Multi-Level BFT

A scalable consensus mechanism that supports high transaction throughput and allows mass participation

Jieyi Long (jieyi@thetatoken.org)

Introduction

For the blockchain technology to enter the mainstream, many technical challenges need to be tackled. These challenges include supporting high transaction throughput and fast confirmation time while maintaining high level of decentralization.

Recent proposals like the Delegated Proof-of-Stake (DPoS) mechanism offers much higher throughput over Proof-of-Work chains like Bitcoin [1] and Ethereum [2]. However, they typically only permit a limited number of validator nodes [3, 4]. This sacrifices the level of of decentralization and thus the transaction security. Ideally, a Proof-of-Stake (PoS) based consensus mechanism should allow thousands of *independent* nodes, each with similar amount of stake, to participate in consensus process. To compromise such a system, an adversary needs to control a significant amount of independent nodes, which is difficult to achieve.

In this technical report we propose a novel **multi-level BFT consensus mechanism** that allows thousands of consensus participants, and yet can achieve high transaction throughput (1000+ TPS).

The core idea is to have a small set of nodes, which forms the **validator committee**, to produce a chain of blocks as fast as possible using a PBFT-like process [5]. With a sufficient number of validators (e.g. 10 to 20), the committee can produce blocks at a fast speed, and yet it is already difficult enough for the adversary to compromise. Hence, it is reasonable to expect that they will produce a chain of blocks without forks with high probability. Then, all the thousands consensus participants, called the **guardians**, can finalize the chain generated by the validator committee. Here finalization means to convince each honest guardian that more than 2/3 of all the other guardians have the same chain of blocks.

Since there are many more guardians than validators, it could take longer time for the guardians to reach consensus than the validator committee. In order for the guardians to finalize the chain of blocks at the same speed as the validator committee produces new blocks, the guardian nodes can finalize the blocks at a coarser grain. To be more specific, they only need to agree on the hash of the **checkpoint blocks,** i.e. blocks whose heights are multiples of some integer T (e.g. T = 100). This **"leapfrogging" finalization strategy** leverages the immutability characteristic of the blockchain data structure — as long as two guardian nodes agree on the hash of a block, with overwhelming probability, they will have exactly the same copy of the entire blockchain up to that block. Finalizing only the checkpoint blocks gives sufficient time for the thousands of guardians to reach consensus. Hence, with this strategy, the two independent processes, i.e., block production and finalization, can advance with the same pace.

Even though block finalization only performs every T blocks, we yet need to find a way to efficiently to convince every honest guardian that indeed more than 2/3 of all guardians have the same checkpoint hash. A naive all-to-all broadcasting of the checkpoint block hash could work. However, it yields quadratic communication overhead, and so cannot scale to a large number of guardians. Instead we propose a **aggregated signature gossip** scheme which could significantly reduce the messaging complexity. The core idea is rather simple. Each guardian node keeps combining the partially aggregated signatures from all its neighbors, and then gossip out this aggregated signature, along with a compact bitmap which encodes the list of signers. This way the signature share of each node can reach other nodes at exponential speed thanks to the gossip protocol. Within $O(\log n)$ iterations, with high probability, all the honest guardian nodes should have aggregated the signatures from all other honest nodes if there is no network partition.

Validators and Guardians

As mentioned above, the **validator committee** is comprised of a limited set of validator nodes, typically in the range of ten to twenty. They can be selected through an EOS-like election process [3], or a randomized process. The committee may be subject to rotation to improve security, for instance using techniques like the “cryptographic sortition” method from Algorand [6], or the “threshold relay” mechanism from DFINITY [7]. To be eligible to join the validator committee, a node needs to lock up a certain amount of stake for a period of time. The locked stake could be slashed if malicious behavior is detected. The blocks that the committee reaches consensus on are called settled blocks, and the process that they produces a chain of block is called the **block settlement process**.

The **guardian network** is a *super set* of the validator committee, i.e. *a validator is also a guardian*. The guardian network contains a large number of nodes, which could be in the range of thousands. With a certain amount of token lockup for a period of time, any node in the network can instantly become a guardian. The guardians download and examine the chain of blocks generated by the validator committee and try to reach consensus on the the checkpoints. By allowing mass participation, we can greatly enhance transaction security. The blocks that the guardian network has reached consensus on are called finalized blocks, and the process that they finalize the chain of blocks is called the **block finalization process**.

