchain in TrustBoost behaves the same as a single validator: transactions are authenticated by local commitment; a

blockchain with honest super-majority ($n_i > 3f_i$) will behave in the same way as an honest validator (e.g., no equivocation), otherwise it may behave arbitrarily (like a Byzantine validator). Thus, let $f = |\{i : n_i \leq 3f_i, 1 \leq i \leq m\}|$ be the number of faulty blockchains. If $m > 3f$, TrustBoost solves the problem of client ledger consensus as defined in Definition 4 (see Appendix A). In other words, TrustBoost securely constructs a combined ledger with total ordering as long as 2/3 of blockchains are secure.

Algorithm 1 Protocol TrustBoost (CCC, k, Π_k)

```
 1: Init:
 2:     txVotes ← an empty map
 3:     m ← number of participating chains
 4: function submit(tx)
 5:     for dst = 1, ··· , m do
 6:         invoke CCC.send(k, dst, ⟨Propose, tx⟩)
 7: upon event CCC.deliver(src, dst, ⟨Propose, tx⟩) do
 8:     txVotes[tx] = ∅
 9:     for dst = 1, ··· , m do
10:         invoke CCC.send(k, dst, ⟨Vote, tx⟩)
11: upon event CCC.deliver(src, k, ⟨Vote, tx⟩) do
12:     txVotes[tx] = txVotes[tx] ∪ {src}
13:     if |txVotes[tx]| = ⌊2m/3⌋ + 1 then
14:         invoke Π_k.submit(tx)
15: function check(tx)
16:     cnt ← 0
17:     for k = 1, ··· , m do          ▷ These m queries are executed concurrently
18:         if Π_k.check(tx) returns true then
19:             cnt ← cnt + 1
20:     if cnt ≥ ⌊m/3⌋ + 1 then
21:         return true
22:     return false
```

4.2 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 committee 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 provides the same interfaces (submit and check) as TrustBoost, which are defined in Algorithm 2.

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).

Algorithm 2 Protocol TrustBoost-Lite (Π_k)

```
1:  Init:
2:      m ← number of participating chains
3:  function submit(tx)
4:      for dst = 1, ··· , m do
5:          invoke Π_k.submit(tx)
6:  function check(tx)
7:      cnt ← 0
8:      for k = 1, ··· , m do                ▷ These m queries are executed concurrently
9:          if Π_k.check(tx) returns true then
10:             cnt ← cnt + 1
11:     if cnt ≥ ⌊2m/3⌋ + 1 and valid(tx) then
12:         return true
13:     return false
14: function valid(tx)
15:     (in, out) ← tx
16:     valid ← true
17:     for input ∈ in do
18:         if check(input.from) returns false then
19:             valid ← false
20:     return valid
```

Security guarantee. The behaviors of blockchains that run TrustBoost-Lite are also determined by local consensus protocols. Similarly, let $f = |\{i : n_i \leq 3f_i, 1 \leq i \leq m\}|$ be the number of faulty blockchains. If $m > 3f$, TrustBoost-Lite solves the problem of ABC client ledger consensus as defined in Definition 5 (see Appendix A), which means honest clients still see the same set of transactions and their transactions will be committed by TrustBoost-Lite even if 1/3 of local blockchains are insecure.

5 Implementation

We implement TrustBoost in the Cosmos ecosystem [20], which is a decentralized network of parallel and interoperable blockchains, each powered by BFT consensus protocols like Tendermint [3], where the CCC is enabled by an inter-blockchain communication (IBC) protocol [16]. In this section, we first give a

brief overview of the Cosmos ecosystem and then highlight the key challenges to implement TrustBoost.

5.1 Cosmos overview

Cosmos SDK. Cosmos-SDK [7] provides tools for building permissioned/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 [8] and the IBC module [9] (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. 3 (Left), by processing two given parameters[5]. 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.

```
1  fn execute(info, executeMsg) {
2      let funcName = executeMsg.funcName;
3      let param = executeMsg.param;
4
5
6
7
8
9      match executeMsg {
10         funcA -> funcA(info, param),
11         funcB -> funcB(info, param),
12         ...
13     }
14 }
15 // 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...
```

Fig. 3: Left: Handler of CosmWasm contract. Right: Handler of AppX contract with TrustBoost proxy.

IBC protocol. IBC is an interoperability protocol for communicating arbitrary data between 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

[5] We omit some semantic details for simplicity, check full descriptions in [8].

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 [10] 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 20 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.

5.2 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. 4. 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[6].

