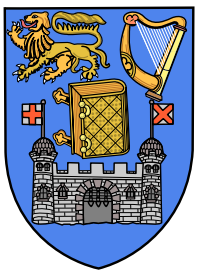
University of Dublin



TRINITY COLLEGE

School of Computer Science and Statistics

Optimized Communication in Sharded Blockchain Networks

James Rowland

BA (Mod) in Science in Computer Science

Supervisor : Dr. Hitesh Tewari

April 2022

A Report submitted in partial fulfilment of the requirements for the degree of BA(Mod) in Science in Computer Science

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

James Rowland

Date

Permission to lend

I agree that the Library and other agents of the College may lend or copy this report upon request.

James Rowland _____ Date _____

Abstract

Sharding has established itself as a vital technology for significantly improving all three aspects of the blockchain trilemma (security, decentralization and scalability). In a sharded blockchain the network of nodes responsible for maintaining the blockchain is divided into separate groups called shards so that each shard can process transactions in parallel. Nodes within each shard have to communicate between themselves to synchronously take action. Here we explore and improve the communication process that happens between nodes in a shard. Specifically we analyze how the Boneh-Lynn-Shacham (BLS) multi-signature scheme [4] is used to increase communication scalability between nodes in a shard. The security techniques required to safely use BLS multi-signatures are explored. We then propose security techniques which offer improved efficiency over existing techniques. Finally, a realistic simulation of the communication process between nodes in a shard using BLS multi-signatures is implemented. This simulation is used to analyze and evaluate the performance of the existing and proposed security techniques.

Acknowledgements

I would like to thank my supervisor Professor Hitesh Tewari for his advice, guidance, insights and support that made this project possible. Your wide range of expertise was invaluable in exploring new technology solutions and your feedback pushed my project to a level way beyond where I could reach by myself.

Thank you to Terry and Margaret Rowland for all the support during a very tough period. Thank you to Claire O'Callaghan for the reassurance, love and help.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Decentralized blockchains have proven to be highly adaptable across many important types of human interaction such as finance, storage, and gaming. A result of the growing list of use cases for decentralized blockchains is the apparent necessity for efficient and scalable transaction processing. Scalable transaction processing was not emphasized in the design of early decentralized blockchains such as Bitcoin where security and decentralization were prioritized. As a result new approaches have been applied to blockchain network design with the goal of addressing scalability and efficiency issues.

Sharding is considered to be a complex solution to the scaling problem. Nodes are grouped into subsets called shards and each shard is assigned a separate and distinct subset of transactions to process. Therefore, the system is able to process transactions in parallel which increases the scalability of the network however it also introduces new pitfalls we need to be aware of. One such pitfall is efficient communication between nodes in a shard.

1.2 My Contribution

Each shard in a sharded blockchain contains a group of nodes called a *committee*. The committee's nodes communicate with one another to reach consensus on decisions such as which block of transactions to include in the blockchain. For the committee to reach consensus the Byzantine Fault Tolerance (BFT) state machine algorithm is run in each node in the committee. Practical Byzantine Fault Tolerance (PBFT) is the name of the BFT variant used in sharded blockchains to reach consensus given adversarial conditions. In each committee a single node is appointed as the primary replica and the remaining nodes are appointed as secondary replicas. The primary replica, often referred to as the leader, is responsible for aggregating and relaying information to and from secondary replicas. Secondary replicas are responsible for communicating whether they want to proceed to the next state in the PBFT state machine. PBFT has five different states that the replicas can be in. These states ensure liveliness, consistency and provide means of eventually reaching consensus. During the communication process replicas broadcast their signature's to every other replica if they approve of the current PBFT state. When a replica collects a large majority of signatures it assumes it is clear to proceed to the next PBFT state even if there are adversaries in our network. This project explores the use of BLS multi-signatures during this communication process in order to increase scalability.

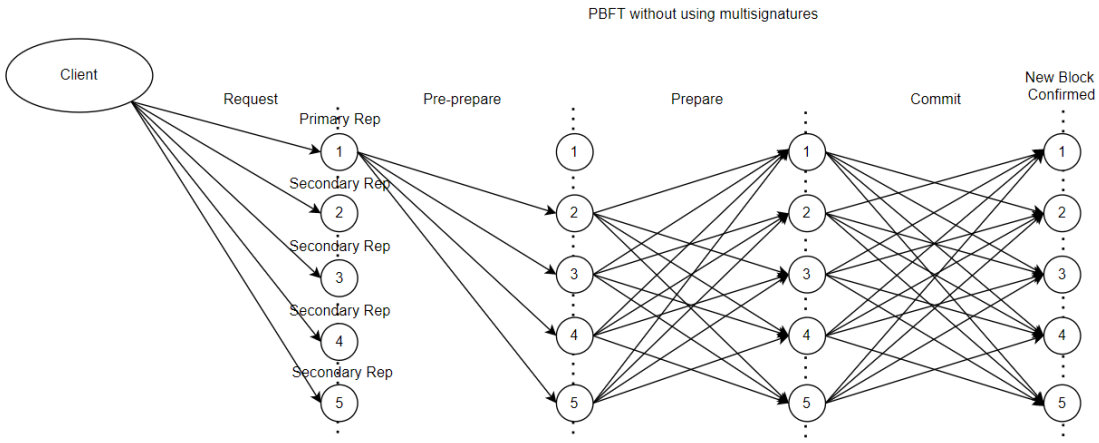


Figure 1.1: PBFT Communication Without Multi-Signatures

Figure 1.1 displays the communication between nodes during a full round of consensus without multi-signature. Communication interaction would act as such :

1. In the pre-prepare state the primary replica broadcasts an announce message containing a reference to the block that the primary replica wants to add to the blockchain. When a secondary replica receives an announce message it enters the prepare state.

2. In the prepare state the secondary replicas broadcast their signature if they are prepared to add the block to the blockchain. All replicas gather and validate these signatures. When enough signatures are gathered by a replica it will move to the commit state.

3. In the commit state the secondary replicas broadcast their signature if they are committing the block to the blockchain. All replicas gather and validate these signatures. Once the voting criteria is met the committee of replicas can assume they collectively agree on the block and include the block in the blockchain.

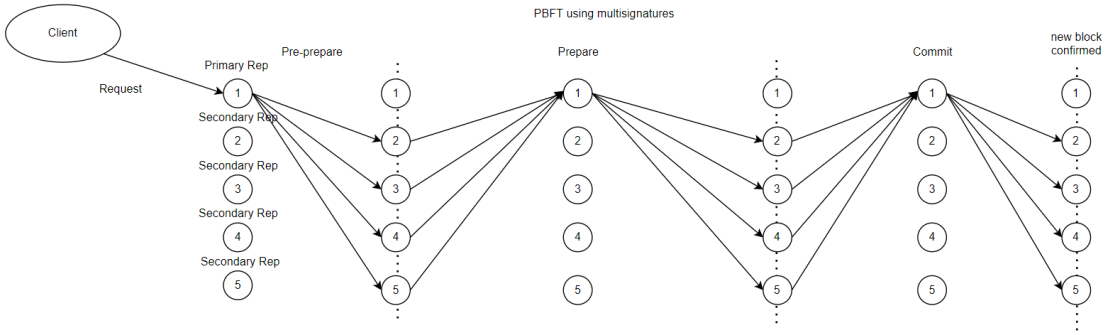


Figure 1.2: PBFT Communication With Multi-Signatures

Figure 1.1 displays the communication process between nodes without multi-signature use. If a secondary replica approves of a PBFT state it has to broadcast a message containing its signature to every other replica. With n replicas in the committee this requires broadcast of n^2 messages just for one single PBFT state. Through the use of multi-signatures we only

require $2 \cdot n$ messages. This reduction comes from approving secondary replicas sending a message containing their signature to the primary replica only as seen in Figure 1.2. The primary replica will aggregate all signature together into a single signature called a multi-signature. The primary replica then broadcasts the multi-signature to all secondary replicas. Secondary replicas can then validate the multi-signature and proceed. In this project we explore how integrating multi-signatures into the communication process can increase communication scalability. To do this, the main tasks are:

- Identify what BLS multi-signatures are and how they work.
- Determine the algorithms and operations involved in multi-signature construction and verification.
- Investigate attacks on the multi-signatures.
- Compare defenses against attacks.
- Design my own defense mechanism.
- Simulate communication using multi-signatures with each defense mechanism.
- Analyze simulation results.
- Evaluate communication efficiency, scalability, security and implement-ability using each different existing and improved defense.
- Conclude if our proposed improved defense mechanisms offered any improvement over existing defenses.

It is important to note that all papers and studies regarding communication in sharded blockchains do not use simulations to analyze the performance of multi-signature's and each different defense mechanism. We provide a realistic and extensible test framework which by itself is a very important and valuable contribution to the space of sharded blockchains. We also propose improved defense mechanism designs that implement modern blockchain infrastructure to try increase the efficiency of communication protocols using multi-signatures which is another huge contribution.

Report Structure

The remainder of the paper is structured as follows : **Chapter 2** Presents a fundamental overview of sharded blockchains.**Chapter 3** Explores PBFT, BLS multi-signatures & defense mechanisms. **Chapter 4** Presents our new improved multi-signature defense mechanisms. **Chapter 5** Introduces a simulation which simulates communication using multi-signatures with each known defense mechanism and our own improved defense mechanism .**Chapter 6** Analyzes and evaluates simulation results and compares existing defenses against our own designed defenses. **Chapter 7** Concludes our paper.

Chapter 2

Sharding Overview

2.1 Sharding Introduction

Sharding is one of the most promising technologies that can significantly improve the scalability of blockchains. Sharding involves partitioning the network of nodes responsible for maintaining the blockchain into subgroups called shards. Each shard is responsible for maintaining a separate portion of the blockchain, leading to a much more scalable system because nodes are only concerned about a subset of the blockchain rather than the whole system. This concept of partitioning work between subgroups is commonly known as *Divide and Conquer*. We group our nodes into subsets that process transactions for a distinct portion of the blockchain which results in parallelizing the transaction processing within the blockchain. Through sharding we can scale out our system as we can increase the number of shards when we have more transactions to process.

A shard will contain a self-governing, self-sufficient group of nodes called a committee that commits blocks to the blockchain. Committees are verifiably secure against Denial-of-Service attacks and hijackings due to the committee's adherence to the PBFT state machine protocol. An instance of the PBFT protocol is replicated in all nodes in the committee which is why nodes are called replicas. PBFT involves communication rounds where replicas

broadcast and collect replies from all other replicas. At the end of a communication round a replica must gather a supermajority by receiving accepting replies from other replicas allowing it to proceed to the next state in the PBFT state protocol. Previously communication rounds in PBFT involved $(n \cdot n)$ number of messages as nodes would broadcast their message to every other node. This messaging process does not scale well as committee size increases. Instead, integrating multi-signatures into the communication process results in a significantly more scalable messaging process in communication rounds which only requires $(2 \cdot n)$ messages.

2.2 Transaction Validation

In sharded blockchains both *Proof-of-X* (PoX) and PBFT work together to validate transactions and to add blocks of transactions to the blockchain. PoX is typically used for transaction validation and PBFT is used for committee consensus. The purpose of committee consensus is to securely and fairly vote on which valid group of processed transactions (called blocks) the committee want to include in the blockchain next. Since there are many transaction validators trying to propose blocks, there needs to be a step where the committee votes on which block to include in the blockchain. PBFT is used for this agreement on which block to include in the blockchain which is called reaching consensus. PoX protocols are typically used to verify transactions and usually require effort (Proof-of-Work in Bitcoin [20]) or resources (Proof-of-Stake in Ethereum [1]) to prove the validity of the processed transactions. In Proof-of-Work (PoW), nodes guess a correct solution to a cryptographic problem. The first node to successfully solve the problem can propose a new block of transactions. In PoW, nodes are more likely to successful if they have more computational power. In Proof-of-Stake (PoS), nodes invest a number of tokens into the blockchain and in return receive back votes. These votes can then be used in proposing a new block of transaction. In PoS, nodes are more likely to propose new blocks if they have more votes.