The name **multi-level BFT consensus mechanism** reflects the fact that the validator/guardian division provides multiple levels of security guarantee. The validator committee provides the first level of protection — with 10 to 20 validators, the committee can come to consensus at a fast pace. Yet it is resistant enough to attacks — in fact, it already provides similar level of security compared to the DPoS mechanism. Thus, a transaction can already be considered safe when it has been included in a settled block, especially for low stake transactions, which are the most common transactions. The guardian network forms the second line of defense. With thousands of nodes, it is substantially more difficult for attackers to compromise, and thus provides a much higher level of security. In the unlikely event that the validator committee is fully controlled by attackers, the guardians can re-elect the validators, and the blockchain can restart advancing from the latest block finalized by the guardians. A transaction is considered irreversible when it is included in a finalized block. We believe this mechanism achieves a good balance among transaction throughput, consistency, and level of decentralization, the three corners of the so-called “**impossible triangle**”.

System Model

Before diving into the details of the block settlement and finalization process, we first list our assumptions of the system. For ease of discussion, **without loss of generality, below we assume each node (be it a validator or a guardian) has the same amount of stake**. Extending the algorithms to the general case where different nodes have different amount of stake is straightforward.

Validator committee failure model: There are m validator nodes in total. Most of the time, at most one-third of them are byzantine nodes. They might be fully controlled by attackers, but this happens only rarely. We also assume that between any pair of validators there is a direct message channel (e.g. a direct TCP socket connection).

guardian network failure model: There are n guardian nodes in total. At any moment, at most one-third of them are byzantine nodes. We do not assume a direct message channel between any two guardians. Messages between them might need to be routed through other nodes, some of which could be byzantine nodes.

Timing model: We assume the "weak synchrony" model. To be more specific, the network can be asynchronous, or even partitioned for a bounded period of time. Between the asynchronous periods there are sufficient long time where all message transmissions between two *directly connected* honest nodes arrive within a known time bound Δ. As we will discuss later in the paper, during the asynchronous period, the system simply stops finalizing new blocks. It would never produce conflicting blocks even during asynchrony periods. During the synchronous phase, block production will naturally resumes, and eventual liveness can be achieved.

Attacker model: We assume powerful attackers. They can corrupt a large number of targeted nodes, but no more than one-third of all the guardians simultaneously. They can manipulate the network at a large scale, and can even partition the network for a bounded period of time. Yet they are computationally bounded. For instance, they cannot forge fake signatures, and cannot invert cryptographic hashes.

The Block Settlement Process

Block settlement is the process that the validator committee reaches agreement and produce a chain of blocks for the guardian network to finalize. Various PBFT based blockchain consensus algorithms including the EOS DPoS-BFT, Tendermint, Casper FFG, Hot-Stuff, Algorand, and DFINITY [3, 4, 6-9] can be employed for the validator committee to produce and settle on a chain of blocks. We will omit the details of these algorithm here. But it is worth pointing out that these algorithms guarantees safety when less than one-third of the validators have byzantine failures. Liveness can be achieved during the synchronous periods. When the network is partitioned, the validator committee stops generating new blocks, which is desirable since it avoids split-brain. However, after the network recovers, the validator committee can continue to produce new blocks. With ten to twenty validators, the committee can produce blocks with fast speed, enabling 1000+ transactions per second throughput. In the meanwhile, if these validators are run by different entities, it is already reasonably difficult for an attacker to gain full control of the committee. Thus, the guardians can expect that the validator committee can produce a chain of blocks without forks most of the time.

The Block Finalization Process

As mentioned earlier, to finalize the chain of blocks generated by the validator committee, the guardians only need to reach consensus on the hashes of the **checkpoint blocks**, which are the blocks whose heights are multiple of of some integer T (e.g. $T = 100$).

To see why it is sufficient to finalize just the checkpoint blocks, we note that the transaction execution engine of the blockchain software can be viewed as a “deterministic state machine”, whereas a transaction can be viewed as a deterministic state transfer function. If two nodes run the same state machine, then from an identical initial state, after executing the same sequence of transactions, they will reach an identical end state. Note that this is true even when some of the transactions are invalid, as long as those transactions can be detected by the state machine and skipped. For example, assume there is a transaction tries to spend more tokens than the balance of the source account. The state machine can simply skip this transaction after performing the sanity check. This way the “bad” transactions have no impact on the state.

In the context of blockchain, if all the honest nodes have the same copy of the blockchain, they can be ensured to arrive at the same end state after processing all the blocks in order. But with one caveat — the blockchain might contain huge amount of data. How can two honest nodes compare whether they have the same chain of blocks efficiently?

