UNIVERSITY OF CALIFORNIA
SANTA CRUZ

PAVE: PRIVACY-PRESERVING AGGREGATED VERIFICATION FOR ENTERPRISES

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE AND ENGINEERING

by

Siaraie Tesfai

December 2024

The Thesis of Siaraie Tesfai
is approved:

Table of Contents

List of Figures

List of Tables

Abstract

PAVE: PRIVACY-PRESERVING AGGREGATED VERIFICATION FOR ENTERPRISES

by

Siaraie Tesfai

The rise of multi-enterprise applications in fields like supply chain management, finance, and healthcare has accelerated the adoption of permissioned blockchain systems. However, many existing systems lack support for privacy-preserving verifiability, creating challenges in scenarios where data and asset ownership must be validated across enterprises without revealing sensitive details. This paper presents PAVE (Privacy-preserving Aggregated Verification for Enterprises), a framework that integrates zero-knowledge proofs (ZKPs) to enable secure and efficient verification of data and assets. To further optimize computational efficiency, PAVE introduces a proof aggregation mechanism, allowing multiple transactions involving the same asset to be verified through a single, consolidated proof. This reduces the overhead associated with generating and verifying individual proofs, providing a scalable solution for high-volume transactions. Our approach demonstrates that integrating ZKPs with aggregation capabilities not only strengthens privacy and security but also enhances scalability, making PAVE suitable for asset-centric, high-throughput applications across various sectors.

Acknowledgments

I am deeply grateful to Professor Xiaoxue Zhang for her invaluable guidance and supervision throughout the development of this paper. Her insights and expertise have been instrumental in shaping the direction and depth of this work, greatly contributing to the results we present here. I would also like to extend my sincere thanks to my advisor, Professor Chen Qian, for providing me with the opportunity to work in his lab. His support and mentorship has significantly improved my academic and professional growth.

I am also very appreciative of the contributions of my reading committee members, Professor Liting Hu and Professor Ram Sundara Raman, who took time from their busy schedules to review and provide thoughtful feedback on this paper. Their input has helped refine and strengthen this work, and their dedication to their roles is sincerely appreciated.

Finally, I would like to acknowledge the Hyperledger Fabric community for their comprehensive tutorials and detailed documentation. The clarity and accessibility of these resources greatly facilitated our development process and allowed us to effectively implement the solution presented here. Thank you to everyone who contributed to this project's success.

Part I

First Part

Chapter 1

Introduction

The growing demand for multi-enterprise applications, such as supply chain management, financial services, and healthcare, has driven the adoption of permissioned blockchain systems. While these systems address issues of scalability and confidentiality, they often lack mechanisms for privacy-preserving verifiability. In scenarios where an enterprise needs to verify data or assets held by another party, traditional solutions require exposing sensitive details, which can compromise trust and security.

For instance, imagine two financial institutions engaged in a cross-border trade transaction. Institution A must prove to Institution B that it holds sufficient funds or collateral to settle the transaction. Using traditional blockchain approaches, this proof may expose unnecessary details about Institution A's other assets or account balances, leading to potential breaches of confidentiality. Such limitations hinder trust and collaboration in sensitive, high-stakes environments.

To address this gap, we introduce PAVE (Privacy-preserving Aggregated Verification for Enterprises), a blockchain framework designed to enable secure and efficient verification of data and assets without compromising privacy. By leveraging zero-knowledge proofs (ZKPs), PAVE ensures that enterprises can validate claims about asset ownership or quantity while maintaining strict confidentiality.

To further enhance performance, PAVE incorporates a proof aggregation mechanism that allows multiple transactions involving the same asset to be verified using a single proof. This optimization is especially valuable in high-volume scenarios where multiple actions on the same asset occur within a short time frame. By reducing the computational overhead of generating and verifying individual proofs, PAVE improves both efficiency and scalability while maintaining robust confidentiality and consistency guarantees.

PAVE is evaluated through simulations and performance benchmarks, demonstrating its efficiency. The evaluation includes latency measurements for proof generation and verification, as well as the benefits of proof aggregation in reducing storage and processing demands. These results highlight PAVE's ability to provide scalable, privacy-preserving solutions for supply-chain systems.