2.3 Committee Validation

After blocks are validated most sharded blockchains use the PBFT state machine protocol as its intra-shard consensus protocol to reach consensus on which block to include in the blockchain. A replica is a node that has a replication of PBFT state machine protocol which is running with other committee replicas. PBFT operates with at least $3f + 1$ replicas in the committee where f is the maximum number of byzantine nodes in the committee. Requiring at least $3f + 1$ replicas in the committee ensures safe consensus can occur assuming there are byzantine nodes in our committee. Byzantine nodes are defined as replicas (or nodes) which cause failure due to software issues, hardware issues, or malicious intention.

PBFT is classified as a leader-based PBFT protocol where the primary replica leads and orchestrates the secondary replicas. The primary replica is responsible for initiating block proposal and relaying information to and from the secondary replicas. Secondary replicas are responsible for communicating whether they approve of the current PBFT state that their replication of the PBFT state machine is in.

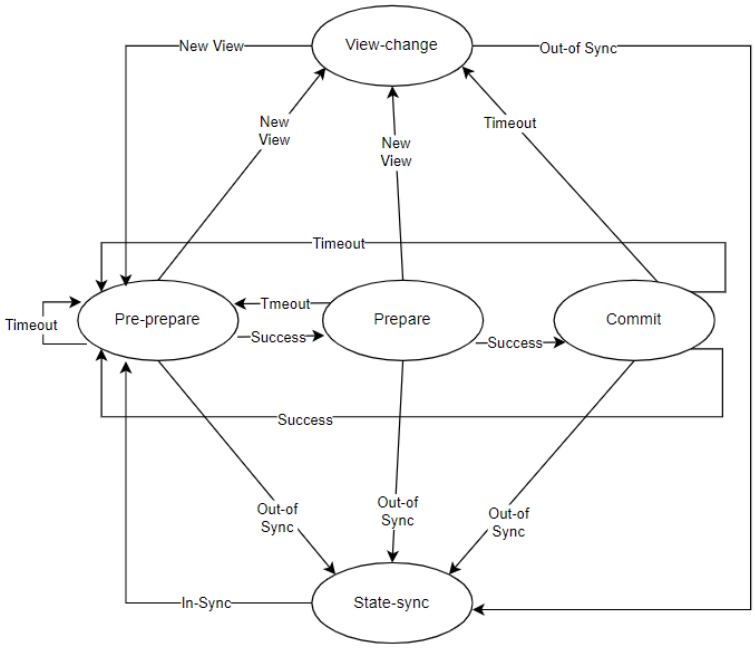


Figure 2.1: Simple PBFT state diagram

To reach consensus all replicas run an agreement process involving three PBFT states (pre-prepare->prepare->commit) as shown in Figure 2.1 which lets the committee of replicas agree on the block to include in the blockchain. PBFT consensus accounts for all error types by including specialized states in the PBFT state machine. If errors occur the replicas either re-enter their current state or enter an error handling state. If consensus is being prevented by a faulty or malicious primary replica, replicas running the PBFT state machine will enter a *view-change* state which elects a new primary replica / leader.

2.4 Shard Scalability

Scaling both the size of the shard and the number of shards is a difficult problem. Although PBFT consensus is an integral part of sharded blockchains there exists a proportional relationship between committee size and communication overhead. When more replicas are added to the committee more communication is required to reach consensus. PBFT con-

sensus was originally designed to only run among a small number of nodes [19]. This limits the number of replicas per shard and the number of shards we can have. In reality, decentralized sharded blockchains should be able to have any number of shards with any number of replicas per shard.

PBFT adopts the use of quorums which are intersections of subsets of the committee. From the use of quorums we can increase the efficiency of our shard since we only require a subset of replicas to communicate while still maintaining a high level of the safety. Epoch randomness has also been a crucial piece of shard scalability. Epoch randomness ensures the best guarantee for a non-malicious majority to exist in each committee and that malicious nodes will be best spread over all committee's. Randomness is used while assigning nodes to shards so that a group of malicious nodes cannot work together to take control of a shard. From using effective randomness in Ethereum's sharded Casper Blockchain it achieved a probability of malicious network takeover of 1 in 100 Trillion. Sharding based blockchains operate in time-slots called epochs where each epoch is the time taken to mine a block of transactions. The number of committees we can make per epoch can scale linearly with the number of nodes in the network and we can change the size of the committee between epochs as well. Therefore, as the network of nodes grows the transaction throughput increases without any additional latency.

As stated by the Ethereum Foundation [10] a key part of sharded blockchain scalability is now going to be communication efficiency. It is very important for future blockchains to have efficient communication protocols. Efficient communication protocols require effective information encryption, decryption, transmission and verification. This explains the importance of the new improved communication protocols and simulation frameworks outlined in this report.

Chapter 3

BFT, Multi-Signatures & Defense Mechanisms

3.1 Byzantine Fault Tolerance

Due to the fact that malicious and honest nodes can join the decentralized blockchain network at any stage, we require strong failure models that account for malicious nodes in the blockchain network. BFT is a state machine replication algorithm that provides the strong operating and failure models we need. State machine replication means the state machine algorithm is replicated in every node called a *replica* in the network. BFT is a voting and agreement model that accounts for malicious attacks, operator mistakes, and software errors. BFT ensures message delays are bounded eventually to prevent Denial-of-Service attacks [5]. Therefore, BFT provides the atomicity, consistency, and isolation qualities that a committee in the blockchain network needs in order to operate given malicious attacking nodes in the network.

3.1.1 Practical Byzantine Fault Tolerance

PBFT which is a specialized variant of BFT has been heavily adopted by popular decentralized blockchains such as Omniledger [9] and ByzCoin [12]. In these blockchains there

are multiple committees where each committee contains a group of replicas. Each replica runs its own isolated replication of the PBFT state machine algorithm. As replicas move from PBFT state to PBFT state they message other replicas if they approve of the current PBFT state. Replicas gather messages from other replicas and decide what state to move to next depending on the messages they gathered. This process of sending and receiving messages, evaluating the received messages and moving to the next PBFT state depending on the evaluation result is how a group of individual replicas reach consensus on which blocks to include in the blockchain.

In PBFT a committee will elect one node to be the primary replica. The primary replica is often referred to as the *leader* because it is responsible for proposing transaction blocks to the secondary replicas. The primary replica is successful if all replicas in the committee accept its proposal before replicas time out. The remaining nodes are appointed as secondary replicas often referred to as *validators* because these secondary replicas validate the primary replica's instructions or validate the current PBFT state by communicating if they agree with the instruction or state.

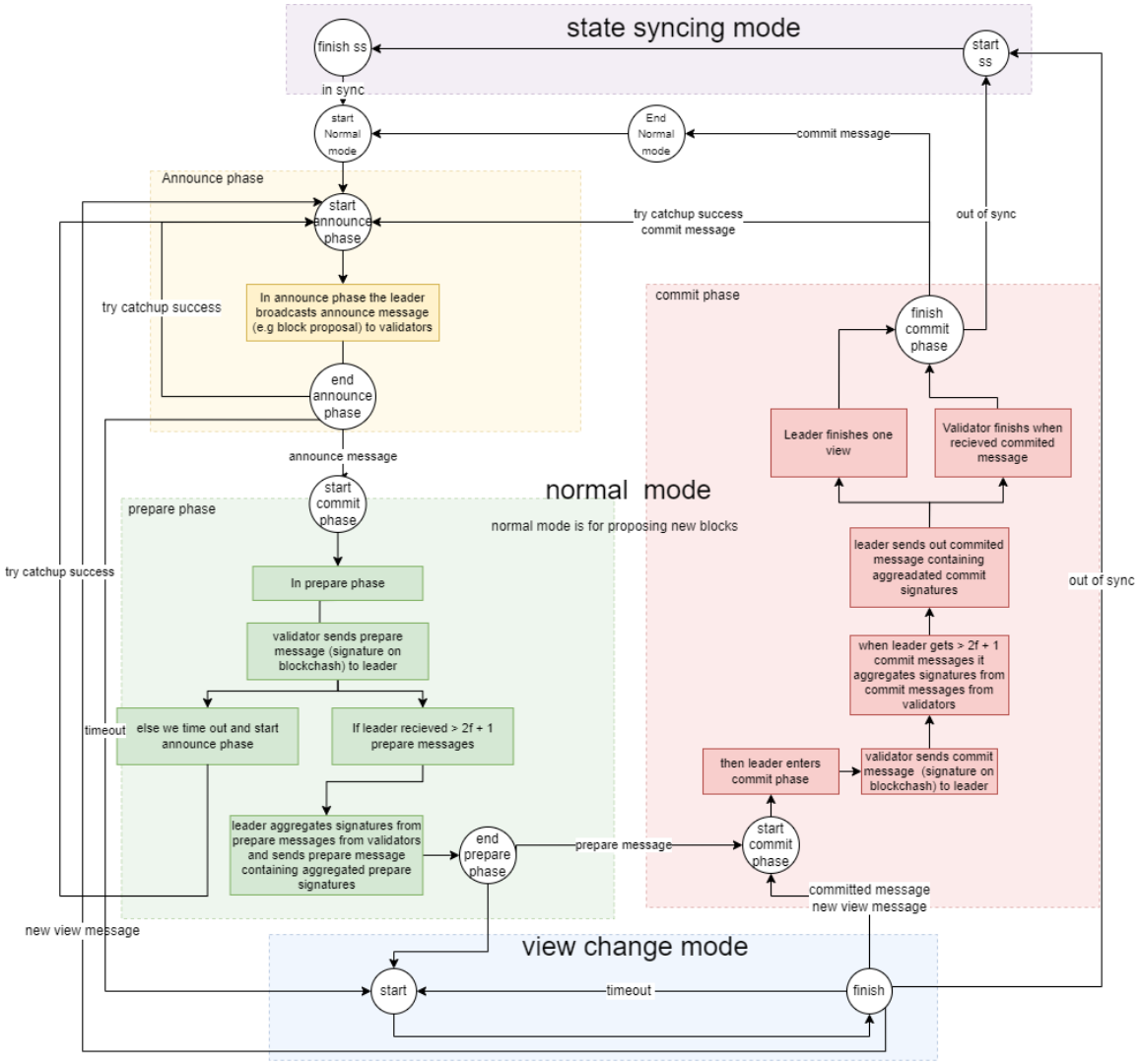


Figure 3.1: PBFT State Machine Diagram

The PBFT state machine consists of five states as displayed in Figure 3.1. The functionality of each state is listed as such :

- **State-Sync** : When a new replica joins the committee it will enter *State-Sync* state to synchronise with other replicas in the committee.
- **View-Change** : The primary replica referred to as the leader is rotated when there is a *View Change* request from the secondary replicas. *View Change* ensures liveness

in the committee by allowing the committee to vote in a new primary replica and progress when the current primary replica fails.

- **Pre-prepare** : *Pre-prepare* is stage one in the block proposal process. The primary replica sends a message to the secondary replicas which will include the proposal block with the view number (corresponding uniquely to the primary replica) and a sequence number, which can be described as the numeral order of the action being undertaken. When a secondary replica receives this message it enters *Prepare* state.

- **Prepare** : *Prepare* is stage two in the block proposal process. All replicas will broadcast a prepare message (e.g. signature on block-hash) to every node. When all replicas receive a supermajority of prepare messages, it will enter *Commit* state.

- **Commit** : *Commit* is the final stage in the block proposal process. All replicas will broadcast a commit message to signify full commitment to include the proposed block in the blockchain. When a validator receives a supermajority of commit messages, it can safely commit the block. This ends one round of the normal consensus process.

3.1.2 PBFT Security Proof

The main security proof for PBFT which ensures both *liveness* and *safety* is that our committee requires $\geq 2f + 1$ participants in a round of communication given f faulty nodes. Requiring $\geq 2f + 1$ participants means we can always finish a round of communication even if all f malicious nodes participate.

Safety is defined as the ability to proceed with operations knowing something bad **will not** occur given adversarial conditions. To ensure safety the secondary replica must execute all requests with a sequence number lower than the current sequence number. This is to ensure all secondary replicas execute requests in the same order. Safety is mainly ensured by requiring $\geq 2f + 1$ participants in the round of communication when we have at most

f adversarial or malicious replicas in our committee. This means even if all f malicious replicas participate in the round of communication the remaining $f+1$ honest replicas represent the majority of participants. Therefore we know an honest round of communication will occur despite the presence of malicious replicas.