Next, we discuss key challenges of implementing TrustBoost.

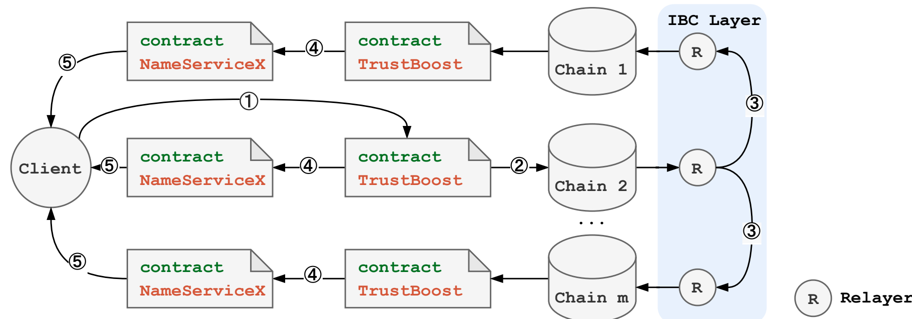

Fig. 4: The flow chart of executing a TrustBoost transaction.

Use PKI or not? In the relatively recent blockchain era, partially synchronous BFT protocols have received renewed attention; performant and efficient BFT

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

protocols have been constructed (e.g., SBFT [18], Tendermint [3] and HotStuff [34]). 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) [1]. 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. 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 [6]).

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. 3 (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.

6 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 also do the same experiment on the single-chain NameService itself without using TrustBoost (i.e., $m = 1$) for comparison. The results are shown in Table 1 (Left).

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

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

Table 1: Left: Performance of TrustBoost with different number of chains. Right: Performance of TrustBoost under different attacks.

From Table 1 (Left), we see that the number of IBC messages and the gas usage scale quadratically. 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 (October 2022). 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 Bitcoin and Ethereum.

Security. In this experiment, we evaluate how TrustBoost performs under active attacks. We do the experiments in the four-chain cases where one blockchain is hacked by the attacker and may behave arbitrarily. The malicious behaviors of the hacked 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; each view is assigned with a predefined primary node. We test the following attacks.

- **Attack I.** The primary blockchain of the first view crashes.
- **Attack II.** The primary blockchain of the first view sends different proposals.

- **Attack III.** A non-primary blockchain sends different votes.
- **Attack IV.** A non-primary blockchain keeps sending abort messages to enter the view change phase.

We measure the confirmation latency under these attacks. The results are shown in Table 1 (Right). 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.

7 Discussion

Unequal weights. Just like many BFT-style PoS protocols [15], 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. In this way, we can maximize the cost 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 [4] can also be brought to TrustBoost. We defer these topics to the future work.
Consensus with less communication and connectivity. In §3, we have seen that ABC client consensus (cf. Definition 2) can be solved without any communication among the validators. One natural question would be: Can we design new client 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 client consensus. For example, it is easy to see that for client consensus when $m > 3f + 1$, the extra $m - 3f - 1$ validators do not even to connect with other validators, while all honest validators must form a connected component in order to solve the traditional consensus problem. However, identifying the minimum requirement on network connectivity for client consensus remains an interesting and challenging problem.
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 [30].

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 zkSNARKs to build trustless and efficient cross chain bridges [22,32]. We believe TrustBoost will have broader applications in the near future when the heterogeneous blockchain network becomes mature.

8 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 and the US Army Research Office under grant W911NF1810332.

References

1. Ittai Abraham and Gilad Stern. Information theoretic hotstuff. *arXiv preprint arXiv:2009.12828*, 2020.
2. 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.
3. Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.
4. Christian Cachin. Asymmetric distributed trust. In *International Conference on Distributed Computing and Networking 2021*, pages 3–3, 2021.
5. Chainlink. Ccip. https://chain.link/cross-chain/.
6. Chainlink. Chainlink keepers. https://docs.chain.link/docs/chainlink-keepers/introduction.
7. Cosmos. Cosmos-sdk. https://github.com/cosmos/cosmos-sdk.
8. Cosmos. Cosmwasm. https://github.com/CosmWasm/cosmwasm.