Here the immutability characteristic of the blockchain data structure becomes highly relevant. Since the header of each block contains the hash of the previous block, as long as two nodes have the same hash of a checkpoint block, with overwhelming probability, they should have an identical chain of blocks from genesis up to that checkpoint. Of course each of the guardian nodes needs to verify the integrity of the blockchain. In particular, they need to verify that the block hash embedded in each block header is actually the hash of the previous block. We note that a node can perform the integrity check on its own, *no communication* with other nodes is required.

Interestingly, the immutability characteristic also enhances the tolerance to network asynchrony or even partition. With network partition, the guardians may not be able to reach consensus on the hash of a checkpoint. However, after the network is recovered, they can move on to vote on the *next* checkpoint. If they can then reach agreement, then all the blocks up to the next checkpoint are finalized, regardless of whether or not they have consensus on the prior checkpoints.

To provide byzantine fault tolerance, an honest node needs to be assured that at least two-third of the guardians has the same checkpoint block hash. Hence it needs to receive signatures for a checkpoint hash from at least two-third of all guardians before the node can mark that checkpoint as finalized. This is to ensure safety, which is similar to the “commit” step in the PBFT protocol.

Since the guardians only need to vote on checkpoint hashes every T blocks, they have more time to reach consensus. A straightforward implementation of checkpoint finalization is thus to follow the PBFT “commit” step where each guardian broadcast its signature to all other guardians. This requires each node to send, receive and process $O(n)$ messages, where each message can be a couple kilobytes long. Even with T blocks time, this approach still cannot scale beyond a couple hundred guardian nodes, unless we select a large T value, which is undesirable since it increases the block finalization latency.

Scale to Thousands of Guardian Nodes

To reduce the communication complexity in order to scale to thousands of guardian nodes, we have designed an **aggregated signature gossip** protocol inspired by the BLS signature aggregation scheme [10, 11] and the gossip protocol. The protocol requires each guardian node to process a much smaller number of messages to reach consensus. Below are the steps of the aggregated signature gossip protocol. It uses the BLS algorithm for signature aggregation.

Protocol: Aggregated Signature Gossip

```
1   $\textit{finalized} \leftarrow \text{false},\ \sigma_i \leftarrow \textbf{SignBLS}(sk_i, height_{cp} \parallel hash_{cp}),\ \hat{c}_i \leftarrow \textbf{InitSignerVector}(i)$
2   for $t = 1$ to $L$ begin
3       send $(\sigma_i, \hat{c}_i)$ to all its neighboring guardians
4       if $\textit{finalized}$ break
5       wait for $(\sigma_j, \hat{c}_j)$ from all neighbors until timeout
6       verify each $(\sigma_j, \hat{c}_j)$, discard if it is invalid
7       aggregate valid signatures $\sigma_i \leftarrow \sigma_i \cdot \prod_j \sigma_j,\ \hat{c}_i \leftarrow \left(\hat{c}_i + \sum_j \hat{c}_j\right) \bmod p$
8       calculate the number of unique signers $s \leftarrow \sum_{u=1}^{n} I(c_{iu} > 0)$
9       if $s \geq 2n/3$ $\textit{finalized} \leftarrow \text{true}$
10  end
```

Algorithm 1*. The Aggregated Signature Gossip Protocol*

The core idea is rather simple. Each guardian node keeps combining the partially aggregated signatures from its neighbors, and then gossip out the newly aggregated signature. This way the signature share of each node can reach other nodes at an exponential speed thanks to the gossip protocol. On the other hand, the signature aggregation keeps the size of the messages small.

In the above diagram, i is the index of the current guardian node. The first line of the protocol uses function **SignBLS**() to generate its initial aggregated signature σ_i. It essentially signs a message which is the concatenation of the height and hash of the checkpoint block using the BLS signature algorithm:

$$h_i \leftarrow H\left(pk_i,\ height_{cp} \parallel hash_{cp}\right) \tag{1}$$

$$\sigma_i \leftarrow (h_i)^{sk_i} \tag{2}$$

In the first formula above, function $H : G \times \{0, 1\}^* \to G$ is a hash function that takes both the public key pk_i and the message as input. This is to prevent the rogue public-key attack [12]. Here G is a multiplicative cyclic group of prime order p with generator g.