Liveness is defined as the ability to proceed with operations **eventually** given adversarial conditions. PBFT synchronises replicas through the use of timers. Each state in the PBFT state replication algorithm will have a time that the state must complete its operation in. If operation within the state hasn't been completed when the time expires a timeout exception will occur in all replicas. This is why PBFT is so powerful. Since all replicas using PBFT will be in the same state, all replicas will automatically move to the next PBFT state together which will handle the cause of the timeout even if replicas time out individually. Therefore timeouts prevent adversarial primary replicas from choosing to simply not send instructions to secondary replicas which would permanently halt the committee's progress.

3.2 Multi-Signatures

Multi-signatures were first introduced in [14] where participants could create signatures by using public and private key encryption. A list of signatures could then by aggregated together into a multi-signature. This multi-signature could be verified using the list of participants' public keys and the original message which was signed by participants.

3.2.1 Schnorr Multi-Signatures

Bitcoin's adoption of Schnorr signatures was the first main integration of multi-signatures into decentralized blockchains. Bitcoin transactions require aggregating multiple signatures into a single signature in order to improve the capacity of the bitcoin network. It is also useful for aggregating and verifying the signatures associated with transactions contained in each block in the blockchain. The size of the bitcoin blockchain with and without Schnorr multi-signatures was investigated in [11] where a possible 30% reduction in the

size of the blockchain was discovered after implementation of Schnorr multi-signatures. However Schnorr signatures are not effective to use for communication for a number of reasons :

1. Multiple communication rounds are required.
2. Reliance on random number generation.
3. Multi-signature scheme is complex.
4. We cannot combine all signatures in the block into a single signature.

BLS multi-signatures do not have the drawbacks listed above and resulting a useful signature scheme for use during communication in sharded blockchains. As well as not being susceptible to the problems listed above, BLS multi-signatures are half the size of Schnorr signatures. ByzCoins BFT [8] communication implementation demonstrated effective use of constant sized Schnorr multi-signatures. However the Schnorr scheme requires an additional round-trip called the secret commitment round. This means Schnorr signatures require two rounds to create a multi-signature compared to BLS multi-signatures single round, which means BLS is at-least 50% faster than Schnorr.

3.3 BLS Multi-Signature

Due to the compactness, scalability and fast verification processes available which we will investigate later, BLS multi-signatures have proven much more suitable for use during communication in PBFT. Integration of BLS multi-signatures into communication with PBFT was explored in [21] and the new BLS multi-signature integrated PBFT protocol was labeled *Fast Byzantine Fault Tolerance* (FBFT).

Variable Definitions	Functions
SK = private key	*H()* = Hash function
PK = public key	*e()* = Elliptic Curve
M = message	
S = signature	
G = generator point	

BLS signatures are unique and deterministic meaning only one private key and message creates one signature and two different signatures cannot be validated by the same public key and message. BLS signatures use a modified hashing algorithm that hashes a message directly to the elliptic curve. The operation used to validate BLS signatures is called the curve pairing operation *e* which is a function that takes two points *P* and *Q* on an elliptical curve and maps them to a number :

$$e(P, Q) \rightarrow n$$

We also require one important property from this function. If we have some secret number α and two points *P* and *Q* we should obtain the same result regardless of which point we multiply by this number:

$$e(\alpha \cdot P, Q) = e(P, \alpha \cdot Q).$$

This means we need to be able to swap multipliers of the points between two arguments without changing the result. In a basic explanation these swaps should give the same result. This important property is what is used to create and validate BLS multi-signatures.

3.3.1 Public & Private Key Generation :

To create a public and private key a replica will generate a random secret number SK which must be kept confidential as it is the replica's private key. SK must be kept

confidential to prevent forgery of signatures. The replica will then use SK to generate a public key PK by multiplying SK by a point G on the elliptical curve named a *generator point*. SK can be used to create a BLS signature and PK is used to verify the BLS signature.

3.3.2 Signature Creation & Verification :

A replica will generate its signature by hashing a message to the elliptical curve and multiplying the resulting point by its private key :

$$S = SK \cdot H(M)$$

A replica can send this signature S along with the message M and the replica's public key PK to other replicas to signify approval of a PBFT state. Replicas will receive this message and verify the signature using the accompanying public key PK and message M using an elliptical curve operation :

$$e(G, S) = e(PK, H(M))$$

To verify BLS signature we just need to check that the public key PK and the message hash $H(M)$ are mapped to the same number as the curve generator point G and the signature S. Figure 3.2 shows how the equation $e(G, S) = e(PK, H(M))$ operates in relation to the elliptical curve.

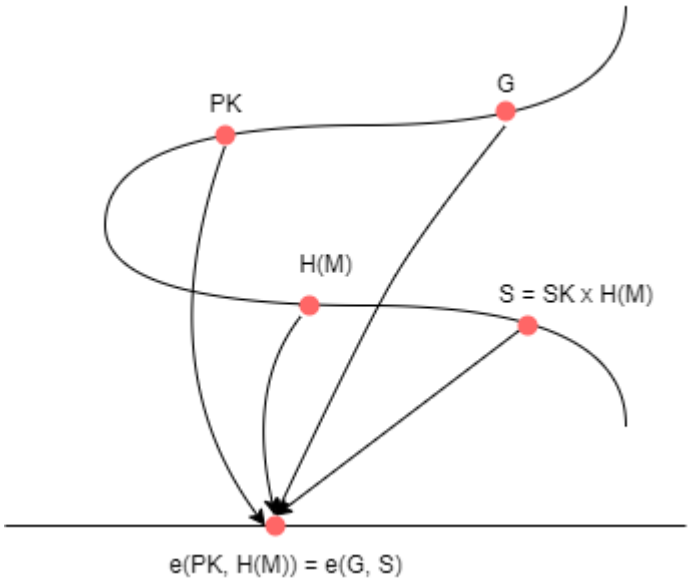


Figure 3.2: Elliptical Verification

3.3.3 Multi-Signature Creation :

We can combine BLS signatures together into a multi-signature. Multi-signatures allow a group of individuals to aggregate their signature's $\sigma_1 \cdot ... \cdot \sigma_n$ into a multi-signature σ such that a verifier can check if each individual participated in creating σ. Aggregated signatures will be calculated as the product of all signatures. We can aggregate signatures σ_i for $i = 1 \rightarrow n$ together into a multi-signature σ, which is the same length as any other BLS signature, by computing :

$$\sigma \leftarrow \sigma_1 \cdot ... \cdot \sigma_n$$

3.3.4 Multi-Signature Verification

Given triples (PK_i, M_i, σ_i) for $i = 1 \rightarrow n$ anyone can verify a multi-signature σ by checking that the following equality holds:

$$e(G, \sigma) = e(PK_1, H(M_1)) \cdot ... \cdot e(PK_n, H(M_n))$$

When all the messages being signed are the same $(M_1 = ... = M_n)$ the verification reduces to a simpler equation that requires only two pairings using the equal message M :

$$e(G, \sigma) = e(PK_1 \cdot ... \cdot PK_n, H(M)).$$

3.4 Attacks on BLS Multi-Signatures

3.4.1 Rogue Key Attack

The rogue key attack is BLS multi-signatures only non-trivial attack. The attack can be mounted by the primary replica during communication rounds to make it seem as if a group of replicas approves of a decision or PBFT state that they might not have approved. The rogue key attack can only be mounted when replicas sign the same message $(M_1 = ... = M_n)$.

Variable Definitions
PK_A = attacking node's public key
PK_H = honest node's public key
RK = rogue key
S_A = attacking node's signature
σ = multi-signature
M = message we are signing

When signers sign the same message the signature aggregation method is insecure by itself due to a rogue key attack, where an attacker calculates a rogue key *RK* :

$$RK := PK_A \cdot (PK_H)^{-1}$$

Using the Rogue Key

The attacker then includes this rogue key RK as it's public key instead of its authentic, real public key in the list of participating public keys that verify the BLS multi-signature. For example, PK_H is a public key of some unsuspecting, honest user Bob. The attacker can then claim that both it and Bob signed the message M by calculating the RK, publishing the RK as the attacking nodes actual public key and then presenting its own signature S_A as the multi-signature σ. Bob will verify the multi-signature (which is actually just the attackers signature S_A) using Bob's public key and the attacker's public key (which is actually the rogue key RK) as such :

$$e(G, S_A) = e(RK \cdot PK_H, H(M)) = e(PK_A \cdot (PK_H)^{-1} \cdot PK_H, H(M)) = e(PK_A, H(M))$$

This equality will validate because when we multiply the honest node's public key PK_H by the rogue key published by attacking node RK we get the attacking node's original public key PK_A. Since the attacking node includes its own signature S_A instead of the actual multi-signature σ the equation will be valid since it is simply checking :

$$e(G, S_A) = e(PK_A, H(M))$$

Attack with Multiple Participants

To verify a BLS multi-signature we multiply together the public key of each replica who contributed their signature to creating the BLS multi-signature. The exact equation used to validate the BLS multi-signature is :

$$e(G, \sigma) = e(PK_1 \cdot .... \cdot PK_n, H(M)).$$

The list of public keys $PK_1....PK_n$ represents the public keys that provided a signature that was used in creating the BLS multi-signature σ. The rogue key attack is an exploit on the process of multiplying all the participating public keys together. The attacker

calculates a rogue key RK that, when multiplied by the other participating public keys, equals the attacker's original public key PK_A. In effect :

$$(PK_1 \cdot ..... \cdot PK_{n-1} \cdot RK) = PK_A$$

The attacker calculates the rogue key by multiplying the attackers real public key PK_A by the inverse of the of all the honest participants public keys multiplied together.

$$RK = PK_A \cdot (PK_1 \cdot ..... \cdot PK_{n-1})^{-1}$$

The attacker includes the *RK* as its own public key in the list of participating public keys. Let us label $(PK_1 \cdot ..... \cdot PK_{n-1})$ as PK_H. When the *RK* is used during multi-signature verification we get the following equation:

$$PK_H \cdot RK = PK_H \cdot PK_H^{-1} \cdot PK_A$$

In the above statement the $PK_H \cdot PK_H^{-1}$ reduces to 1 leaving us with the attacker's original public key PK_A. After the public keys reduce to the attacker's original public key, we simply check if the attacker's public key validates the multi-signature. Since the attacking node replaces the multi-signature with its own signature the check explained above will validate as true. Therefore the rogue key attack worked.

3.5 Defense Mechanisms

There are two main existing defense mechanisms against the Rogue Key attack. These defense mechanisms are namely *Distinct Messages* and *Proof-of-Possession*.

3.5.1 Distinct Messages

We can require that all the messages being signed are distinct and unguessable. To aggregate signatures together into a multi-signature, the committee primary replica will gather each participating secondary replica's public key, distinct message and signature.

Once a supermajority has participated, the primary replica will aggregate signatures together into a multi-signature by computing :

$$\sigma \leftarrow \sigma_1 \cdot ... \cdot \sigma_n \in G0.$$

After the primary replica aggregates signatures into a multi-signature it broadcasts a message that includes the multi-signature; the list of participating public keys, and the list of distinct messages to all replicas in the committee. Each replica then verifies the multi-signature using each of the public keys and distinct messages as such :

$$e(G, \sigma) = e(PK_1, H(M_1)) \cdot ... \cdot e(PK_n, H(M_n))$$

There is no way to calculate and use a rogue key since we don't aggregate the public keys together into a single public key to verify the multi-signature. A rogue key simply does not exist.

Protocol :

1. If a secondary replica approves of the current PBFT state it signs a distinct, unguessable message to create its signature.
2. The secondary replica sends its signature, public key, and distinct message to the primary replica.
3. The primary replica collects signatures, public keys, and distinct messages from approving secondary replicas.
4. The primary waits for the supermajority of secondary replicas to participate (i.e send their signature, public key, and message to the primary replica).
5. After this condition is met the primary replica aggregates all collected signatures into a BLS multi-signature.

6. The primary replica broadcasts this BLS multi-signature including all participating public keys and associated distinct messages to every replica in the committee.
7. Each replica receives this message,
8. Each replica validates the BLS multi-signature with the list of public keys and the list of messages using the equation outlined above.
9. If the multi-signature validates we have completed the current round of communication.