In summary this paper provides the following contributions to supply-chain issues, which are highlighted below:

- **Integration of Zero-Knowledge Proofs (ZKPs)**: The work incorporates zk-SNARKs into a supply chain blockchain system to enable privacy-preserving verifiability. This ensures that enterprises can validate asset ownership and transactions without revealing sensitive data.

- **Proof Aggregation Mechanism**: A proof aggregation method is proposed, enabling multiple transactions involving the same asset to be verified using a single consolidated proof. This reduces computational overhead and enhances scalability in high-volume transaction scenarios.

This paper is organized as follows. Related works in section 2, background into technologies used and supply chain issues in section 3, overview on the attacker model assumed and the system in section 4, details in the design and security analysis of the system in section 5, evaluation on the solution compared to baselines and optimizations in section 6, and finally the conclusion in section 7.

Chapter 2

Related Work

Qanaat [8] is a scalable, multi-enterprise permissioned blockchain system that addresses confidentiality and scalability challenges in workflows involving multiple organizations. It uses a hierarchical data model with public, private, and intermediate data collections to enable selective data sharing, ensuring confidential collaborations remain hidden from non-involved parties. A privacy firewall mechanism separates ordering nodes from execution nodes to prevent data leakage, even in the presence of Byzantine failures. Qanaat supports both intra- and cross-shard transactions for high-volume scalability and overcomes performance bottlenecks found in systems like Hyperledger Fabric, making it ideal for applications such as supply chain management and healthcare.

Zk-Bridge [27] is a trustless cross-chain communication protocol that leverages zero-knowledge proofs (ZKPs) to enable secure and efficient interoperability between different blockchain networks. By using ZKPs, zk-Bridge ensures that data transferred across chains remains private and verifiable without relying on trusted intermediaries.

This approach enhances security and reduces the complexity of cross-chain interactions by providing cryptographic guarantees of correctness. zk-Bridge is designed to handle cross-chain transactions with minimal overhead, making it highly scalable and well-suited for decentralized finance (DeFi), gaming, and other blockchain applications requiring secure multi-chain interactions.

The solution that relates to cross-chain data sharing explores verifiability in cross-chain sharing by using Merkle trees to confirm data integrity and ensure trustworthy interactions across blockchain networks [25]. By leveraging Merkle tree properties, the proposed technique verifies data without a central authority, preserving data authenticity during exchanges.

CubeChain [21] proposes a scalable framework for verifiable, authenticated queries within and across blockchain networks, addressing data isolation in separate chains. Using a Directed Acyclic Graph structure and two-layer hashing, CubeChain organizes blocks from different chains as vertices linked by edges, allowing it to aggregate data from multiple chains into a unified queryable structure. Aggregation is achieved through data bridges that connect related vertices across chains, which groups data for efficient retrieval. By using intra- and inter-gap hashing, CubeChain enables lightweight verification of aggregated data, allowing users to access verified cross-chain data without holding the full dataset, thus enhancing both query efficiency and data coherence across blockchains.

The paper written by Gabriel Kaptchuk, Matthew Green, and Ian Miers combines stateless trusted execution environments with public ledgers to ensure verifiability and privacy in decentralized applications [18]. This approach provides secure, stateful computation without persistent storage, allowing verifiable computations and private interactions even when a malicious host is involved.

A2L [23] introduces a cryptographic protocol that achieves atomicity and privacy for payments in channel hubs, enhancing both privacy and scalability. A2L enables unlinkable transactions by relying on digital signatures and timelocks, ensuring user privacy and transaction atomicity without a dependency on advanced scripting languages.

Tesseract [12] leverages trusted execution environments to enable secure, real-time cross-chain trading with strong guarantees for privacy and verifiability. By isolating trade operations in a trusted environment, Tesseract prevents frontrunning attacks and maintains secure trading while ensuring all-or-nothing settlement for trades.

A2L and Tesseract are both solutions tailored specifically for cryptocurrency applications. While both solutions are meant for cryptocurrency environments, their mechanisms for privacy preservation, verifiability, and secure cross-chain data handling are applicable to issues that are faced in supply-chain scenarios.