The protocol also uses function **InitSignerVector()** to initialize the **signer vector** $\hat{c}_i$, which is a n dimensional integer vector whose u^{th} entry represents how many times the u^{th} guardian has signed the aggregated signature. After initialization, its i^{th} entry is set to 1, and the remaining entries are all set to 0.

After the initialization, the guardian enters a loop. In each iteration, the guardian first sends out its current aggregated signature σ_i and signer vector $\hat{c}_i$ to all its neighbors. Then, if it has not considered the checkpoint as finalized, it waits for the signatures and signer vectors from all its neighbors, or wait until timeout. Upon receiving all the signature and signer vectors, it checks the validity of $(\sigma_j, \hat{c}_j)$ received from neighboring node j using the BLS aggregated signature verification algorithm.

$$h_u \leftarrow H(pk_u,\ height_{cp} \,\|\, hash_{cp}) \tag{3}$$

$$\text{check if } e\left(\sigma_j,\ g\right) \ = \ \prod_{u}^{n} (e\,(h_u,\ pk_u))^{c_{ju}} \tag{4}$$

where c_{ju} is the u^{th} entry of vector $\hat{c}_j$, and $e : G \times G \to G_T$ is a bilinear mapping function from $G \times G$ to G_T, another multiplicative cyclic group also of prime order p. To guard against byzantine nodes, all the invalid signatures and their associated signer vectors are **discarded** for the next aggregation step. It is worth pointing out that besides $height_{cp}$, $hash_{cp}$, the above check also requires the public key pk_u of the relevant guardians as input. All these information should be available locally, since when a guardian locked up its stakes, its public key should have been attached to the stake locking transaction which has already been written into the blockchain. Hence, no communication with other nodes is necessary to retrieve these inputs.

The aggregation step aggregates the BLS signature σ_j, and updates the signer vector $\hat{c}_j$. Note that for the vector update, we take $\mathrm{mod}\ p$ for each entry. We can do this because $e\,(h_u,\ pk_u) \in G_T$, which is a multiplicative cyclic group of prime order p. This guarantees that vector $\hat{c}_j$ can always be represented with a bounded number of bits.

$$\sigma_i \leftarrow \sigma_i \cdot \prod_{j} \sigma_j\,,\ \hat{c}_i \leftarrow \left(\hat{c}_i + \sum_{j} \hat{c}_j\right) mod\, p \tag{5}$$

The algorithm then calculate the number of unique signers of the aggregated signature for node i.

$$s \leftarrow \sum_{u=1}^{n} I\,(c_{iu} > 0) \tag{6}$$

Here c_{iu} is the u^{th} entry of vector $\hat{c}_i$. Function I: $\{\text{true, false}\} \to \{1,\ 0\}$ maps a true condition to 1, and false to 0. Condition $c_{iu} > 0$ indicates that node u has signed the aggregated signature that node i currently possesses at least once. Hence the summation counts how many unique signers have contributed to the aggregated signature. If the signature is signed by more than 2/3 of all the guardians, the guardian can consider the checkpoint as finalized.

If the checkpoint is finalized, the aggregated signature will be gossipped out in the next iteration. After this gossip, within $O(log(n))$ iterations all the honest guardians will have an aggregated signature that is signed by more than two-third of all the guardians if the network is not partitioned. The for-loop has L iterations, L should be in the order of $O(log(n))$ to allow the signature to propagate through the network.

Analysis

Correctness of the Aggregated Signature Gossip Protocol

To prove the correctness of the aggregated signature gossip protocol, we need to prove two claims. First, if an aggregated signature is correctly formed by honest nodes according to Algorithm 1, it can pass the check given by Formula (4). Second, the aggregated signature is secure against forgery. Stated more formally, forging a fake aggregated signature in the context of Algorithm 1 means to find $\sigma \in G$ and integers $c_1, c_2, \ldots c_n$ which satisfy the equation below

$$e(\sigma, g) = \prod_{u=1}^{n} (e(h_u, pk_u))^{c_u}$$

for randomly chosen $pk_1 = g^{sk_1}, \ldots, pk_n = g^{sk_n} \in G$, and random message hashes $h_1 \ldots, h_n \in G$. We can show that this is is as hard as the Computational Diffie-Hellman (CDH) problem [10, 11]. We will prove both claims in the appendix.

