Optimized Communication in Sharded Blockchain Networks

James Rowland[1] and Hitesh Tewari[1]

Trinity College Dublin, Ireland
{rowlanja, htewari}@tcd.ie

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 such that each shard can process transactions in parallel. In this paper we explore and improve the communication process that happens between nodes in a shard by making use of multi-signatures. Furthermore, a realistic simulation of the communication process between nodes in a shard using multi-signatures is implemented. This simulation is used to analyze and evaluate the performance of the different sharded network communication implementations.

Keywords: Sharding, Multi-Signatures, Public-Key Certificates, PKI, Blockchain

1 Introduction

Blockchains have proven to be highly adaptable across many important types of human interactions. A result of the growing list of use cases for decentralized blockchains is the necessity for scalable transaction processing. Scalable transaction processing was not emphasized in the design of early blockchains such as Bitcoin where security and decentralization were prioritized. New approaches are now being 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. Sharding also introduces new pitfalls one needs to be aware of. One such pitfall is efficient communication between nodes in a shard.

1.1 Sharding

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 [8] 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". Nodes are grouped into subsets that process a distinct portion of transactions for the blockchain, which

results in parallelizing the transaction processing within the blockchain. As transaction load increases a sharded blockchain network can scale out the system by increasing the number of shards or scale up the system by increase the nodes per shard.

A shard will contain a self-governing, self-sufficient group of nodes called a *committee* that commits blocks to the blockchain. For the committee to reach consensus the byzantine fault tolerance (BFT) state machine replication algorithm is executed in each node in the committee. Practical byzantine fault tolerance (PBFT) [11] is the name of the BFT variant used in sharded blockchains to reach consensus given adversarial conditions. 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 a large majority of replicas, allowing it to proceed to the next state in the PBFT state protocol.

Transaction Validation In sharded blockchains both *proof-of-x* (PoX) and PBFT work together to validate transactions and to add processed transactions to the blockchain. PoX protocols are typically used to verify transactions and usually require effort (Proof-of-Work in Bitcoin [9]) or resources (Proof-of-Stake in Ethereum [10]) 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 be 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 transactions. In PoS, nodes are more likely to propose new blocks if they have more votes.

Committee Consensus After blocks are validated most sharded blockchains use the PBFT state machine protocol to reach consensus on actions such as which block to include in the blockchain next. 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 faulty replicas in the committee. Faulty replicas are defined as replicas 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. Secondary replicas are responsible for communicating whether they approve of the current PBFT state that their replication of the PBFT state machine is in.

To reach consensus all replicas run an agreement process involving three PBFT states (*pre-prepare*, *prepare*, *commit*) as shown in Figure 1, which lets

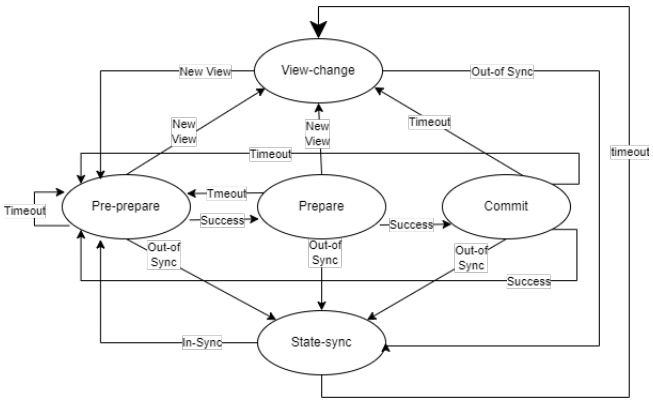

Fig. 1. Simple PBFT State Machine Diagram

the committee of replicas agree on the block to include in the blockchain. PBFT consensus accounts for all error types by including specialized error-handling states or actions in the PBFT state machine.

Shard Scalability PBFT consensus scalability is limited from the proportional relationship between committee size and communication overhead. When more replicas are added to the committee, more communication is required to reach consensus. PBFT adopts the use of *quorums* which are intersections of subsets of replicas in 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 committees. 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. As stated by the Ethereum Foundation [12], 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.

2 Multi-Signatures

Multi-signatures were first introduced in [5] where participants could create signatures by using public and private key encryption with a message. 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.