Chapter 3

Background

3.0.1 HyperLedger Fabric

Hyperledger Fabric is an open-source, enterprise-level permissioned blockchain framework designed for distributed ledger applications [5]. It facilitates secure, private, and scalable networks where multiple organizations can collaborate on a shared ledger. Each organization in a Fabric network operates a set of nodes, called peers, that maintain a copy of the ledger and execute the smart contracts, which are called chain-code in Fabric. These peers provide validation of transactions and endorses them according to specific rules.

The ledger in Fabric is composed of two main parts. The first is the blockchain, which stores an immutable sequence of blocks, each containing multiple transactions. The second part is the world state, which is a key-value database reflecting the current state of the ledger, making it easier to query the latest asset values. This world state is stored in databases like CouchDB [1] or LevelDB [2], depending on the configuration.

To manage the transaction flow, the network uses an ordering service that handles transaction requests and ensures their proper sequencing. This ordering service collects endorsed transactions, orders them into blocks, and distributes these blocks back to the peers for validation and eventual commitment to the ledger. One of Fabric's strengths lies in its endorsement policies, which specify which peers must approve a transaction before it can be accepted. These policies allow flexibility in determining transaction validity, ensuring that only trusted parties within the network approve changes to the ledger. Fabric's permissioned nature means that only authorized participants can join and transact within the network. This permissioning is handled by a Certificate Authority (CA), which issues digital certificates to the network's participants.

One of the key features of Hyperledger Fabric is the concept of channels. Channels allow subsets of organizations to transact privately within the larger network. Each channel maintains its own ledger, and only authorized participants can view the transactions within that channel. This provides data isolation and confidentiality for sensitive business processes. Additionally, Fabric offers Private Data Collections, where organizations can share specific data privately among select participants while still recording a hash of the data on the public ledger. This ensures privacy for sensitive data without compromising the integrity of the network.

Smart contracts, or chaincode, define the business logic that governs how transactions are processed. These smart contracts are written in languages like Go, JavaScript, or Java, and they are executed on peers during the endorsement process.

Transactions proposed by clients are first sent to the endorsing peers, which simulate the execution of the smart contract without actually updating the ledger. If the proposed transaction meets the requirements of the endorsement policy, it is forwarded to the ordering service, which processes the transactions and creates blocks. These blocks are then sent back to the peers for validation and commitment.

During the validation phase, peers check that the transaction endorsements are valid and that there are no conflicting transactions in the block. Once validated, the transactions are committed to the blockchain and the world state is updated accordingly. This modular design allows Fabric to support various consensus mechanisms like Solo, Kafka, and Raft, providing flexibility depending on the network's needs.

Hyperledger Fabric is highly modular and customizable, making it suitable for a variety of enterprise use cases, including supply chain management, financial services, healthcare, and government applications. Its design ensures high throughput, scalability, and strong privacy, which are critical features for enterprise-grade blockchain applications. By using channels, endorsement policies, and private data collections, Hyperledger Fabric ensures that data privacy and confidentiality can be maintained in a distributed, decentralized environment, making it useful for collaborative business processes.

3.0.2 Supply Chain

Supply chains are often plagued by a lack of transparency due to the involvement of numerous intermediaries, each maintaining their own isolated records. This fragmented data storage creates visibility gaps, making it difficult for stakeholders to have a clear view of a product's journey from origin to destination. For instance, a retailer may struggle to verify whether a shipment labeled as "organic" was genuinely sourced from certified organic farms. Without complete transparency, it becomes challenging for consumers, retailers, and regulators to confidently confirm the authenticity and provenance of goods [13].

Inefficiencies and delays further complicate supply chain operations. Traditional systems rely heavily on manual processes and paperwork to track goods, handle transactions, and enforce contracts, which slows down operations. For example, if a shipment of electronics is held up at customs, resolving the issue often requires time-consuming back-and-forth communication between the various parties involved. These inefficiencies, coupled with the lack of real-time data sharing, contribute to bottlenecks that increase operational costs and hinder overall supply chain performance.

The lack of secure tracking mechanisms also opens the door to fraud and counterfeiting. In industries like pharmaceuticals or luxury goods, counterfeit products can easily be introduced into the supply chain and passed off as genuine, posing risks to consumers and damaging the reputations of brands [22]. Verifying the authenticity of goods can be difficult and expensive, especially when there is no central system in place to track the movement of products or ensure their legitimacy across borders.