Messaging Complexity

The aggregated signature gossip protocol runs for L iterations, which is in the order of $O(log(n))$. In each iteration, the guardian needs to send message $(\sigma_i, \hat{c}_i)$ to all its neighboring guardians. Depending on the network topology, typically it is reasonable to assume that for an average node, the number of neighboring nodes is a constant (i.e. the number of neighbors does not grow as the total number of nodes grows). Hence the number of message a node needs to send/receive to finalize a checkpoint is in the order of $O(log(n))$, which is much better than the $O(n)$ complexity in the naive all-to-all signature broadcasting implementation. We do acknowledge that each message between two neighboring guardians contains an n dimensional signer vector c_i, where each entry of c_i is an integer smaller than prime p. However, as will be shown in the appendix, this vector can be represented rather compactly since most of its entries are small integers ($\ll p$).

To get a more concrete idea of the messaging complexity, let us work out an example. Assume that we pick a 170-bit long prime number p for the BLS signature, which can provide security comparable to that of a 1024-bit RSA signature. And there are 1000 guardian nodes in total. Under this setting, $\hat{c}_i$ can be naively represented with about twenty kilobytes without any compression. Since most of the entries of $\hat{c}_i$ are far smaller than p, $\hat{c}_i$ can be compressed very effectively to a couple kilobytes long. Plus the aggregated signature, the size of each message is typically in the kilobytes range. Moreover, if we assume on average a guardian connects to 20 other guardians, then L can be as small as 5 (more than twice of $log_{20}(1000) = 2.3$). This means finalizing one checkpoint just requires a guardian to send/receive around 100 messages to/from its neighbors, each about a couple kilobytes long.

We will dive deeper into the messaging complexity analysis in the appendix. There we will give a theoretical model for the maximum message size, and present some simulation results. In our simulations, even with a couple thousands of guardian nodes, the biggest entry of $\hat{c}_i$ can be represented with just a single byte when every honest node has received an aggregated signature with shares from more than 2/3 of all nodes. Thus, each message is at most a couple kilobytes long. This renders the aggregated signature gossip protocol very practical to implement and can easily scale to thousands of guardian nodes.

Finalization Safety and Liveness

Safety: Safety of the block finalization is easy to prove. Under the 2/3 supermajority honesty assumption, If two checkpoint hashes for the same height both get signatures from at least 1/3 of all guardians, at least one honest guardian has to sign different hashes for the same checkpoint, which is not possible.

Liveness: Without network partition, as long as L is large enough, it is highly likely that after $O(log(n))$ iteration, all the honest nodes will see an aggregated signature that combines the signatures of all honest signers. This is similar to how the gossip protocol can robustly spread a message throughout the network in $O(log(n))$ time, even with up to 1/3 byzantine nodes. When there is network partition, consensus for a checkpoint may not be able to reach. However, after the network partition is over, the guardian network can proceed to finalize the next checkpoint block. If consensus can then be reached, all the blocks up to the next checkpoint are considered finalized. Hence the finalization process will progress eventually.

Reward and Penalty for Validators and Guardians

The token reward and penalty structure is essential to encourage more nodes to participate in the consensus process, and honestly follow the protocol.

Both the validators and guardians can obtain token reward. Each block includes a special **Coinbase transaction** that deposits newly minted tokens to the validator and guardian addresses. All the validators can get a share of tokens for each block. For guardians, rewarding every guardian for each block might not be practical since their number is large. Instead, we propose the following algorithm to randomly pick a limited number of guardians as the reward recipient for each block.

Denote the height of the newly proposed block by l, and cp is the most recent finalized checkpoint. The proposer should have received the aggregated signature σ_{cp} and corresponding signer vector $\hat{c}_{cp}$ for checkpoint cp. After validating $(\sigma_{cp}, \hat{c}_{cp})$, the proposer can check the following condition for each guardian whose corresponding entry in vector $\hat{c}_{cp}$ is not zero (i.e. that guardian signed the checkpoint)

$$H(pk_i, \sigma_{cp} \| B_{l-1}) \leq \tau \tag{7}$$

where B_{l-1} is the hash of the block with height $l-1$, and $H : G \times \{0, 1\}^* \to G$ is the same hash function used in the BLS signature algorithm. If the inequality holds, the proposer adds the guardian with public key pk_i to the Coinbase transaction recipient list. Threshold τ is chosen properly such that only a small number of guardians are included. The proposer should also **attach** $(\sigma_{cp}, \hat{c}_{cp})$ to the Coinbase transaction as the proof for the reward.