Pros:

1. Distinct message defense is provably secure against the rogue key attack. The rogue key attack is a vulnerability in the public key aggregation stage for multi-signature verification when signers sign the same message. When signers sign distinct messages during signature aggregation we cannot aggregate public keys together and instead need to validate the signature against each participant's signature and public key separately. As a result, we do not aggregate the public keys together and therefore an adversary cannot forge a rogue key to publish as its own public key.
2. Distinct Messaging defense is simple to implement. Distinct message defense does not require additional changes outside of the messaging protocol like *Proof-of-Possession*. The only requirement is that the message being signed by the replicas must be distinct. This means the message must change for each communication round. This prevents a malicious replica from storing and reusing an honest node's signature.

Cons:

1. Since all messages are distinct, we cannot take advantage of the efficiency multi-signature verification equation that applies when replicas sign a common

message m. This significantly reduces scalability which is displayed in the later analysis and evaluation section.

3.5.2 Proof-of-Possession

PoP works by ensuring a replica has access to the private key associated with the public key it claims to own. For BLS signature schemes, this requires a validation process to occur outside of the consensus process. Instead, the process occurs in *State-Sync*, where replicas create a signature called a *proof* by signing a publicly available message using the private key associated with its public key. Replicas must then send both its public key and proof to other replicas in the committee. Replicas check that the proof verifies the public key and if the proof is valid replicas will store the public key in a local table. This table represents the valid public keys within the committee. PoP is an intricate defense mechanism and requires relatively more care to integrate. A replica must check each participating public key that accompanies a BLS multi-signature is contained in the local public key and proof table whenever a BLS multi-signature is being verified. If the public key is not contained in the table it is unverified and must be discarded, therefore preventing the rogue key attack.

Protocol :

1. If a secondary replica approves of the current PBFT state it signs a publicly available message to create its signature
2. The secondary replica will then send this signature and its public key to the primary replica.
3. The primary replica collects signatures and public keys from approving secondary replicas.
4. The primary replica waits for the supermajority of secondary replicas to send their signatures and public keys to the primary replica.

5. After this condition is met the primary replica aggregates all collected signatures into a BLS multi-signature.
6. The primary replica broadcasts this BLS multi-signature along with all participating public keys to every replica in the committee.
7. Each replica receives this message.
8. Each replica checks the public keys accompanying the BLS multi-signature are contained in the public key and proof table.
9. If this check is valid, each replica validates the BLS multi-signature with the list of public keys.
10. If the multi-signature validates we have completed the current round of communication.

Pros:

1. We can allow replicas to sign the same publicly available message. When all the messages being signed are the same $(M_1 = ... = M_n)$ the verification equation reduces to a simpler test that requires only two pairings:

$$e(G, \sigma) = e(PK_1 \cdot ... \cdot PK_n, H(M)).$$

A result of the simpler multi-signature verification is that we can increase the number of replicas participating in creating a multi-signature without causing a huge increase in the time it takes to verify the multi-signature. This significantly increases scalability within the committee.

Cons:

1. Integrating PoP takes much more care and attention. PoP assumes that replicas contain a list of verified public keys, where each public key has been verified by a

proof of possession. To satisfy this assumption, functionality that maintains the primary and secondary replicas list of verified public keys must be in place. This functionality is suitably implemented in the *State-Sync* state in PBFT as *State-Sync* is responsible for synchronizing the state and information of all members and the leader in our group. The simplest way to ensure such functionality is by requiring all replicas to broadcast their PoP proof and public key when they enter *State-Sync* as shown in Figure 3.3

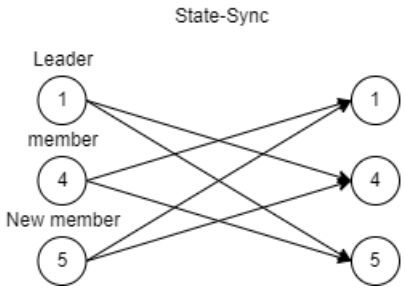


Figure 3.3: State Sync with Proof-of-Possession

Assuming that each member and the leader has a verified list of public keys the group can exit the *State-Sync* state and return to normal consensus.

Chapter 4

Proposed Defense Designs

In this section we outline our new improved defense designs that prevent rogue key attacks on BLS multi-signatures used during communication in sharded blockchains. The two improved designs are called *Public Key Cert Table* protocol and *Leader Excluded* protocol. Both of the improved designed solutions demonstrate important improvements over the existing *Proof-of-Possession* and *Distinct Messages* defenses which will be outlined and explained in the evaluation and analysis sections.

4.1 Public Key Cert Table Protocol

Public key infrastructure (PKI) and associated technologies have gradually been applied to a growing range of problems. PKI technology has proven to be very useful in coordination, registration, verification, and storage of network information. PKI technology also typically guarantees secure information handling. Done correctly, the integration of PKI technology into blockchains does not need to come at the expense of decentralization, security or speed. In the *Public Key Cert Table* solution we explore how we can use PKI technology to provide security against rogue key attacks, reduce message size and to reduce the time it takes to reach consensus.

In this proposed solution we use PKI technology to solve the following issues :

1. By integrating PKI technology into our system we reduce the amount of bytes required to transmit the list of participating public keys from the primary replica to the secondary replicas by 97%. In all existing BLS implementations a primary replica must include the list of public keys who participated in creating the multi-signature signature when messaging the secondary replicas. If we assume we have 300 secondary replicas in our committee and $f = 100$. f is the maximum number of malicious nodes in our committee in order to have a safe communication round. We require a minimum of $\geq 2f + 1$ participants which equates to 201 participating secondary replicas. This means the primary replica must send a public key list of size $9.6kB$ (each public key is 48 bytes long). In our proposed solution we use PKI technology to reduce the public key list down to a simple bitmap. A bitmap is defined as a mapping from some domain (in our case the public keys which participated in creating the multi-signature) to bits. With *N* replicas in our committee we reduce the public key list of size $N \cdot 48$ bytes to a bitmap of size $N \cdot 1$ bit. In our above example with 201 participating replicas we reduce the 9.6 kB public key list to a 201 bit long bitmap.

2. By incorporating public key certs we open potential avenues for additional security. We know the information stored in the public key certs is legitimate & authentic.

Therefore, we can prevent rogue key attacks on BLS multi-signatures by exploiting PKI's technology. This is because PKI technology offers public key certificate management, user identity validation, secure user certificate storage, and auxiliary user information storage. This method has not been replicated anywhere else.

4.1.1 Protocol, Security, Pros & Cons

In the designed Public Key Cert Table solution there exists a trusted third party (TTP) called the *Certificate Authority* (CA). The CA is a trusted entity authorized by a group of nodes to

create, assign and handle public key certificates. A public key certificate is a verifiable data structure containing the required information for any node / replica to validate, verify, and authenticate the information contained in the certificate. The certificate should contain the name of a cert owner such as a node ID, the expiration period for the cert, and the node's public key. The CA will sign the information using the CA's private key and will also include this signature in the certificate. The CA's public key will be publicly available so that nodes can verify the CA signature contained in certificates. When a node is registering to participate on the blockchain network it sends its information to the CA for certificate creation. The CA will create a certificate with the node's information and store the cert on the blockchain. All certificates are then stored on the blockchain so they can be accessed by any node on the blockchain network and used at a later stage. The CA will reply to the registering node with a reference to where the certificate is stored on the blockchain once the CA has confirmation that the certificate has been successfully stored on the blockchain. This way the node can send the certificate reference to other nodes. Figure 4.1 displays how public key certificates will be stored in the blockchain as such :

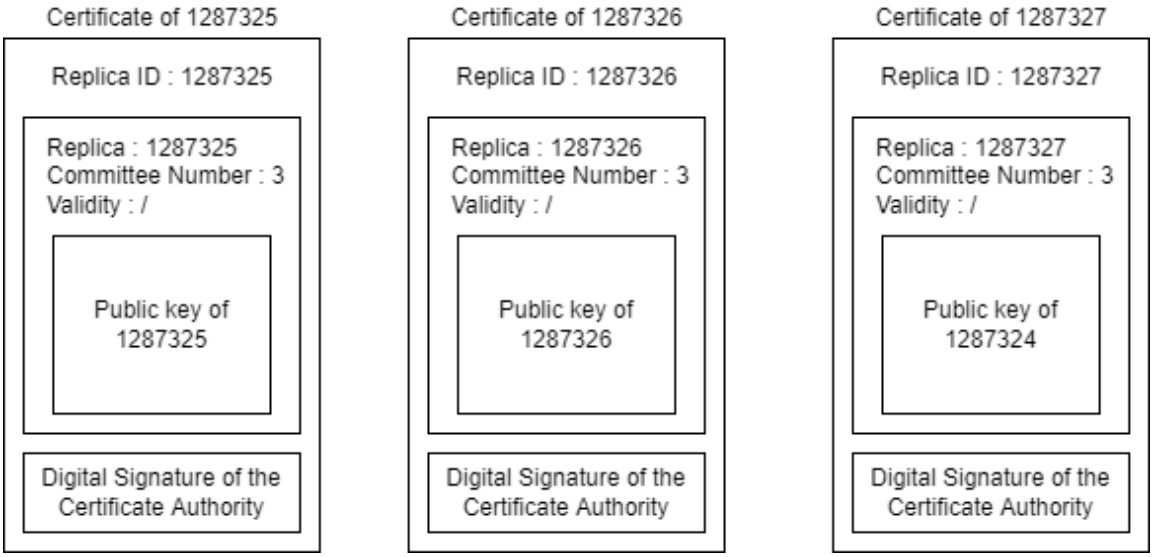


Figure 4.1: Certs Stored on Blockchain

In our designed Public Key Cert Table solution every replica must have a table con-

taining the public key cert of each replica in the committee. To create our public key cert table we can add a step to *State-Sync* state where all replicas send the primary replica their reference to the location on the blockchain where their public key cert is stored along with their proof of possession. The primary replica will broadcast all received references and proofs to all replicas in the committee. A replica can then retrieve any new public key certs from the blockchain using the list of public key certificate references received from the leader. A replica will validate each certificate that was retrieved from the blockchain using the PoP proof that accompanied the public key certificate reference. A replica can also remove any public key certs it did not receive a reference to in order to remove outdated certificates. The result will be each replica now having a local, secure copy of public key certs that contain the valid public keys that exist in the replica's committee as shown in Figure 4.2. It is important to note that a public key cert table is cached in each replica. The public key cert table for each committee will change over time as replicas join and leave the committee, so permanently storing certificates is a bad idea and instead certs should be cached.

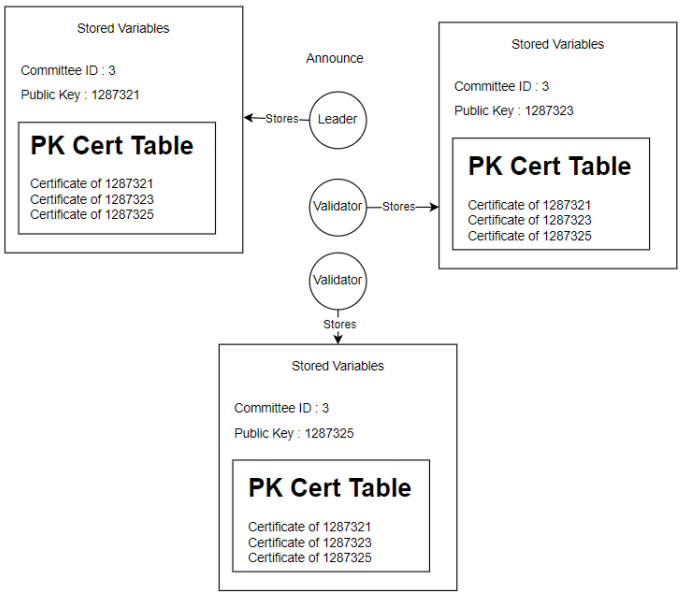


Figure 4.2: Cert Table Cached in Each Replica

All replicas can now reference the locally cached public key certificate table as shown in Figure 4.2 during multi-signature verification. When a primary replica broadcasts a BLS multi-signature to all secondary replicas it will include a bitmap that references participating public keys as shown in Figure 4.3.