Another major issue is the lack of traceability and accountability when problems arise. If a defect or safety issue is discovered, it can be extremely difficult to trace the problem back to its source. For instance, during a food recall, companies might struggle to identify where contamination occurred or which supplier is responsible, resulting in widespread recalls that increase costs and delay corrective actions. This lack of traceability not only affects operations but also exposes companies to legal and reputational risks.

Data inconsistency between organizations further worsens these challenges. Different parties within the supply chain may record conflicting versions of events, such as discrepancies in the quantity of goods shipped versus what was received. These inconsistencies lead to disputes over deliveries, payments, or product conditions, which can require external auditors or even legal intervention to resolve. Disputes like these not only disrupt operations but also strain relationships between supply chain partners.

Finally, enforcing contracts within the traditional supply chain model can be inefficient and difficult. Contracts often involve specific terms, such as delivery timelines or quality standards, that are hard to monitor and enforce manually. If a supplier fails to meet its obligations, resolving the issue typically requires manual oversight and lengthy negotiations. This inefficiency can lead to missed deadlines, penalties, and financial losses for all parties involved.

Together, these issues create significant challenges in managing supply chains, from inefficiencies and fraud to difficulties in traceability and dispute resolution. Addressing these problems requires solutions that improve visibility, trust, and collaboration across the entire network of participants.

3.0.3 Zk-Snark

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) [16] is a cryptographic proof system that allows one party (the prover) to prove to another party (the verifier) that a statement is true without revealing any additional information about the underlying data. zk-SNARKs achieve this by encoding the proof in a compact form that can be verified quickly. The defining characteristic of zk-SNARKs is their succinctness, meaning that the size of the proof and the time required for verification are both independent of the complexity of the underlying statement. zk-SNARKs are widely used in privacy-preserving applications, particularly in blockchain technologies, where they enable the validation of transactions or computations without exposing sensitive data.

One of the primary benefits of zk-SNARKs over other zero-knowledge proof systems is their efficiency in terms of proof size and verification time. In many zero-knowledge proof schemes, the prover and verifier must engage in multiple rounds of interaction, which can introduce latency and increase computational overhead. zk-SNARKs are non-interactive, meaning the proof is generated and verified in a single message exchange, making them highly suitable for distributed systems like blockchains where reducing communication complexity is critical. Furthermore, zk-SNARKs are succinct, allowing even complex statements or computations to be verified in a time-efficient manner. This is particularly advantageous in environments with limited resources, such as decentralized networks, where participants need to verify proofs quickly and with minimal computational expense. Compared to other zero-knowledge systems like Bulletproofs or zk-STARKs, zk-SNARKs offer faster verification at the cost of requiring a trusted setup phase, which can be a security concern in some use cases.

Snarkjs is a JavaScript library that facilitates the use of zk-SNARKs, providing tools to compile, generate, and verify zk-SNARK proofs [4]. It is commonly used alongside Circom, a circuit compiler designed for creating arithmetic circuits that form the basis of zk-SNARKs. With snarkjs, developers can easily interact with zk-SNARK proofs by generating necessary cryptographic parameters, such as the proving and verification keys, as well as managing the witness and proof generation processes. The library also includes functionality to perform the trusted setup ceremony, which is an essential step in the creation of zk-SNARKs, and provides tools to export the verification logic into different environments, such as smart contracts on Ethereum [26] or Bitcoin [19]. This makes snarkjs a practical tool for integrating zk-SNARKs into blockchain applications and decentralized systems.

Chapter 4

Overview

4.0.1 Attacker Model

For our system using Hyperledger Fabric [5] with private data collections (PDCs), we assume no concerns about malicious nodes within an organization, the focus is shifted toward external threats and inter-organization security. This simplifies the internal security requirements and places emphasis on protecting private data from external attackers and ensuring secure interaction between collaborating organizations.

One primary threat is posed by external attackers targeting confidentiality. These attackers aim to intercept communications between organizations in the network. Their goal is unauthorized access to private data stored in PDCs, often exploiting vulnerabilities in communication channels or network infrastructure. Such breaches could result from weak encryption or improperly configured network protocols, enabling attackers to eavesdrop or extract sensitive transaction details, undermining confidentiality guarantees.