2.1 BLS Multi-Signature

Due to the compactness, scalability and fast verification processes available, the BLS multi-signature scheme [2] [7] has proven much more suitable for use during communication in sharded blockchains. Integration of BLS multi-signatures into communication with PBFT was explored in a sharded blockchain design outlined in [4]. The new BLS multi-signature integrated PBFT protocol was labeled fast byzantine fault tolerance (FBFT). The following elaboration of BLS multi-signatures will refer to symbols defined in table 1.

Table 1. Symbol Lookup Table.

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.

2.2 BLS Signature Creation and 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 with the accompanying public key PK and message M using an elliptical curve operation:

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

The equation above verifies a BLS signature by checking 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.

2.3 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 can 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:

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

2.4 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 Rogue Key Attack

The rogue key attack (RKA) is BLS multi-signatures' only non-trivial attack. The following elaboration of the rogue key attack will refer to symbols defined in table 2. 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 RKA can only be mounted when replicas sign the same message ($M_1 = ... = M_n$).

Table 2. Symbol Lookup Table.

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 RKA, where an attacker calculates a rogue key (RK):

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

3.1 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 :

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

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:

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

4 Defense Mechanisms

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

4.1 Distinct Messages

We can require that all the messages being signed are distinct and unguessable. 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 (DM) to all replicas in the committee. Each replica then verifies the multi-signature using the public keys and associated 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 do not aggregate the public keys together into a single public key to verify the multi-signature. A rogue key simply does not exist.

DM defense is provably secure against the RKA. Since we do not aggregate the public keys together, an adversary cannot forge a rogue key to publish as its own public key. The DM defense is simple to implement. The only requirement is that the message being signed by the replicas must be distinct to prevent malicious attacks. The disadvantage for DM defense is that 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.

4.2 Proof-of-Possession

Proof-of-possesion (PoP) [3] prevents rogue key attacks 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. As seen in Figure 2, in *State-Sync*, 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 that can participate within the committee. 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 RKA.

The advantage of PoP is that we can allow replicas to sign the same publicly available message. When all the messages being signed are the same the verification equation reduces to a simpler test that requires only two pairings. This significantly increases communication scalability within the committee. The disadvantage is that 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 PoP proof. To satisfy this assumption, functionality that maintains the primary and secondary replicas list of verified public keys must be in place

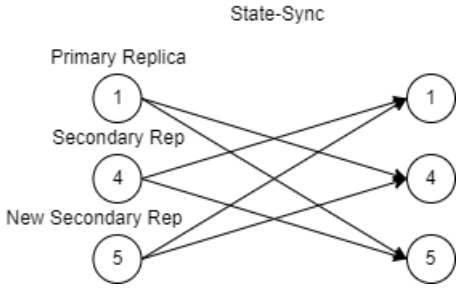

Fig. 2. State Sync with Proof-of-Possession

5 Proposed Defense Mechanism

In this section we outline an improved defense design that prevents RKAs on BLS multi-signatures used during communication in sharded blockchains. The improved design is called the *Public Key Cert Table* protocol, which focuses on optimizing existing defenses and utilising new technologies to prevent the RKA.

5.1 Public Key Cert Table

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. In the public key cert table (PKCT) protocol we explore how we can use PKI technology to provide security against RKAs, reduce the message size, and to reduce the time it takes to reach consensus.

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 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.

By incorporating public-key certs we open potential avenues for additional security. We know the information stored in the public-key certs is legitimate and authentic. We can also use the locally stored public key cert table to prevent man-in-the-middle attacks. We can prevent RKAs on BLS multi-signatures by employing a PKI. This is because PKI technology offers public-key certificate management, user identity validation, secure user certificate storage, and auxiliary user information storage.

5.2 Protocol Details

In the PKCT 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 (i.e. to become a replica) it sends its information to the CA for certificate creation. The CA will create a certificate with the node's information and store the certificate on the blockchain so that the certificate 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. This way the node can send the certificate reference to other nodes. Figure 3 displays how public key certificates are stored in the blockchain:

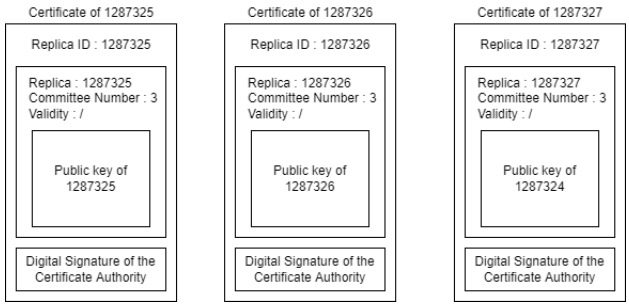

Fig. 3. Certs Stored on Blockchain

In our PKCT solution every replica must have a table containing the public-key cert of each replica in the committee. To create our public-key cert table we can add a step to the *State-Sync*, 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 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 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

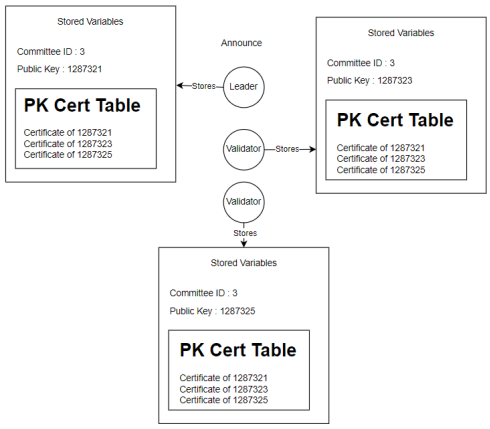

Fig. 4. Cert Table Cached in Each Replica

to remove outdated certificates. The result will be that each replica now has 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. 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 ineffective.

All replicas can now reference the locally cached public-key certificate table as shown in Figure 4, during multi-signature creation and verification. When a primary replica broadcasts a BLS multi-signature to all secondary replicas, it will include a bitmap that references participating public keys in the public-key cert table as shown in Figure 5.

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 6, 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.

Since we have prevented malicious replicas mounting a RKA we can allow replicas to sign a publicly available, shared message. 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 [6] 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 PKCT solution we use the same idea of implementing bitmaps to significantly

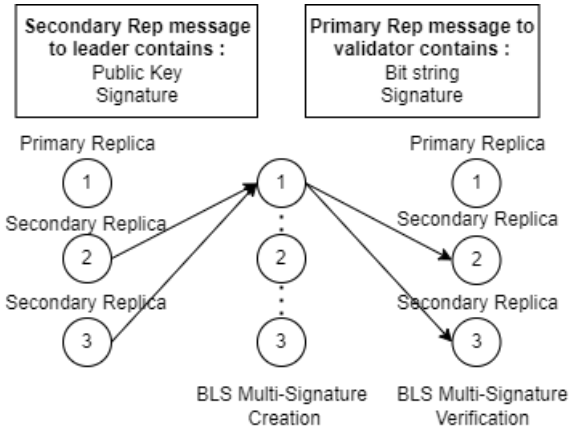

Fig. 5. PBFT Communication using Certs

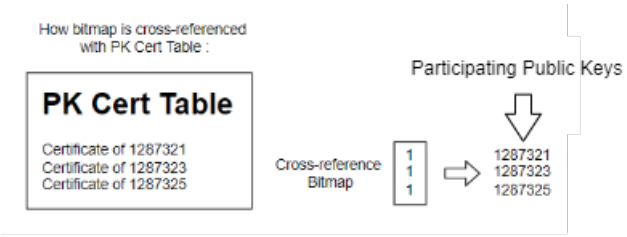

Fig. 6. Bitmap Use Case

increase communication efficiency. We do not require a trusted leader. The advantage of the PKCT protocol is that it is significantly more efficient than any existing BLS-aggregation defense protocol. 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. Since the PKCT protocol is secure against RKA, we can allow 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. The main disadvantage of the PKCT protocol is that implementation requires an overhaul of the blockchain system. We cannot implement the solution without implementing the required systems to create, store, retrieve and verify the public key certs.

Security

This defense solution has the same approach to securing against RKAs as PoP. We prevent an adversary calculating and including a rogue key by restricting the public keys that can be used while verifying the multi-signature. This defense solution also opens new avenues for information security on the blockchain.

Preventing Man-in-the-Middle Attacks

In order to maintain secure multi-signature verification we need to ensure a man-in-the-middle (MITM) attacker cannot alter the bitmap during transmission from the primary replica to secondary replicas. To detect a MITM attack we implement the following protocol. Since all replicas know what the current block number is meant to be we can use the block number as a *checksum*. In our proposed PKCT defense, the primary replica will create a new secure bitmap by concatenating the block number onto the bitmap, and then signing 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.