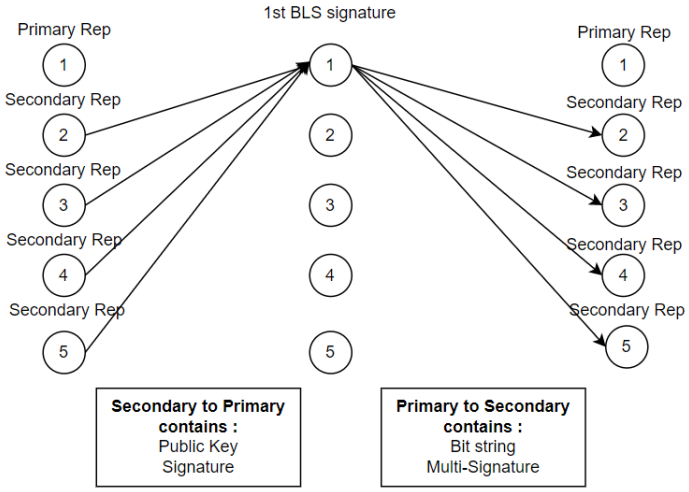


Figure 4.3: PBFT Communication using Certs

The bitmap indicates whether a public key participated in creating the most recent multi-signature. A ‘1’ indicates the corresponding public key participated and ‘0’ indicates non-participation. Each secondary replica cross-references its local public key cert table as shown in Figure 4.5 to deduce which public keys contained in its public key cert table participated in creating the multi-signature. After deducing the participating public keys from the public key cert table the replica validates the multi-signature using the list of participating public keys and the publicly available message M which was signed.

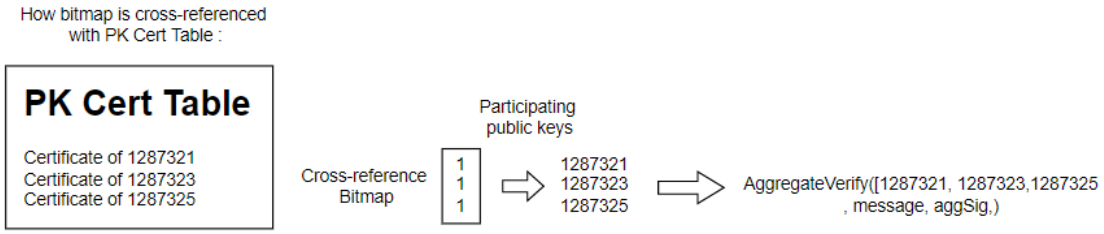


Figure 4.4: Bitmap Use Case

We have now figured out which public keys have participated in creating the multi-signature without having the primary replica transmit a list of public keys to the secondary replicas. Instead, the primary replica only transmitted a bitmap. Replicas can now speedily validate the multi-signature using the list of participating public keys in only two elliptical operations. Representing information as a bitmap is something that has been done in a more basic fashion in [15] where the the MOTOR protocol is introduced. MOTOR is a robust and scalable BFT-consensus protocol suitable for sharded blockchains. The MOTOR protocol implements the use of bitmaps to indicate which public keys have participated in the aggregated signature. In the Public Key Cert Table solution we use the same idea of implementing bitmaps to significantly increase communication efficiency. We do not require a trusted leader. One major flaw in MOTOR's consensus protocol is that it requires a trusted leader. Realistically, in decentralized blockchains we cannot assume the leader is trusted.

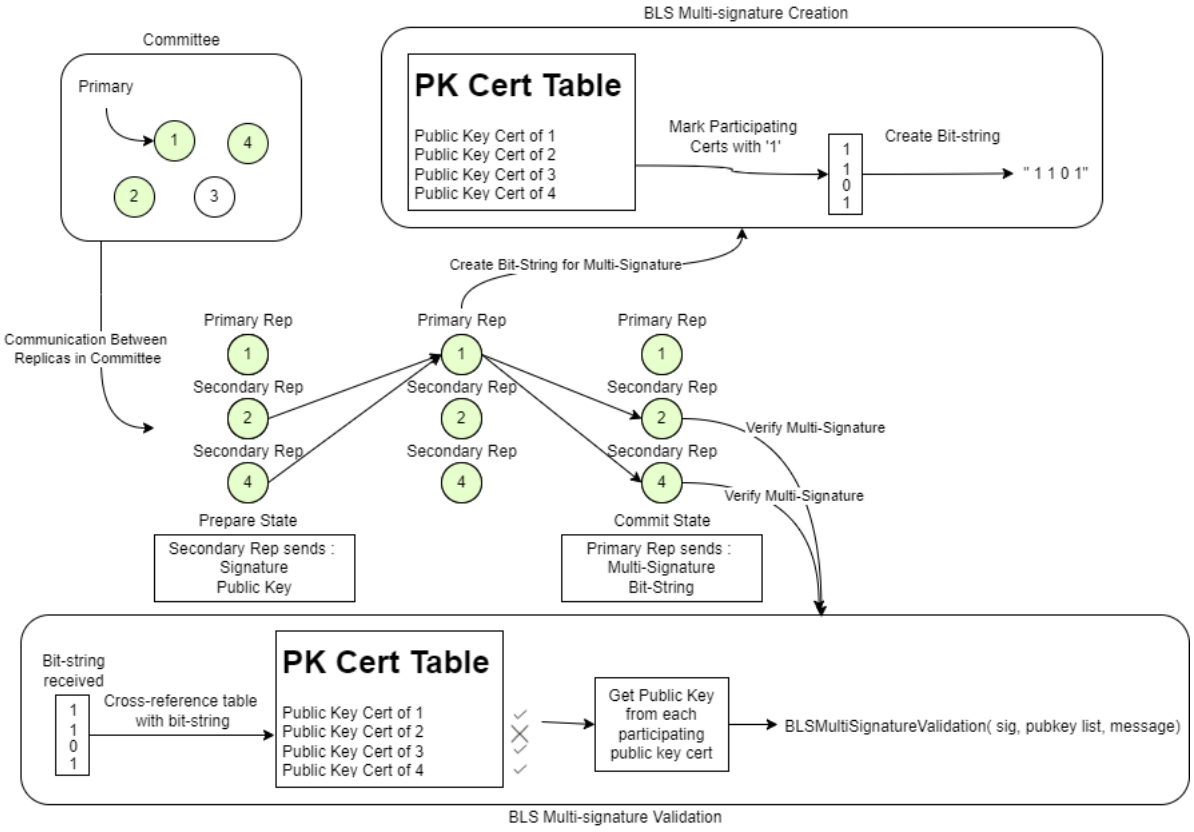


Figure 4.5: Pub Key Cert Table Protocol

Protocol :

Assuming each node has gone through the necessary steps to initialize and store a legit local public key cert table our protocol will operate as such :

1. If a secondary replica approves of the current PBFT state it signs a publicly available message to create its signature .
2. The secondary replica will send its signature and public key to the primary replica.
3. The primary replica collects signatures and public keys from approving secondary replicas.
4. The primary waits to collect the supermajority of signatures and public keys.

5. After this condition is met the primary replica aggregates all collected signatures into a BLS multi-signature.

6. The primary replica goes through its local public key cert table and adds a 1 to the bitmap if the public keys contained in the public key cert participated in creating the multi-signature. The primary replica adds a 0 to the bitmap if the public key did not participate.

7. The primary replica broadcasts this multi-signature and bitmap to all secondary replicas.

8. Each secondary replica receives this message.

9. Each secondary replica cross-references its local public key cert table with the bitmap. If the bit equals 1 it means the corresponding public key cert in the table participated and if the bit is 0 it means it didn't participate. This is how secondary replicas parse the list of participating public keys.

10. Each replica validates the BLS multi-signature with the parsed list of participating public keys.

11. If the multi-signature validates we have completed the current round of communication.

Security

This method prevents any adversaries from trying to register / include more than one signature in the aggregated signature. This method is not 100% rogue key resistant, however the primary replica cannot calculate and issue a rogue key. A rogue key attack is when the primary replica issues a public key that, when multiplied by the non-adversarial public keys during public key aggregation, equals the attacker's original public key. If there exists a public key in the public key cert table cached in replicas equal to the rogue public key,

the attacker could include that public key as participating by marking the public keys corresponding bit to '1' in the bitmap and issuing the bitmap to the secondary replicas. When secondary replicas calculate the aggregated PK according to the bitmap it will equal the adversaries PK. Therefore, the attacking primary replica managed to mount a rogue key attack.

However the odds of a rogue public key being in the public key cert table is extremely small (approx. $1/3.940210^{115}$). To make the attack even more unlikely the adversary would need to get three rogue public keys in a row for a full round of consensus which we can assume is impossible. As there is an extremely small risk for a rogue key attack, our proposed solution can be considered a valid defense solution.

Preventing Man-in-the-Middle Attacks

A man-in-the-middle (MITM) attack is a cyberattack where an attacker inserts themselves between two honest parties in a network. The attacker relays and possibly alters the communications between the two parties who believe that they are directly communicating with each other. A concern in a blockchain network is an attacker sitting between two replicas. The attacker could alter the information being sent between the replicas resulting in a denial of service or, in the worst case scenario, a complete takeover of the committee.

We need to ensure a MITM attacker is not altering the bitmap during transmission from the primary replica to secondary replicas in order to maintain verification of multi-signatures. To detect a MITM attack we note two facts. Firstly, we note that when the public key cert table is being created the primary replica will indicate which public key cert it owns to secondary replicas. This means secondary replicas can access, validate and use the primary replica's public key from the public key certificate. Secondly, we note that during the Pre-prepare state the primary replica broadcasts the block number to secondary replicas. This block number is a label for the new block being proposed by the primary replica and the block number being broadcasted changes every communication round be-

cause each round is responsible for proposing a different block.

Since all replicas know what the current block number is meant to be we can use the block number as a checksum. Previously, if a MITM attacker changed the bitmap there was no way to detect it. Now in our proposed Public Key Cert Table defense the primary replica will create a new signature by concatenating the block number onto the bitmap and then sign the result with it's private key. We will call this new signature the secure bitmap. The primary replica broadcasts the multi-signature along with the new secure bitmap to secondary replicas. Secondary replicas decrypt the secure bitmap with the primary replicas public key and check if the block number is concatenated onto the end of the bitmap correctly. If the block number is incorrect, secondary replicas can assume the bitmap has been tampered with and will request a view change which will elect a new primary replica since the responses from the current primary replica are being altered.

Pros

1. Public Key Cert Table protocol would be significantly more efficient than any existing BLS-aggregation defense protocol. From our simulation we run below we see a 97% reduction in the size of messages and a slight 10% reduction in the time taken to run one full round of consensus when compared to the most efficient existing defense mechanism *PoP*. This is a massive improvement and demonstrates the power of locally storing public key information in every node in our committee and referring to public key information with bitmaps.

2. We can allow secondary replicas of a committee to sign the same message. This allows the use of the *fast multi-signature verification* equation which reduces the number of elliptical operations required to verify the multi-signature from n where n equals the number of secondary replicas in a committee to two.

3. Increase security avenues. By using public key certs we open up new possibilities for storing and verifying public key information.

Cons

1. Creating the public key cert table solution requires an overhaul of the blockchain system. We cannot implement the public key cert table solution without implementing the required systems to create, store, retrieve and verify the public key certs. As a result care must be taken when implementing the complex public key cert table solution.

4.2 Leader Excluded Protocol

In the Leader Excluded protocol we use the fact that the primary replica is the only node that can mount a rogue key attack on the group of secondary replicas. We can use this fact to prevent a rogue key attack by ensuring the primary replicas signature and public key doesn't equal the multi-signature and the result of multiplying all public keys together during multi-signature verification. This functionality requires the primary replica to include their public key PK_0, signature σ_0 and the message M_0 used to create the signature when sending the multi-signature σ to secondary replicas. Secondary replicas require additional checks to prevent a possible rogue key attack. Secondary replicas must first verify :

1. Verify primary replica's information : $e(G, \sigma_0) = e(PK_0, H(M_0))$
2. Verify multi-signature does not equal primary replica's signature : $\sigma \neq \sigma_0$
3. Verify rogue key hasn't been included : $(PK_1 \cdot ... \cdot PK_n) \neq PK_0$
4. Verify multi-signature : $e(G, \sigma) = e(PK_1 \cdot ... \cdot PK_n, H(M))$.

If all checks verify we accept the multi-signature.