Another significant threat involves integrity risks. Data integrity is crucial for maintaining trust within the network. External attackers might attempt to tamper with transactional data, causing inconsistencies or errors that disrupt operations. Malicious external organizations may collude to manipulate the endorsement process, potentially introducing fraudulent transactions into the ledger. Such integrity breaches could lead to financial or reputational damage, eroding the trust among network participants.

Network-level attacks form another category of threats. External actors may attempt to intercept or alter communications between trusted organizations. Replay attacks, where valid data is resent to deceive the system, and delay attacks, where critical communications are postponed, can compromise system performance and security. Without robust transaction ordering and message integrity checks, these vulnerabilities could disrupt the network's functionality.

In conclusion, this attacker model, in which we exclude the risk of malicious nodes within an organization, focuses on threats from external actors and interactions between organizations. The most significant risks include external entities attempting to breach confidentiality through side-channel attacks or interfering with the consensus process. In Section 5.0.3 we explain how PAVE is resilient to the types of attacks listed above.

4.0.2 System overview

In our system, we designed a series of specific actions that both the receiver and sender of an asset can perform. These actions encompass various functionalities, including querying, generating and verifying proofs, agreeing to transfers, and executing the actual asset transfer. Each of these actions can be viewed as distinct states, through which parties transition depending on specific events and conditions. For instance, when a party is idle and wishes to initiate a transfer, they can trigger certain transitions in response to the system's current state. For simplicity we assume there are two organizations that wish to perform a transfer of assets between each other. The two organizations are connected via a HyperLedger Fabric channel so each organization will maintain a copy of a ledger to record transactions that occur within the channel. The ledgers will be synced by the orderer nodes that use Raft [20] as their consensus protocol.

The transfer process begins when a node, wishing to receive an asset, initiates a query directed at the party who currently owns the asset. This query is stored within a query pool using HyperLedger Fabrics private data collection feature that utilizes json format collections that both parties have access to. After storing the query into the pool, the recipient will send the key value used to store the query to the sender via a traditional network channel via SSL/TLS. The asset owner will retrieve this query using the received key, which is generated using HyperLedger Fabric's API. This initial step sets the stage for asset exchange and establishes communication between the parties, as depicted in Figure 4.1. The security of this traditional network channel is beyond the scope of this paper. We assume that privacy and security is maintained during the transmission of all data in this network channel.

Once the query has been created, sent, and then received, the sender (the party owning the asset) proceeds to generate a proof using a zk-SNARK implementation [16].

This proof is based on the quantity of the asset specified in the query. After generating the proof, the sender transmits it, along with other verification objects, to the recipient. This communication occurs through the same channel used to transmit the query key to ensure privacy and integrity. This process of proof generation and transmission is illustrated in Figure 4.2.

When the recipient receives the proof, they parse and process it using the zk-SNARK verification method to ultimately verify the sender's claims. This step is critical, as it determines whether the sender can satisfy the asset request. The verification occurs on-chain, allowing the transaction to be logged on the ledger, thus providing both transparency and accountability. At this stage, the sender can also verify that the recipient has received the proof. The verification process, shown in Figure 4.2, forms a key part of the trust mechanism between the parties.

After verifying the proof, if the verification succeeds then the recipient will continue the transfer with an agreement, else terminate the transaction process. If the recipient chooses to proceed, they send a TransferAgreement message to the sender. This message is stored in a separate collection, which both the sender and recipient can access, similarly to the query collection. Similar to sending the query key, the transfer agreement key is also sent via the SSL/TLS channel. The flow of this agreement is depicted in Figure 4.3.

Finally, once the sender confirms that the recipient has agreed to proceed with the transaction, they can initiate the asset transfer. Depending on the agreement, the sender may either transfer a portion of the asset or the entirety of it. The asset transfer marks the completion of the transaction, as shown in Figure 4.4.