We also enforces token penalty if any malicious behavior is detected. In particular, in the block settlement process, if a block proposer proposes conflicting blocks for the same height, or if a validator votes for conflicting blocks for the same height, it should be penalized. Earlier we mentioned that to become either a validator or a guardian, a node needs to lock up a certain amount of tokens for a period of time. The penalty will be deducted from their locked tokens. The node that detects the malicious behavior can submit a special Slash transaction to the blockchain with the proof of the malicious behavior (e.g. signatures for conflicting blocks) attached. The penalty tokens will be pulled from the malicious node and awarded to the node that submitted the first Slash transaction.

In the unlikely event that more than one-third of the validators are compromised, the malicious validators can attempt to perform the double spending attack by forking the blockchain from a block that is settled but not yet finalized. However, this is detectable by the guardian network, since forking will generate multiple blocks with the same height, but each signed by more than two-third of the validators. In this case, the entire validator committee will be re-elected. After the validator committee is reinstated, the old validators that conducted double signing will be penalized, and the blockchain can continue to advance from the most recent finalized checkpoint.

Remarks

We would like to remark that the blockchain data structure only needs to be slightly extended to support the multi-level BFT mechanism — as discussed in the previous section, the Coinbase transaction carries the aggregated signature of the guardian nodes. By processing the Coinbase transactions, a node is aware of to what height the blockchain has been finalized. No extra change to the blockchain structure is required.

Furthermore, we would like to point out that the multi-level BFT frameworks can be applied for both **permissioned** blockchains and **permissionless** blockchains. In the permissioned chain setting, each node (be it a validator or guardian) may have the same voting power, and block finalization requires signatures from more than 2/3 of all guardian nodes. Also as mentioned earlier, techniques like “cryptographic sortion” and “threshold relay” from Algorand and DFINITY can be borrowed for electing and rotating the validator committee. In the permissionless setting, any node can become a guardian by staking a certain amount of tokens above the required minimum. Voting power of a node is proportional to the amount of tokens it staked.

We also note that there is a line of works from Bitcoin-NG [13] to recent proposals like ByzCoin [14], IOST [15] that employ a single leader to produce blocks in order to achieve high throughput, with a number of nodes validate the blocks afterwards. While bearing some similarities, in our approach we use a validator committee instead of a single leader, which significantly enhances the security while still can achieve high throughput. Our guardian network leverages a BLS based signature aggregation scheme to reach agreement. Compared to the Schnorr signature based CoSi scheme [16] employed by ByzCoin, our approach might be simpler to implement thanks to the non-interactive nature of the BLS signature aggregation. Also in our approach every node aggregates signatures from its neighbors. It could be more robust in the presence of byzantine nodes compared to CoSi, which uses multicast trees to aggregate signatures.

References

[1] *Nakamoto*. Bitcoin: A Peer-to-Peer Electronic Cash System

[2] *Buterin et al*. A Next-Generation Smart Contract and Decentralized Application Platform

[3] *Larimer et al*. The EOS Technical whitepaper

[4] *Buchman et al*. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains

[5] *Castro et al*. Practical Byzantine Fault Tolerance

[6] *Gilad et al*. Algorand: Scaling Byzantine Agreements for Cryptocurrencies

[7] *Hanke et al*. DFINITY Technology Overview Series Consensus System

[8] *Yin et al*. HotStuff: BFT Consensus in the Lens of Blockchain

[9] *Buterin et al*. Casper the Friendly Finality Gadget

[10] *Boneh et al*. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps

[11] *Boneh et al*. A Survey of Two Signature Aggregation Techniques

[12] *Boneh et al*. BLS Multi-Signatures With Public-Key Aggregation

[13] *Eyal et al*. Bitcoin-NG: A Scalable Blockchain Protocol

[14] *Kogias et al*. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing

[15] *IOST team*. The IOS Token Whitepaper

[16] *Syta et al*. Keeping Authorities “Honest or Bust” with Decentralized Witness Cosigning

Appendix

Aggregated Signature Gossip Correctness

In this section, we will prove two claims. First, if an aggregated signature is correctly formed according to Algorithm 1, it can pass the check given by Formula (4). Second, the aggregated signature is secure against forgery.