4.2.1 Protocol, Security, Pros & Cons

Protocol

The Leader Excluded protocol requires replicas to sign the same publicly available message. This means we can use the fast multi-signature verification equation which allows us to valid the multi-signature in only two operations. The protocol requires additional verification outlined above to prevent rogue key attacks. The Leader Excluded protocol operates as such :

1. If a secondary replica approves of the current PBFT state it sends a publicly available message and sends its signature along with its public key to the primary replica.
2. The primary collects signatures and public keys from approving primary replicas
3. The primary waits for the supermajority of secondary replicas to send their signatures and public keys to the primary replica.
4. After this condition is met the primary replica aggregates all collected signatures into a BLS multi-signature σ.
5. The primary replica broadcasts σ, all participating public keys, pk_0, σ_0 and M_0 to every replica in the committee.
6. Each replica receives this message
7. Each replica runs the Rogue Key checks outlined above
8. If this condition satisfies each replica validates the BLS multi-signature with the list of public keys
9. If the multi-signature validates we have completed the current round of communication.

Security

The Leader Excluded protocol is verifiably secure against the Rogue Key attack. We use the fact that the primary replica also known as the leader is the only replica in the committee which can mount a rogue key attack. We can therefore single out and validate just the primary replica / leader using the following verification process :

1. Verify primary replica included correct information : $e(G, \sigma_0) = e(PK_0, H(M_0))$
2. Check for a rogue key attack $\sigma \neq \sigma_0$ and $(PK_1 \cdot ..... \cdot PK_n) \neq PK_0$
3. Verify multi-signature $e(G, \sigma) = e(PK_1 \cdot ... \cdot PK_n, H(M))$.

After all the above checks have been validated we know the leader has not tried to mount the rogue key attack.

Pros

1. With larger committee sizes we have a reduced message size from the primary replica to secondary replicas compared to the existing distinct message protocol. However due to the fact that we have to include additional information such as the primary replicas signature, message and public key the message size for the Leader Excluded protocol compared to proof of possession is larger. Therefore we reduced the message size compared to Distinct messages however increased compared to proof of possession
2. We can use *fast multi-signature verification*. Since every message is the same we can use the faster verification equation that only requires two elliptic curve operations for verification instead of the costly normal multi-signature verification which requires $2 \cdot N$ elliptic curve operations.
3. Very easy to implement. The Leader Excluded protocol has the same level of difficulty to implement as the Distinct Messaging defense protocol which is by far the easiest existing defense mechanism to implement. The only additional complexity in

the Leader Excluded protocol comes from the additional required checks to prevent the rogue key attack.

Cons

1. Leader Excluded protocol has significantly worse scalability compared to Proof of Possession and Public Key Cert Table defenses. Even though Leader Excluded protocol is significantly easier to implement it still has worse scalability in regards to the time taken for one full round of consensus which involves one pre-prepare, prepare and commit phase using BLS multi-signatures.

Chapter 5

Simulation

Something that is missing from all existing studies is a simulation of a committee reaching consensus using BLS multi-signatures with each defense mechanism. We design and build a simulation of the communication process that occurs between replicas using multi-signatures with the two most popular existing defense mechanisms *Proof-of-Possession* and *Distinct Messages* and our two designed defense mechanisms *Leader Excluded* and *Public Key Cert Table*. We only simulate the pre-prepare, prepare and commit state as these states are the most common states for a committee running PBFT as they are responsible for reaching block consensus within the committee. In the pre-prepare state the primary replica will broadcast the blockhash to all secondary replicas in the committee. In the prepare and commit states if a secondary replica agrees that the current PBFT state is valid and wants to proceed to the next state the secondary replica will send its signature, and additional information to the primary replica. Any secondary replica that sent its signature and information to the primary replica is considered a 'participating secondary replica' as the secondary replica is participating in consensus and communicating with the primary replica. The primary replica will aggregate signatures received from participating secondary replicas into a BLS multi-signature and will broadcast the multi-signature, and additional information to all secondary replicas. A secondary replica will validate the multi-signature in order to safely move to the next state.

5.1 Simulation Implementation

In the simulation a committee moves from PBFT state to PBFT state (namely from pre-prepare to prepare to commit). Due to this behavior of moving between well-defined states we define the simulation as a discrete event simulation (DES). A DES is defined as "the process of codifying the behavior of a complex system as an ordered sequence of well-defined events". An in-depth analysis of DES provided by [13] a DES should have the following components : entities, activities and events, resources, global variables, random number generator, a calendar, system state variables and statistics collectors. A DES must also be a dynamic model of a dynamic system. In our case the dynamic system is the communication process between a group of replicas reaching block consensus in sharded blockchains. The dynamic model is our accurate and true simulation of this process. This process is dynamic because as more nodes are added to the group the communication process efficiency and speed dynamically changes too.

The dynamic model is programmed in python and was developed in the git repository [rowlanja/BLSAggregatedSignatureImplementation (github.com)] where you can track the changes as the simulation was built. This repository contains all information needed to run the simulation locally. To reflect the realistic behavior of an actual sharded blockchain, proper networking techniques were used to build the network of nodes that act as the primary replica and secondary replicas in a committee. These networking techniques involve socket programming and threading.

Socket programming is a way of connecting two nodes on a network so they can communicate with each other. *Node A* creates a TCP socket and listens on a particular port at an IP. *Node B* creates its own TCP socket and reaches out to Nodes A socket to form a connection. Now *Node A* and *Node B* can send messages in byte form over the connection. Sockets have two forms which are server sockets and client sockets. Client sockets connect to the server socket by specifying the server endpoint and then sending data. When

the client connects to a server a connection is pushed onto the server-side queue where it sits until the server accepts the connection. In a committee containing one primary replica and a group of secondary replicas, the primary replica will operate as the server and use a server socket and the secondary replicas will operate as clients using the client socket.

In order to handle multiple client connections a server must use threading. In an abstract definition threading allows us to have parts of a program run concurrently. When a primary replica receives a connection request from a secondary replica it will create the connection and also create a new thread θ to handle the connection over. The primary replica can then continue listening for any new connection request as θ is handling operations over the most recent connection. This is how realistic communication between secondary replicas and the primary replica is implemented as existing sharded blockchain networks would also use socket and threading programming to implement network communication.

Elliptical curve operations are also needed in our simulation. A BLS-Signature library is utilised for these operations. The library offers the required elliptical curves needed to create and verify signatures and to create and verify multi-signatures. This library also offers standard public and private key generation. The library is provided by Chia-networks and is utilized by existing decentralised blockchains.

The blockchain network consists of two main types which are defined as Node and Committee. Nodes form a committee. A committee is a group of nodes where one node acts as a primary replica and the other nodes act as secondary replicas. All secondary replicas follow the practical byzantine fault tolerance state machine. The primary replica of the committee relays information to and from secondary replicas so that secondary replicas know what PBFT state other secondary replicas want to move into. This way secondary replicas are synchronized and move from one PBFT state to another together even though each node is running its own independent instance of the PBFT protocol. Abstractions for all object types are using object-oriented programming.

Two more types called Simulation and Analyze are also needed. The Simulation data type carries out the simulation and saves results to datasets. The Simulation type creates a simulation by :

1. Set a X = minimum committee size and Y = maximum committee size
2. Initialize a group of Node types of size equal to X
3. Create a Committee with that group of Node types
4. Run a full consensus round of PBFT (pre-prepare, prepare and commit state)
5. Save analytics
6. Increment X
7. If $X == Y$: Exit Simulation
8. Else : Return to step 2

The Analyze data type is responsible for analyzing the data created by the Simulation data type. The Analyze type reads and parses the datasets and produces meaningful graphs. All data types can be found in the stated git repository.

5.2 Measured Statistics

Since are focused on optimizing communication in sharded blockchains we carefully select what metrics we track and analyze. Analysis is done on the following metrics :

1. **Message Size from Secondary Replica to Primary Replica** : During the prepare and commit state of PBFT a secondary replica must at minimum send its signature and its public key to the primary replica. Depending on which rogue key defense mechanism is implemented, secondary replicas also include additional information

when sending its signature and public key to the primary replica. There is very little to optimize for this metric.

2. **Message Size from Primary Replica to Secondary Replica** : The primary replica collects public keys and signatures from secondary replicas, aggregates the signatures into a multi-signature and messages every participating secondary replicas with at minimum the multi-signature and the list of public keys. Depending on which rogue key defense mechanism is implemented the primary replica also includes additional information. This metric is very important as it impacts communication scalability significantly.

3. **Time Taken to Reach Consensus** : Time taken to reach consensus represents the time taken for a committee to choose what block to include in the blockchain. Each different defense mechanism requires a different number of elliptical curve operations to verify the multi-signature and to detect and prevent rogue key attacks. Also each different defense mechanism requires a different amount of information to be relayed and verified to and from the primary replica. Both of these differences between defense mechanisms have an interesting effect on how long it takes to reach consensus and it is difficult to predict without a simulation.

4. **Replica Storage Requirements** : Replica storage requirements represents how much space a replica requires to store the information used to validate the multi-signature. Each different multi-signature defense mechanism takes a different approach to preventing Rogue Key attacks and as a result requires different information to be stored for validating the multi-signature. Analysing this metric is important because the blockchains of the future require low storage overhead for nodes on the network.

Message Size from Secondary Replica to Primary Replica

The largest message size from secondary replicas to the primary replica is Distinct Messages which is due to the fact that a secondary replica must include its signature, unique

message and its public key when participating in the multi-signature. However all message sizes from secondary replicas to the primary replica are between $143 - 148$ bytes. From analyzing message size from secondary replicas to the primary replica we see a constant message size as more replicas are added to the committee. This is logical as secondary replicas only include personal information such as its signature, public key and potentially other information. The personal information sent from secondary replicas to the primary replica is the same irrespective of the number of secondary replicas in the committee.

Message Size of Primary Replica to Secondary Replica

This metric is an important component of the communication scalability within sharded blockchains. As more secondary replicas are added to the committee more replicas participate in creating the the BLS multi-signature. As a result the amount of information needed to verify the multi-signature also grows meaning the primary replica must include more verification information when sending the multi-signature to secondary replicas. Finding an efficient and scalable way to represent the multi-signature verification information sent from the primary replica to secondary replicas is important.

Time Taken to Reach Consensus

To analyze how long it takes the committee to reach block consensus for one round of PBFT using multi-signatures we need to complete one Pre-prepare state, one Prepare state and one Commit state.

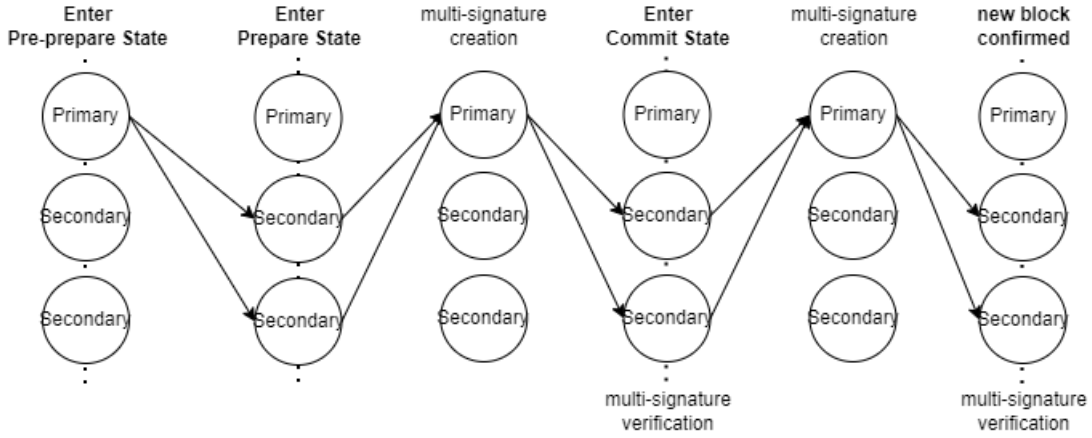


Figure 5.1: Block Consensus w/ PBFT

Pre-prepare state does not involve creating and verifying a BLS multi-signature. In the pre-prepare state the primary replica will broadcast an announcement message such as the proposal block to other secondary replicas. When a secondary replica receives an announce message, it enters the prepare state. In the prepare state, replicas will send a prepare message (e.g. signature on blockhash, public key and potentially other information to the primary replica. The primary replica aggregates signatures together into a multi-signature and replies to participating secondary replicas with the multi-signature, public keys and potentially other information. After validating the multi-signature and checking the multi-signature involves $\geq 2f + 1$ participants, secondary replicas will enter the commit state. In commit state, secondary replicas will send a commit message which is a signature to the primary replica with its public key and potentially other information. The primary replica aggregates signatures together into a multi-signature and replies to participating secondary replicas with the multi-signature, public keys and potentially information. After validating the multi-signature and checking the multi-signatures involving $\geq 2f + 1$ participants it will commit a block to the blockchain. Since all defense mechanisms involve different numbers of operations to run pre-prepare, prepare and commit states we need to time this process as the committee size grows and analyze the results.