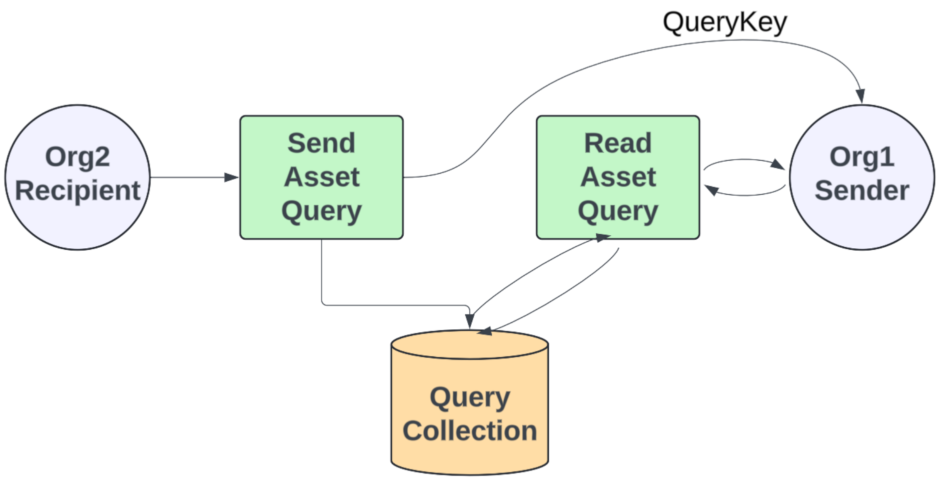


Figure 4.1: Flow Diagram between sender & recipient on transmitting an asset query.

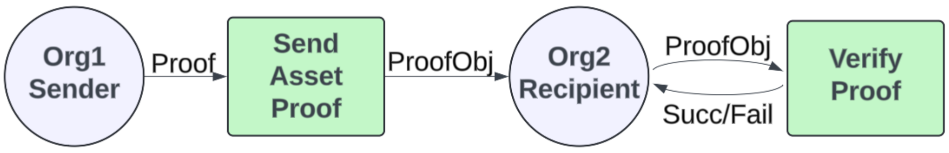


Figure 4.2: Flow Diagram between sender & recipient on transmitting proof object and recipient performing verification on proof.

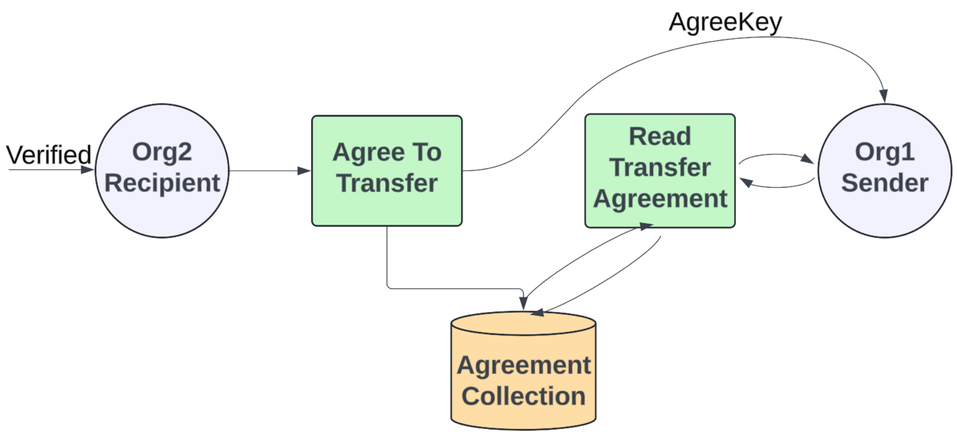


Figure 4.3: Flow Diagram between sender & recipient on transmitting transfer agreement to initiate transferring asset.

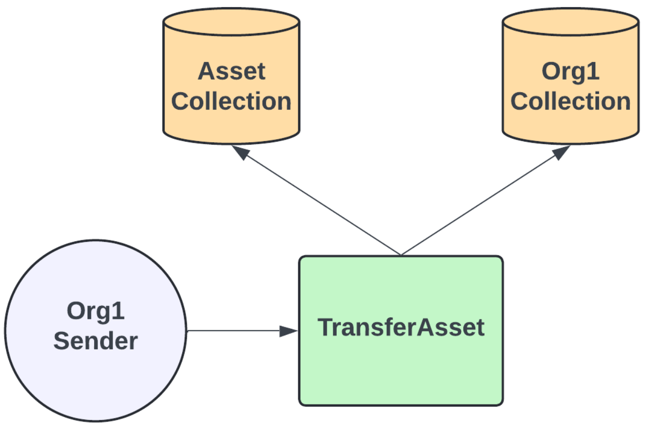


Figure 4.4: Flow Diagram between sender & recipient on transferring the asset and writing to the asset collection with the modified quantities