To prove **the first claim**, we note that both group G and G_T are cyclic groups. A cyclic group is always an Abelian group, and hence we can switch the order of elements in multiplications. Furthermore, since group G_T is has a prime order p, given a group element $v \in G_T$, for any integer c, $v^{c} = v^{c \bmod p}$. With these notes, it is straightforward to prove the first claim using the definition of bilinear mapping. In the prove below, Nb denotes the set of neighboring nodes of node j, σ_{jk} is the partially aggregated signature collected from neighbor node k in the current iteration, and σ_u is the signature of node u given by Formula (2).

$$e\left(\sigma_j, g\right) = e\left(\prod_{k\in Nb} \sigma_{jk}, g\right) = \prod_{k\in Nb} e\left(\sigma_{jk}, g\right) = \prod_{u}^{n} (e(\sigma_u, g))^{c_{ju}}$$

$$= \prod_{u}^{n}\left(e\left(h_u{}^{sk_u},\, g\right)\right)^{c_{ju}} = \prod_{u}^{n}\left(e\left(h_u,\, g^{sk_u}\right)\right)^{c_{ju}}$$
$$= \prod_{u}^{n}(e(h_u,\, pk_u))^{c_{ju}}$$

Now let us prove **the second claim**. Stated more formally, it asserts that for randomly chosen $pk_1 = g^{sk_1}$, …, $pk_n = g^{sk_n} \in G$, and random message hashes $h_1 \ldots,\ h_n \in G$, finding $\sigma \in G$ and integers $c_1,\ c_2,\ \ldots\ c_n$ which satisfy the equation below is as hard as the Computational Diffie-Hellman (CDH) problem [10, 11].

$$e(\sigma,\, g) = \prod_{u=1}^{n} (e(h_u,\, pk_u))^{c_u}$$

To prove this claim, let us first look at a special case where $n = 1$, i.e., there is only one guardian node. In this case, for the adversary, forging a single guardian's aggregated signature means to find $\sigma \in G$ and an integer c that satisfy

$$e(\sigma,\, g) = (e(h,\, pk))^{c} \tag{8}$$

Note that

$$(e(h,\, pk))^{c} = e(h^{c},\, pk) = e(h^{c},\, g^{sk}) = e\left(h^{sk\cdot c},\, g\right)$$

We thus have

$$\sigma = \left(h^{sk}\right)^{c}$$

Next, using the Extended Euclidean algorithm, in polynomial time we can find an integer d such that $c \cdot d \equiv 1 \bmod p$, where p is the order of G, which is a prime number. Thus

$$\sigma^{d} = \left(h^{sk}\right)^{c\cdot d} = \left(h^{sk}\right)^{c\cdot d \bmod p} = h^{sk}$$

which means that the adversary can compute h^{sk} from g, g^{sk}, and h in polynomial time using σ and c. This indicates that finding $\sigma \in G$ and integer c satisfying equation (8) is as hard as the CDH problem.

When there are $n > 1$ guardian nodes, signature forgery means that the adversary can find $\sigma \in G$, and integers $c_1,\ c_2,$ ... c_n that satisfies the equation below

$$e(\sigma,\, g) = \prod_{u=1}^{n} (e(h_u,\, pk_u))^{c_u} \tag{9}$$

Again, we show this is equivalent to solving the CDH problem. To see why, for randomly chosen g, g^{sk_1} and h_1, the adversary can randomly generate $n - 1$ key pairs $(sk_2,\, pk_2),\ \ldots,\ (sk_n,\, pk_n)$ where $pk_u = g^{sk_u}$ for $u = 2, 3, \ldots, n$. He then plugs these values into Equation (9), and solve for $\sigma \in G$ and integers $c_1,\ c_2,\ \ldots\ c_n$. Note that

$$e(\sigma,\ g) = \prod_{u=1}^{n} (e(h_u,\ pk_u))^{c_u} = \prod_{u=1}^{n} e\left(\left(h_u{}^{sk_u}\right)^{c_u},\ g\right) = e\left(\ \prod_{u=1}^{n} \left(h_u{}^{sk_u}\right)^{c_u},\ g\right)$$

This means

$$\sigma = \prod_{u=1}^{n} \left(h_u{}^{sk_u}\right)^{c_u} \tag{10}$$

Since the adversary knows $sk_2, \ldots, sk_n$, he can plug these private keys into Equation (10), which then gives

$$\left(h_1{}^{sk_1}\right)^{c_1} = \sigma \cdot \left(\prod_{u=2}^{n} \left(h_u{}^{sk_u}\right)^{c_u}\right)^{-1} = \sigma \cdot \prod_{u=2}^{n} h_u{}^{-sk_u \cdot c_u}$$

Similar to the reasoning for $n = 1$, in polynomial time we can find an integer d_1 such that $c_1 \cdot d_1 \equiv 1 \bmod p$. Thus we have

$$\left(\sigma \cdot \prod_{u=2}^{n} h_u{}^{-sk_u \cdot c_u}\right)^{d_1} = \left(h_1{}^{sk_1}\right)^{c_1 \cdot d_1} = \left(h_1{}^{sk_1}\right)^{c_1 \cdot d_1 \bmod p} = h_1{}^{sk_1}$$

Hence, after solving Equation (9), the adversary can efficiently compute $h_1{}^{sk_1}$ for randomly chosen g , g^{sk_1} and h_1 . This implies that solving Equation (9) is as hard as solving the CDH problem.