9. Cosmos. Ibc. https://github.com/cosmos/ibc.
10. Cosmos. Relayer. https://github.com/cosmos/relayer.
11. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
12. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.
13. 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.
14. Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ledger combiners for fast settlement. In *Theory of Cryptography Conference*, pages 322–352. Springer, 2020.
15. 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.
16. Christopher Goes. The interblockchain communication protocol: An overview. *arXiv preprint arXiv:2006.15918*, 2020.
17. 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.
18. 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.
19. 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.
20. Jae Kwon and Ethan Buchman. Cosmos whitepaper. *A Netw. Distrib. Ledgers*, 2019.
21. Deus Labs. Nameservice contract in rust using cosmwasm.
22. Succinct Labs. Towards the endgame of blockchain interoperability with proof of consensus. https://www.succinct.xyz/post/2022/09/20/proof-of-consensus/.
23. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.
24. Kartik Nayak and Ittai Abraham. Consensus for state machine replication. https://decentralizedthoughts.github.io/2019-10-15-consensus-for-state-machine-replication/.
25. Polkadot. Xcm. https://polkadot.network/cross-chain-communication/.
26. 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.
27. 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.
28. Jakub Sliwinski and Roger Wattenhofer. Abc: Proof-of-stake without consensus. *arXiv preprint arXiv:1909.10926*, 2019.

29. Informal Systems. An overview of interchain security v1. https://informal.systems/2022/02/02/interchain-security-v1/.
30. 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.
31. TrustBoost. Trustboost contract in rust. https://github.com/trustboost/TrustBoost.
32. 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.
33. Yingjie Xue and Maurice Herlihy. Cross-chain state machine replication. *arXiv preprint arXiv:2206.07042*, 2022.
34. 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.

Appendix

A Definitions of Ledger Consensus

It is known that consensus protocols can be used to solve the problem of *ledger consensus*, i.e., *state machine replication (SMR)* [24]. In our model, we also slightly modify the definition of ledger consensus such that it now requires all clients to maintain a list of transactions that grows in length, called a *public ledger* (cf. Definition 3), with the help of the validators. Validators participate in the protocol to decide a total ordering of all transactions. Each client 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 ledger $\mathcal{L}$ is a growing list of transactions that provides the following two client interfaces:*

- *Submit: a client can submit a transaction tx to $\mathcal{L}$ by calling* $\mathsf{submit}(tx)$*.*
- *Check: a client can check whether $tx \in \mathcal{L}$ by calling* $\mathsf{check}(tx)$*. If* $\mathsf{check}(tx)$ *returns true, the client will commit tx.*

In the problem of client ledger consensus, we allow the following synchronous out-of-band communications among clients: 1) clients can send transactions to validators, but still clients send no other message to validators; 2) clients 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 clients). The guarantees of client ledger consensus are defined as follows.

Definition 4 (Client ledger consensus)**.**

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

- ***Termination****: If a client submits tx to all honest validators, tx will be committed by all honest clients. If messages are delivered quickly, tx will be committed quickly.*

Similarly, ABC consensus 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 [28,2,17]. Suppose the transaction space is now equipped with a conflict relation (e.g. double spend transactions in Bitcoin), then the problem of ABC client ledger consensus is defined as follows.

Definition 5 (ABC client ledger consensus)**.**

- ***Weak agreement****: If some honest client commits tx, every honest client will also commit tx.*
- ***Honest termination****: If a client submits tx to all honest validators and there are no conflicting transactions, tx will be committed by all honest clients. If messages are delivered quickly, tx will be committed quickly.*

It is not hard to see that any traditional (ABC) ledger consensus protocol can be used to solve (ABC) client ledger consensus in our model: the validators just run a traditional (ABC) ledger consensus protocol and send the committed ledgers to the clients. This allows us to construct TrustBoost in a black-box manner. On the other hand, similar to ABC client consensus, ABC client ledger consensus can be solved without any communication among the validators. This observation leads to the protocol TrustBoost-Lite in §4.2, where no CCC is needed.

B Proof of Theorem 1

Proof. Since in our model, the distributed 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) Client consensus:
Only if: Seeking a contradiction, let us assume there is a protocol that solves client consensus in passive mode. Divide all the clients into two sets: X and Y each with at least one honest client. We first construct the following $m+1$ worlds. See Fig. 5.
World $1.i$ **(**$0 \leq i \leq m$**):** In world $1.i$, the first i validators start with value 0 and the rest $m-i$ validators start with value 1. By termination, in all $m+1$ worlds, clients 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 $0 \leq j \leq m-1$ 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 clients in X in this world will be indistinguishable to their views in world $1.(j+1)$ and the view of