Chapter 5

System Design

5.0.1 Design

In Hyperledger Fabric [5], chain-code serves as the business logic or smart contract that defines and enforces the rules governing the interactions between participants on the blockchain network. Chain-code is executed by peers in the network to perform various functions related to asset management, ledger updates, and transaction processing.

Private data collections are used to facilitate secure communication between organizations within the channel. Each collection can be configured to provide read and write permissions either to all organizations within the channel or to a specific subset of them. Our implementation includes multiple collections, each serving a distinct function.

- **Query Collection**: Holds information regarding queries made from recipients so that asset owners can retrieve requested values. Keys are generated using the a query object string, asset ID, along with the recipients unique ID.

- **Agreement Collection**: Holds all agreements sent by recipients so that senders can identify when a transaction proof has been verified and agreed on. Keys is generated by using a combination of agreement object string, recipients organizations unique key, and the asset ID.
- **Asset Collection**: Stores the entirety of the asset information besides private information such as quantity. All organizations within channel can access this collection. Keys are generated using a combination of the asset object string, owning organizations unique ID, along with the asset ID.
- **Organization Specific Collection**: Used to store private information of an asset. Only users within the same organization can access this collection. Key is generated using only the asset ID.

To communicate with the ledger we developed smart contract API functions to facilitate the reading and writing to private data collections. Some of the smart contract functions such as CreateAsset, AgreeToTransfer, and TranferAgreement were modified from an existing test network within HyperLedger Fabric's tutorial [5]. The remaining chain-code functions were developed to add an extra layer of communication prior to transferring the asset to ensure there was more verifiability for the recipient.

- **CreateAsset**: This function is responsible for creating a new asset and storing it within the private data collection of the owning organization. It uses transient data to pass the asset's properties (e.g., ID, color, size, and quantity value) and ensures that the asset does not already exist. It also validates that the client requesting the asset creation belongs to the same organization as the peer, preventing unauthorized access to private data from other organizations. After validation, the asset is saved both in the assetCollection and in the organization-specific private data collection.

- **SendAssetQuery**: This function allows a potential buyer to send a query for a specific asset to check whether the sender has enough quantity of that asset. It takes the asset ID, quantity value, and sender organization as input and stores the query in the private data collection queryCollection for later processing by the asset owner. The query is indexed using a composite key consisting of the query object string, asset ID, organization ID. From Figure one we can see the sender of the assets

- **ReadAssetQuery**: The sender organization uses this function to read the asset query submitted by a buyer. It retrieves the asset query from the queryCollection using the composite key (created from the asset ID and sender organization) and returns the query details. The function also handles de-serialization of the query. After reciving the query from the queryCollection the function deletes the query to prevent repetitive reads of the same query.

- **SendAssetProof**: This function is not a smart contract API but is handled within the asset owners application code. This function allows the sender of an asset to send a Zero-Knowledge Proof (ZKP) to the buyer, verifying that the sender possesses the required quantity of the asset without revealing specific details. The proof, verification key, and public signals are sent over a traditional network channel using SSL/TLS.

- **ReadProofVerify**: This function reads and verifies the ZKP provided by the asset sender using the snarkjs.groth16.verify() method. It takes the asset ID, verification key, proof, and public signals as inputs. If the proof is valid and the public signals indicate that the sender meets the buyer's requirements, the function returns a TransientProofResults object indicating the proof's validity.
- **AgreeToTransfer**: This function allows a buyer to agree to an asset transfer by recording the agreed-upon asset value in the buyer's organization's private collection and associating the buyer's client ID with the asset in the agreementCollection. The function ensures that the client belongs to the correct organization and creates a transfer agreement for the asset using a composite key consisting of asset object string, owning organizations ID, and asset ID.
- **TransferAsset**: This function handles the transfer of an asset from the current owner to the buyer. It verifies that the client initiating the transfer is the asset owner and checks that the buyer has agreed to the same asset quantity. The function either updates the asset's ownership and reduces the remaining quantity for the owner or fully transfers the asset if the entire quantity is transferred. It also manages the deletion of transfer agreements once the transaction is complete.