6 Simulation

Simulating the communication process that occurs between replicas using multi-signatures with the two most popular existing defense mechanisms PoP and DM, and the proposed defense mechanism PKCT exposes important scalability characteristics.

In the simulation, a committee moves from one PBFT state to another (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) as outlined in [1]. A DES is defined as "the process of codifying the behavior of a complex system as an ordered sequence of well-defined events". 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 simulation of the dynamic system.

To reflect the realistic behavior of an actual sharded blockchain network, the appropriate 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 involved socket programming and threading.

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 and verification. The library is provided by Chia-networks and is utilized by existing decentralised blockchains.

6.1 Measured Statistics

Since we are focused on optimizing communication in sharded blockchains we carefully select what metrics we track and analyze:

- **Message Size from Secondary Replica to Primary Replica**: A secondary replica sends its signature, public key and additional information depending on which defense mechanism to the primary replica. They send this information during the prepare and commit state of PBFT.
- **Message Size from Primary Replica to Secondary Replica**: A primary replica sends the multi-signature, the list of participating public keys and additional information, depending on the defense mechanism when messaging secondary replicas.
- **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.
- **Replica Storage Requirements**: Replica storage requirements represents how much space a replica requires to store the information used to validate the multi-signature during communication rounds.

7 Evaluation

This paper is concerned with optimizing communication efficiency in sharded blockchains. To grade communication implementations we analyze the metrics stated in Section 6.1. Evaluating these metrics outlines important efficiency and scalability characteristics of communication protocols. In order to test our protocol changes we ran a number of simulations. The simulator recorded specific metrics that help deduce how efficient communication is. The simulator implements the two popular existing defense mechanisms PoP and DM, as well as the proposed improved defense mechanism PKCT in the simulation. We analyze and evaluate the outlined metrics and compare the existing defense techniques against our improved proposed defense technique.

7.1 Message Size

We measure the size of messages sent between replicas as we increase the number of nodes in the committee.

Figure 7 shows the size of the message sent from the secondary to the primary replica is constant. This is because a secondary replica will only include information about itself when messaging the primary.

Figure 8 shows the size of the message sent from the primary to secondary replica increases as the number of replicas in the committee increases. This metric increases due to the fact that as we add more replicas to the committee, we increase the number of replicas participating in communication rounds. As a result, the primary replica must include more public keys and other information in the message from the primary replica to the secondary replicas.

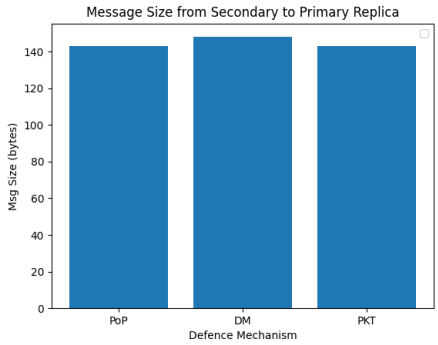

Fig. 7. Secondary to Primary Replica Message Size

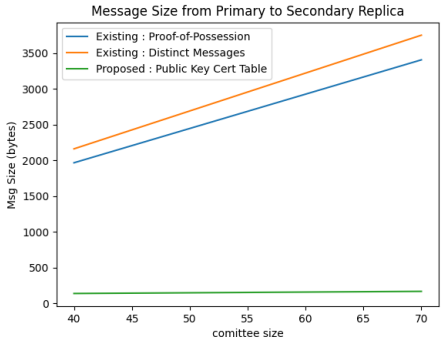

Fig. 8. Primary to Secondary Replica Message Size

Evaluating the message size from primary replica to secondary replicas as the committee size grows gives us useful insights into the scalability of communication using each existing and proposed defense mechanism. Our proposed PKCT defense has the most efficient message size from primary to secondary replicas compared to any other defense mechanism. As outlined in Figure 8, the proposed PKCT defense mechanism reduces message size from primary to secondary replicas by 97% when compared to all other defenses.

7.2 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 9, the time taken to reach consensus increases as the committee size increases because more information needs to be transmitted and verified.