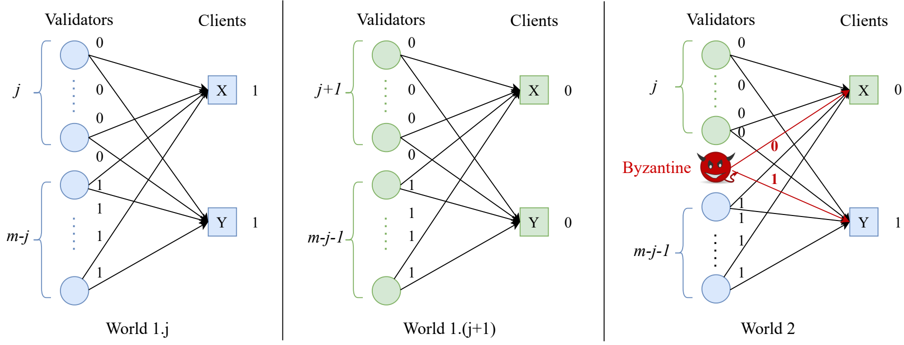

Fig. 5: Different worlds in the proof of Theorem 1.

clients in Y in this world will be indistinguishable to their view in world $1.j$. In world 2, the first j validators start with value 0 and the last $m-j-1$ validators start with value 1. The adversary will use its Byzantine power to corrupt the $(j+1)$-th validator to perform a split-brain attack and make X and Y each believe that they are in their respective worlds. The $(j+1)$-th validator 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 validators, 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 client consensus in the passive mode. The protocol is simple: the validators just send their values to all clients; and the client commits a value if it receives the same value from all m validators; otherwise if the client 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 validators.

(2) ABC client consensus:
Only if: Seeking a contradiction, let us assume there is a protocol that claims to solve ABC client consensus with $f \geq m/3$ Byzantine validators. Divide the m validators into three sets: A, B, and C each with at least one validator and at most f validators. Divide all the clients into two sets: X and Y each with at least one client. 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 validators in A and B start with the value 1. Validators in C are Byzantine but pretend to be honest with initial value 0. Since C has at most f participants, the clients in X must eventually commit a value by honest termination. For validity to hold, all the clients in X will output 1.

World 2: In World 2 validators in B and C start with the value 0. Validators in A are Byzantine but pretend to be honest with initial value 1. Since A has at most f participants, the clients in Y must eventually commit a value by honest termination. For validity to hold, all the clients 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 client consensus in passive mode. The protocol is simple: the validators just send their values to all clients; and the client commits a value if it receives the same value from at least $2f + 1$ validators. It is easy to check that this protocol satisfies agreement, validity and honest termination; Moreover, there is no communication among the validators. □

C Proof of Theorem 2

Proof. Since in our model, the distributed 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) Client consensus:
Only if: Seeking a contradiction, let us assume there is a protocol that claims to solve client consensus with $f \geq m/3$ Byzantine validators. Divide the m validators into three sets: A, B, and C each with at least one validator and at most f validators. Divide all the clients into two sets: X and Y each with at least one client. 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. 6.
World 1: In World 1 validators in A and B start with the value 1. Validators in C and clients in Y have crashed. Since C has at most f participants, the clients in X must eventually commit a value by termination. For validity to hold, all the clients 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 validators in B and C start with the value 0. Validators in A and clients in X have crashed. So all the clients in Y will output 0 by termination and validity. Again, from the perspective of B,

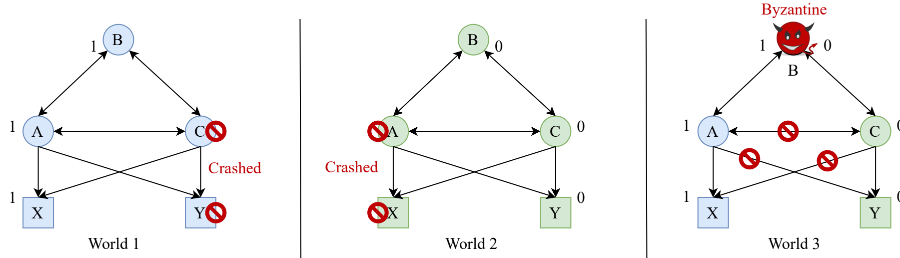

Fig. 6: Different worlds in the proof of Theorem 2.

C and Y, they cannot distinguish between a crashed A vs. an honest A whose messages are delayed.

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 client consensus in active mode. The validators run any partial synchronous consensus protocol and send the committed value to the clients. The client commits a value if it receives the same value from at least $2f + 1$ validators. It is easy to check that this protocol satisfies agreement, validity and termination.

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

If: Same as the algorithm that solves ABC client consensus in passive mode (in the proof of Theorem 1). □