Replica Storage Requirements

Efficient communication protocols should not require nodes to store massive amounts of information in order to communicate. Restricting storage overhead during communication is something that must be considered for the blockchains of the future. To incentivise more nodes to join the blockchain network you need to reduce the hardware requirements of becoming a node. The authors of [2] outlined how nodes which join the blockchain network have to maintain a local copy of the full blockchain which in Bitcoins case is around 360GB. This storage factor is a considerable hindrance to the blockchain network's scalability. Since this project is concerned with communication efficiency in sharded blockchains we analyze the storage overhead required for nodes to store the required information used to validate multi-signatures during communication. In our simulation we analyze the local storage requirements as the committee grows using each different rogue key defense.

Chapter 6

Evaluation & Analysis

Our study is concerned with optimizing communication efficiency in sharded blockchains. Sharding has proven to be a vital part of the future of blockchains so improving communication in sharded blockchains is also very important. To grade communication efficiency is we analyze the following metrics :

1. How big are the messages being sent between replicas in a committee
2. How long does it take a committee of replicas to complete a full round of consensus.
3. How much information has to be stored by each replica for communication validation.

Evaluating these metrics is important because it may outline important efficiency characteristics of communication protocols. Improving communication efficiency requires a holistic approach where is it important to improve all metrics together. We do not want to improve a metric at the expense of another because future sharded blockchains requires efficient messages, fast communication, reduced consensus speeds and low storage requirements.

6.1 Evaluation

In order to test our protocol changes we ran a number of simulations. The designed and built simulator recorded specific metrics that help deduce how efficient communication is. The simulator implements the two popular existing defense mechanisms *Proof-of-Possession* and *Distinct Messages* as well as the two proposed improved defense mechanisms *Leader Excluded* and *Public Key Cert Table* in the simulation. We can then analyze and evaluate the outlined metrics and compare the existing defense techniques against the improved proposed defense techniques. From this comparison we can deduce if we succeeded in designing new improved defense mechanisms for use with BLS multi-signatures during communication in sharded blockchains.

6.1.1 Message Size

We measure the size of messages sent from a) secondary replicas to the primary replica and from b) the primary replica to the secondary replicas. As we are analyzing metrics a) and b) we increase the number of nodes in the committee which will cause an increase in metric b). The increase in metric b) is due to the fact that as we add more replicas to the committee we increase the number of replicas participating in communication rounds. During a round of communication if a secondary replica approves of the current state in the PBFT state machine it will send its signature, public key, and other information to the primary replica. The primary replica aggregates all approving secondary replica signatures together into a BLS multi-signature and broadcasts a message containing the multi-signature, participating public keys and other information to all secondary replicas. As you can see from the above explanation of the communication process if we have more secondary replicas participating and sending approval messages to the primary replica we have more public keys and other information to include in the message from the primary replica to the secondary replicas.

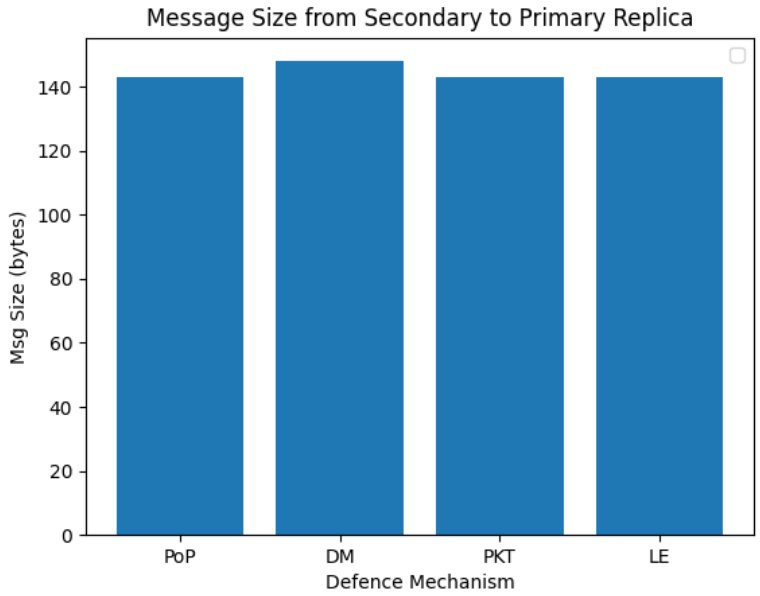


Figure 6.1: Secondary to Primary Replica Message Size

Metric a) which is the size of the message from the secondary to the primary replica is constant as shown in Figure 6.1. This is because a secondary replica will only include information about itself when messaging the primary. This information does not change as the number of secondary replicas in our committee increases. It is very hard to find a way to reduce Metric a) since very little information is included in the message.

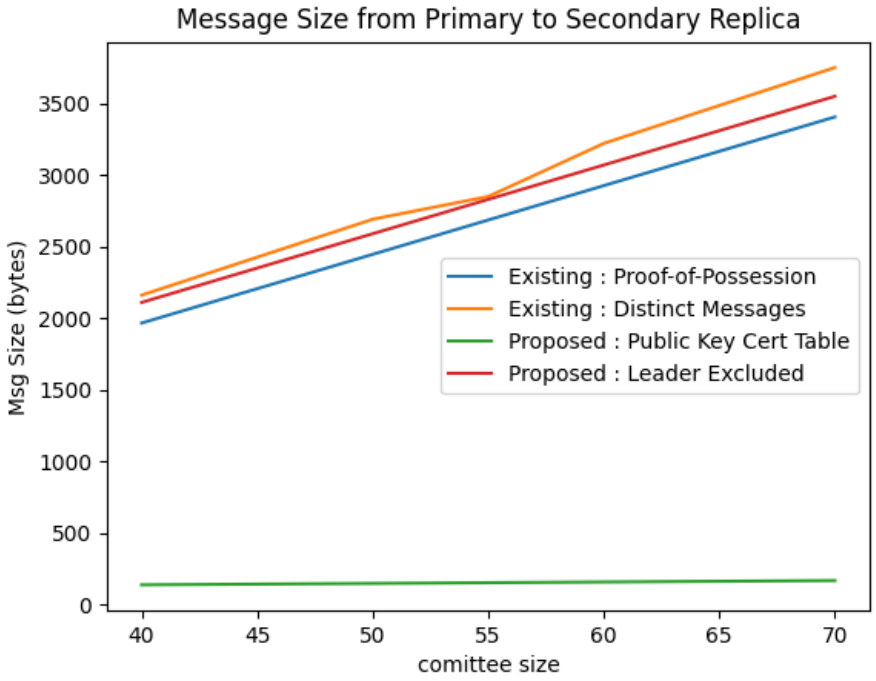


Figure 6.2: Primary to Secondary Replica Message Size

Metric b) which is the size of the message from the primary to secondary replica increases as the number of replicas in the committee increases as shown in Figure 6.2. Evaluating the message size from primary to secondary as committee size grows gives us much more useful insights into the scalability of communication using each existing and proposed defense mechanism.

Our proposed public key cert table has by far the most efficient message size from primary to secondary replicas. This is because we reduce the public key list which is used for verifying the BLS multi-signature to a simple bitmap. This means we reduce each 48 byte public key to 1 bit. Then the primary replica broadcasts the bitmap with the multi-signature to secondary replicas who cross-reference their locally stored public key cert table with the bitmap to determine which public keys are needed to verify the multi-signature. As outlined in Figure 6.2 the designed Public Key Cert Table defense has the greatest reduction in

message size from primary to secondary replicas showing a massive 97% reduction when compared to existing defenses such as Proof-of-Possession and Distinct Messages and the proposed Leader Excluded defense. This means when an optimised and intricate defense mechanism can be afforded for communication in sharded blockchains the proposed Public Key Cert Table design can replace all existing defenses.

Our proposed Leader Excluded protocol scales much better than the existing Distinct Messaging protocol. Leader Excluded and Distinct Message defenses are significantly easier to integrate than the existing Proof-of-Possession defense and the proposed Public Key Cert Table defense. Leader Excluded defense is also significantly faster consensus speed than Distinct Message defense offering a $70\% \rightarrow 80\%$ reduction. This means when a easy to implement defense mechanism is needed for communication in sharded blockchains the proposed Leader Excluded design can replace the existing Distinct Message defense.

6.1.2 Consensus Latency

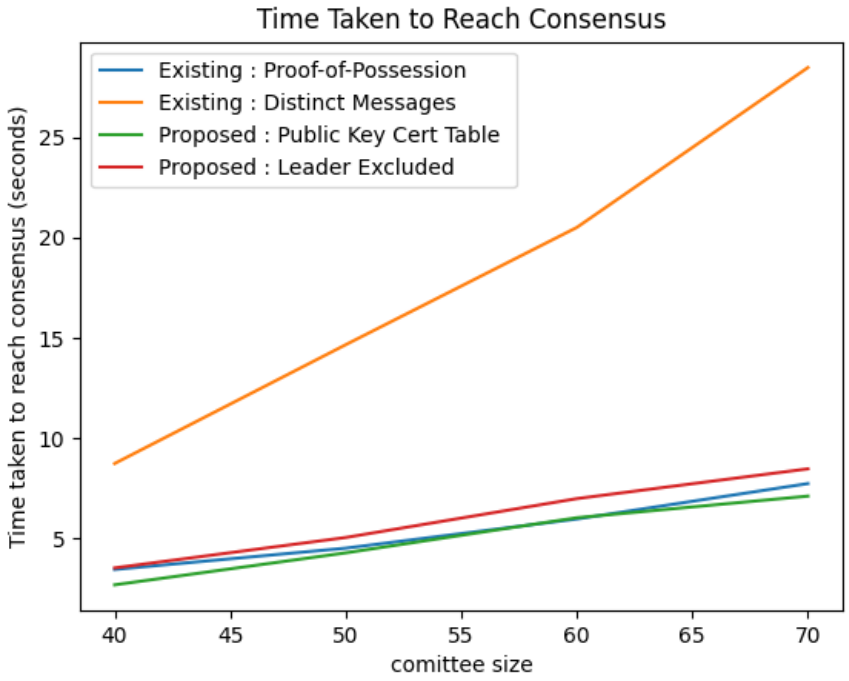


Figure 6.3: Consensus Latency

Consensus latency represents how long it takes a committee of replicas to complete one full round of consensus using BLS multi-signatures with existing defense mechanisms and the proposed defense mechanisms. As shown in Figure 6.3, in general, as committee size increases so does the time taken to reach consensus. This is because as more secondary replicas are added to the committee there will be more replicas participating in the consensus round. As a result consensus rounds require more information to be transmitted and verified between replicas. There are two ways to reduce consensus latency during communication in sharded blockchains. Method a) is by using the fast multi-signature verification method by requiring all replicas to sign the same message and method b) is by optimising the information transmission process. In Figure 6.3 we can see how poorly the existing Distinct Message defense scales as more replicas are added to the committee. This is due

to the fact that Distinct Message defense is the only defense which does not use the fast multi-signature verification method. Proof-of-Possession, Distinct Messages and Leader Excluded defense all use the fast multi-signature verification method and as you can see they all scale similarly in the consensus latency graph however the Public Key Cert Table defense scales the best by a small margin.