In summary, we can conclude that our proposed aggregated signature gossip protocol is correct and secure.

Messaging Complexity Analysis

In this section, we will introduce a theoretical model to analyze the messaging complexity of the aggregated signature gossip protocol. We will also present some simulation results which demonstrate the practicality of the protocol.

To model the message size, we would need to introduce the following notations. First, we define vector $\hat{x}_k^t$, which is composed of the k^{th} element of the **signer vector** $\hat{c}_j$ that each guardian node possesses in iteration t

$$\hat{x}_k^t = \left(c_{1k}^t,\ c_{2k}^t,\ \ldots,\ c_{nk}^t\right)^T$$

As a special case, initially when $t = 0$, $\hat{x}_k^0 = (0,\ 0,\ \ldots,\ 1,\ \ldots,\ 0)^T$ where only the k^{th} element is 1. We also define matrix Δ which represents the connectivity among the guardian nodes.

$$\Omega = \left(\delta_{ij}\right)_{n \times n}$$

where $\delta_{ij} = 1$ only when $i = j$, or node i and node j are neighbors (i.e., there is a direct TCP socket connection between node i and node j). Otherwise $\delta_{ij} = 0$.

It can be shown that in Algorithm 1, if we do not break the loop (i.e. remove line 4), the following equation holds

$$\hat{x}_k^{t+1} = \Omega \cdot \hat{x}_k^t$$

Thus we have

$$\hat{x}_k^t = \Omega^t \cdot \hat{x}_k^0$$

We know that if $\det(\mathbf{\Omega}) \neq 0$, matrix $\mathbf{\Omega}$ always has an eigendecomposition

$$\Omega = V\ (\lambda_i)_{n \times n} V^{-1}$$

where $(\lambda_i)_{n \times n}$ is an $n \times n$ matrix whose diagonal are the eigenvalues, and the non-diagonal elements are all zero. Hence we have

$$\hat{x}_k^t = V\ (\lambda_i)_{n \times n}^t V^{-1} \cdot \hat{x}_k^0$$

This indicates that the biggest entry of $\hat{x}_k^t$, and hence the signer vectors, is in the order of λ_{max}^t , where λ_{max} is the biggest eigenvalue of the connectivity matrix. Although in the first glance, the entries increase exponentially, however, we note that Algorithm 1 runs only for $O(\log n)$ iterations. Thus it can be shown that the biggest entries is in the order of $O(n)$, which can be represented with $O(\log n)$ bits.

Notice that the above analysis assumes line 4 in Algorithm 1 is removed. With line 4, a guardian node stops aggregating signature and signer vectors after its current signature contains shares from more than 2/3 of all guardians. This can further reduce the bits required to represent the signer vectors.

We have conducted various experiments to simulate the message sizes. To generate the connectivity matrix, we note that typically when a guardian node joins the network, it first connects to a few seed nodes. These seed nodes then send a list of candidate nodes to the new node, and the new node randomly selects a subset to connect to. This process continues until the maximum neighbor count is reached. In our experiments, we generated random connectivity matrices following this process.

We have simulated guardian networks with 1000, 2000, and 3000 nodes. In each case, the maximum neighbor count is set to 30. We have also simulated with byzantine nodes whose percentage ranges from 0% to 30%. These byzantine nodes either send their neighbors fake signatures, or send out no signature. In all of our simulations, upon convergence (i.e. each honest guardian node gets signature shares from more than 2/3 of all nodes), the biggest entry is less than 256, and thus each entry in the signer vector can be represented by one single byte. In many cases, the biggest entry is less than 64, and thus can be represented with merely 6 bits. In any case, even with a couple thousand nodes, the size of each messages is just a couple kilobytes. And each honest guardian nodes only need to send/receive less than 200 messages to reach consensus for a checkpoint block. This indicates our proposed aggregated signature gossip protocol is very practical and robust to a high number of nodes with byzantine faults.