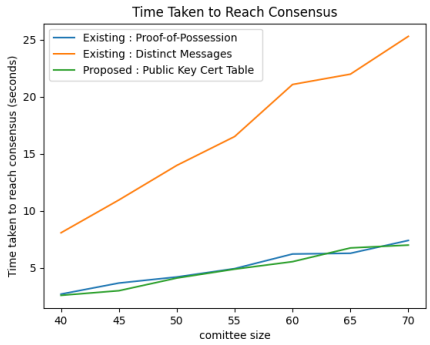

Fig. 9. Consensus Latency

Figure 9 shows how the consensus latency for existing DM defense scales poorly when more replicas are added to the committee. This is due to the fact that DM defense is the only defense which does not use the fast multi-signature verification method. PoP and PKCT defense all use the fast multi-signature verification method. It must be noted that the proposed PKCT scales the best in reducing consensus latency, however, only by a small margin.

7.3 Replica Storage Requirements

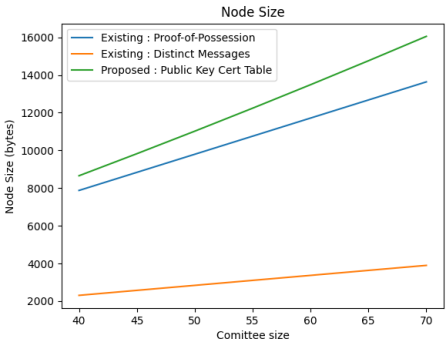

Fig. 10. Node Storage Requirements

Figure 10 displays how the proposed PKCT solution has a worse storage overhead when compared to the existing PoP defense. This is the only drawback of the PKCT defense. This drawback is caused by the extra storage required by each replica to store the public key certificates for each replica in the committee.

8 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.

8.1 Security Analysis

For the security analysis it must be stated all existing and proposed defense mechanisms are RKA resistant. 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 at least 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.

8.2 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 scalability 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. PoP and PKCT defenses use fast multi-signature verification and DM uses normal multi-signature verification. Message size scalability is approximately similar for PoP and DM. The scalability of message size from primary replica to secondary replicas for PKCT defense is significantly improved by reducing the public key list to a simple bitmap. Replica storage requirements scales the best for DM defense and the worst for PKCT defense due to the PKCT defense requiring local storage of replica public key certificates.

8.3 Feasibility Analysis

We can group the feasibility of each defense mechanism into two types. Type A restricts changes to within the communication protocol such as message information and message authentication. Type A defenses include the DM 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 and more. Type B defenses include PoP and PKCT defenses. Type B defenses take more effort, create additional complexity and require more caution to implement so vulnerabilities are not introduced. We can deduce that the proposed PKCT defense offers valuable improvements over existing defenses. Due to PKCT's reduced message size and minimal consensus latency the proposed PKCT defense is recommended when a sharded blockchain can feasibly implement a more efficient, scalable and secure defense.

9 Conclusion

In this paper we presented an improved RKA defense mechanism. We outlined how the proposed PKCT defense mechanism offers multiple new avenues for secure and verifiable information storage for blockchain networks. We then outlined why our proposed mechanism can be used as an alternative to the existing PoP defense. Finally we contributed 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.

References

1. Introduction to Simulation , Ricki G. Ingall, Winter Simulation Conference, (2011)
2. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps,Boneh, Lynnn, Shacham (2003),
3. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks, Thomas Ristenpart, Scott Yilek, EUROCRYPT, (2007)
4. Harmony Technical Whitepaper,Harmony Team, (2019),
5. A public-key cryptosystem suitable for digital multisignatures,K. Itakura and K. Nakamura, NEC Research and Development, (1983),
6. Robust and Scalable Consensus for Sharded Distributed Ledgers, Eleftherios Kokoris-Kogias, (2019),
7. BLS Multi-Signatures With Public-Key Aggregation, Dan Boneh,Manu Drijvers, Gregory Neven, (March 24, 2018),
8. Omniledger: A secure, scale-out, decentralized ledger via sharding, E.Kokoris-Kogias, P.Jovanovic, L.Gasser et al, (2018)
9. Bitcoin: A Peer-to-Peer Electronic Cash System, Satoshi Nakamoto,(2008)
10. Proof of Stake with Casper, Akshita Jain , Sherif Arora et al, (2018)
11. Practical Byzantine Fault Tolerance, M.Castro,B.Liskov, (1999)
12. Ethereum Research Foundation, Pragmatic signature aggregation with BLS, (2018)