6.1.3 Replica Storage Requirements

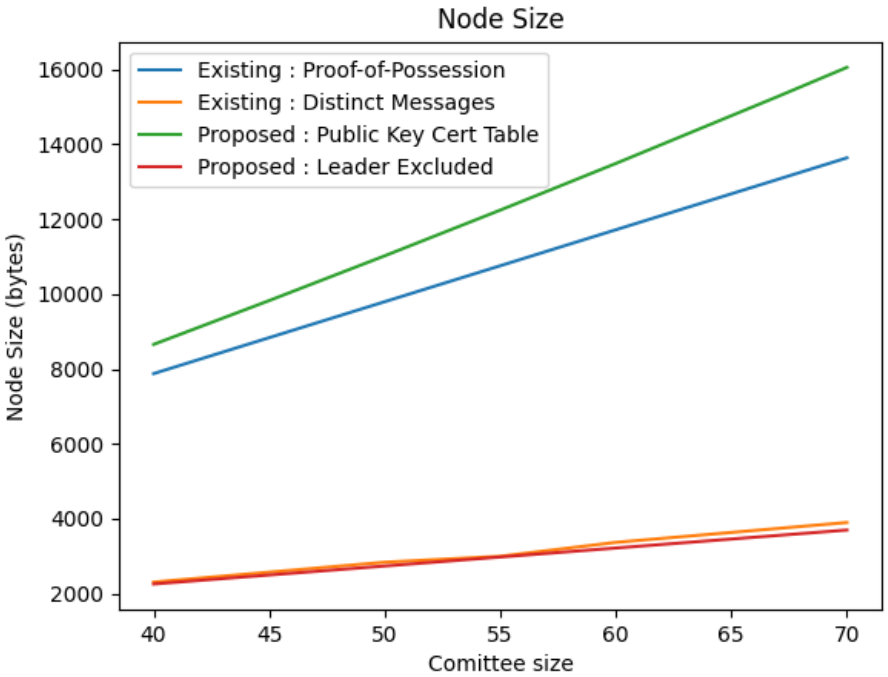


Figure 6.4: Node Storage Requirements

Figure 6.4 displays how Distinct Message and Leader Excluded defense have very similar storage overheads. This is an important benefit for our proposed Leader Excluded protocol since the existing Distinct Message defense is considered the “lightweight”, low storage overhead solution. The proposed Public Key Cert Table solution has a worse storage overhead when compared to the existing Proof-of-Possession defense. This is the only

drawback of the Public Key Cert Table defense and is caused by the extra storage required by each replica to store the public key certificate for every other replica in the committee.

6.2 Analysis

In this section, we give a theoretical analysis of the security, scalability, and feasibility of each existing and proposed defense mechanisms which are used during communication in a sharded blockchain.

6.2.1 Security Analysis

For the security analysis we start off by stating all existing and proposed, improved defense mechanisms are rogue key resistant. Rogue key resistance is outlined for existing defense mechanisms in Chapter 3 and for our proposed defense mechanisms in chapter 4. We also must ensure that BLS multi-signatures are being used correctly. This is defined as the *safety* property of communication using BLS multi-signatures with each defense mechanism. Safety indicates inability for a malicious node or group of nodes M to hijack a committee and maliciously use multi-signatures. Let f equal the maximum number of malicious replicas in a committee.

All existing and proposed defense mechanisms achieve *safety* if there are less than f malicious replicas in our committee. Assuming there are less than f malicious replicas in our committee we guarantee either valid communication and consensus will occur or no round of consensus will occur. We know each round of communication is honest because each round of communication requires the supermajority ($2 \cdot f + 1$) to participate in the round of communication. We know our BLS multi-signature must be honest because atleast a majority of the signatures that are aggregated together into the multi-signature are from honest replicas. Therefore we know multi-signature will be created and used safely.

6.2.2 Scalability Analysis

For our scalability analysis of the communication protocol we mainly pay attention to how we reduce the message size from primary replicas to secondary replicas and how long it takes to reach consensus. Analyzing consensus speed can be vaguely estimated without a realistic simulation by checking what multi-signature verification algorithm is used. If fast verification is not used it will take significantly longer to verify a multi-signature and reach consensus compared to if fast verification is used.

Symbols Definitions

Variable Definitions
X = Elliptical operation to message sign
Y = Elliptical operation to message verify
Z = Multi-Signature creation
Q = Multi-Signature verification
N = Number of replicas in committee

Algorithmic Scalability Analysis

To fit the table on the page we write table entries in shorthand. **Msg** means message, **RK** means Rogue Key, **Fast-Verify** means can this defense use the fast multi-signature verification that only involves two elliptical operations, **PKCT** means Public Key Cert Table, **LE** means Leader Excluded, **POP** means Proof-of-Possession and DM means Distinct Message. **P** means primary replica and **S** means secondary replica.

6.2.3 Feasibility Analysis

We can group the feasibility of each defense mechanism into two types. Type A requires no changes outside of the communication protocol. Type A defenses only requires changes within the communication protocol such as information included in messages and how we authenticate messages. These defenses are very straightforward to implement in sharded

Name	Msg Size (S to P)	Msg Size (P to S)	Operations	RK Safe	Fast-Verify
POP	144 Bytes	$N \cdot 46 + 96$ Bytes	X+Y+Z+(2*Q)	Yes	Yes
DM	148 Bytes	$N \cdot 144 + 96$ Bytes	X+Y+Z+(N*Q)	Yes	No
PKCT	144 Bytes	$N \cdot 1$Bit$+96$ Bytes	X+Y+Z+(2*Q)	Yes	Yes
LE	144 Bytes	$N \cdot 46 + 240$ Bytes	X+2Y+Z+(N*Q)	Yes	Yes

blockchain networks due to their limited scope. Implementing these defenses usually requires no upstream changes. Type A defenses include Distinct Message and Leader Excluded defense. Type B defenses involve changes outside of the communication protocol. These defenses could involve changes to the information stored in nodes, adding or altering data stored in the blockchain, creating auxiliary services that provide functionality to our main blockchain and so on. Type B defenses include Proof-of-Possession and Public Key Cert Table defenses. Type B defenses take much more effort, create additional complexity and require more caution to implement so further security vulnerabilities aren't introduced.

We can deduce that both our proposed defenses offer valuable improvements over existing defenses. When a sharded blockchain requires BLS multi-signatures with a quick and simple defense against rogue key attacks it can implement our proposed Leader Excluded defense instead of traditionally implementing the existing Distinct Message defense. This is because our proposed Leader Excluded defense offers a $60\% \rightarrow 70\%$ reduction in time taken to reach consensus and a slight reduction in message size from primary to secondary replicas when compared to the alternative existing Distinct Message defense. When a sharded blockchain wants to implement a more efficient and scalable defense with more avenues for information security and verification it can implement our proposed public key cert table solution instead of traditionally implementing the existing proof-of-possession

protocol. This is because the Public Key Cert Table defense offers the fastest time taken to reach consensus and offers a substantial 97% reduction in message size from primary to secondary replicas when compared to the all alternative defense mechanisms.

Chapter 7

Conclusion

An in depth analysis of communication in sharded blockchains using the PBFT protocol is presented. Important sharded blockchain design and network architecture concepts were then outlined and contextualized. We then outline how by integrating BLS multi-signatures we get significantly more scalable communication. Preventing BLS multi-signatures only non-trivial vulnerability which is the Rogue Key attack is mandatory when discussing BLS multi-signatures so a detailed explanation of the attack as-well as an in-depth analysis of the existing defense mechanisms against the Rogue Key attack is provided.

Several very important and meaningful contributions are then made to the blockchain space. The first contributions are two new improved Rogue Key defense called the Leader Excluded and Public Key Cert Table defense mechanisms. We outline how the proposed Leader Excluded defense can be used as a quick and easy defense against the Rogue Key attack and can be considered an alternative to the existing Distinct Message defense. Algorithmic and simulated results are then provided to show how our proposed Leader Excluded defense takes 50% less time to reach consensus and reduced the message size by 10%. The second proposed defense called Public Key Cert Table defense incorporated public key infrastructure technology such as public key certificates into the defense mechanism. This concept,which has never been explored before, offers multiple new avenues for secure and

verifiable information storage for blockchain networks. We analyze the efficiency of the proposed Public Key Cert Table defense. It was shown how it took around 10% less time than the existing Proof-of-Possession defense (which is the fastest existing defense mechanism) and a massive 97% reduction in the message size required during communication. It was then outlined why our proposed Public Key Cert Table defense can be used as an alternative to the existing Proof-of-Possession defense when a blockchain requires a more in-depth, efficient solution.

Finally we contribute a realistic simulation of the PBFT block consensus process that is run by a committee of replicas in a sharded blockchain in order to agree on what block to commit in the blockchain. The simulation uses BLS multi-signatures with existing defense mechanism and our proposed improved defense mechanisms. From our simulation we analyze how the message size, the time taken to reach consensus and the replica storage requirements changes as more replicas are added to the committee. The proposed simulation is another great contribution as it provides an extensible and realistic simulation and testing framework for the future of BLS multi-signatures being used in communication in sharded blockchains.

Bibliography

[1] Sherif Arora et al Akshita Jain. “Proof of Stake with Casper.” In: (2018) (cit. on p. 7).

[2] P. Chen et al. “Ladder A Blockchain Model of Low-Overhead Storage.” In: *Blockchain and Trustworthy Systems* 1490 (2021), pp. 116–129 (cit. on p. 49).

[3] Frederik Armknecht et al. “Sharding PoW-based Blockchains via Proofs of Knowledge.” In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 1067.

[4] Shacham Boneh Lynnn. “Aggregate and Verifiably Encrypted Signatures from Bilinear Maps.” In: (2003) (cit. on p. i).

[5] Miguel Castro. “Practical Byzantine Fault Tolerance.” In: (2001) (cit. on p. 11).

[6] Gregory Neven Dan Boneh Manu Drijvers. “BLS Multi-Signatures With Public-Key Aggregation.” In: (March 24, 2018).

[7] Gregory Neven Dan Boneh Manu Drijvers. “Compact Multi-Signatures for Smaller Blockchains.” In: (2021).

[8] P. Jovanovic et al E. Kokoris-Kogias. “Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing.” In: *25th USENIX Conference on Security Symposium* (2016) (cit. on p. 16).

[9] L.Gasser et al E.Kokoris-Kogias P.Jovanovic. “Omniledger: A secure, scale-out, decentralized ledger via sharding.” In: (2018) (cit. on p. 11).

[10] Ethereum Research Foundation. “Pragmatic signature aggregation with BLS.” In: (2018), pp. 1–7 (cit. on p. 10).

[11] Yannick Seurin et al Gregory Maxwell Andrew Poelstra. “Simple Schnorr Multi-Signatures w/ Applications to Bitcoin.” In: *Blockstream* 87 (2019), pp. 2139–2164 (cit. on p. 15).

[12] D.Malkhi et al I.Abraham. “Solida: A blockchain protocol based on re-configurable byzantine consensus.” In: (2017) (cit. on p. 11).

[13] Ricki G. Ingall. “Introduction to Simulation.” In: *Winter Simulation Conference* (2011) (cit. on p. 43).

[14] K. Itakura and K. Nakamura. “A public-key cryptosystem suitable for digital multisignatures.” In: *NEC Research and Development* (1983) (cit. on p. 15).

[15] Eleftherios Kokoris-Kogias. “Robust and Scalable Consensus for Sharded Distributed Ledgers.” In: (2019) (cit. on p. 33).

[16] Marie-Sarah Lacharité. “Security of BLS and BGLS signatures in a multi-user setting.” In: *2021 IEEE Symposium on Computers and Communications (ISCC)* (2021), pp. 1–7.

[17] Mint Layer. “Mintlayer Adoption of Bls Signature.” In: *Mint Layer Research Journal* (2017).

[18] Yanan Liu et al. “OverlapShard: Overlap-based Sharding Mechanism.” In: *2021 IEEE Symposium on Computers and Communications (ISCC)* (2021), pp. 1–7.

[19] B.Liskov M.Castro. “Practical Byzantine Fault Tolerance.” In: (1999) (cit. on p. 10).

[20] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System.” In: (2008) (cit. on p. 7).

[21] Harmony Team. “Harmony Technical Whitepaper.” In: (2019) (cit. on p. 16).

[22] Scott Yilek Thomas Ristenpart. “The power of proofs-of-possession: Securing multi-party signatures against rogue-key attacks.” In: *EUROCRYPT* 4515 (2007), pp. 228–245.