Chain-Code API	**Input**	**Output**	**Description**
CreateAsset	assetprop	nil	Generates asset for the calling node
SendAssetQuery	assetID, quantreq, send_org	nil	Sends Asset Query for asset to owning org
ReadAssetQuery	assetID, send_org	QueryObj	Retrieves asset query from collection
ProofVerify	assetID, vkey, proof, pubsig	bool	Runs proof verfication, returns validity
AgreeToTransfer	assetID	nil	Receiver agree/continue transfer of asset
TransferAsset	assetID, buyerID	nil	Owner agree and performs asset transfer
ReadTransferAgree	assetID	TransAgreeObj	Retrieves TransAgreeObj from collection
ReadAsset	assetID	AssetObj	Reads asset with no private info
ReadAssetPrivateDetail	assetId, collection	PrivAssetObj	Reads asset with private info
DeleteAsset	assetID	nil	Removes entry in collection for Asset
DeleteTransferAgree	assetID	nil	Removes TransAgreeObj from collection
PurgeAsset	assetID	nil	Removes asset holding sensitive info

Table 5.1: Chain-code library APIs used to communicate with HyperLedger Fabrics ChaincodeStub Interface.

Inputs	**Description**
assetprop	Details of asset (ID, quantity, ect.)
assetID	Unique identifier for asset
quantreq	Requested Quantity from a recipient
send_org	Organization identifier for the recipient
vkey	Verification key used to verify proof
proof	Generated proof created by sender
pubsig	PublicSignal used by recipient to verify proof
buyerID	Unique identifer for the recipient of transaction
collection	Collection name, either public collection for all org or restricted collection for each org

Table 5.2: Chain-code library inputs.

A full list of all chain-code API methods can be seen within Table 5.1. This table shows the previously mentioned methods as well as read and cleanup methods whose functionality is self explanatory. Along with this is a list and description of all arguments the chain-code API can pass in Table 5.2.

In this Hyperledger implementation, organizations wishing to perform transactions with each other join a shared channel, allowing them to freely transact and have access to both public and private collections. For instance, if Node B in org2 wishes to receive a portion of asset1 from Node A in org1, a specific sequence of steps ensures the transaction is securely processed.

First, Node B initiates the transaction by formulating a query for asset1 and directing it to Node A. This query takes the form of a *TransientAssetQueryObject*, a JSON object containing details about the desired asset, the requested quantity, and the buyer's ID. This JSON is keyed with the asset ID and the organization's unique identifier to distinguish which organization generated the request. Node B then stores this query in the *QueryCollection* using the *SendAssetQuery* chaincode method. Node B will also transmit the generated key to Node A using a secure SSL channel, notifying Node A there is a new query to read. Upon receiving the key Node A will retrieve and parse the query object from the query pool to identify the potential buyers and the quantities they've requested. In the optimized proof aggregation implementation Node A will wait a static time window for incoming queries in order to pack as many requesters as possible into a single proof. In this window a check is done to ensure Node A does not commit to too many requesters they can not satisfy with their current quantity value.

Next, using the quantity specified by Node B, Node A generates a simple arithmetic proof with JavaScript's zk-SNARK [16] implementation (snarkjs) [4] to verify if its stored quantity of the asset is sufficient to meet Node B's request. This proof, along with the verification key and public signals object, is then transmitted to Node B over a secure SSL socket connection. this occurs outside the chain-code API.

Upon receiving these proof objects, Node B calls the *ProofVerify* chaincode method to verify the proof and records receipt of the proof on the ledger. If Node B confirms the proof's validity, it proceeds by agreeing to the transaction terms and records the incoming quantity in its organization's private data collection. This agreement requires a call to the *AgreeToTransfer* chaincode method, which creates a new asset object to store information about the incoming asset in Node B's specific private collection. The agreement is also stored in an *AgreementCollection* for Node A's use. Node B will also send the transfer ageement key to Node A using the SSL channel previously mentioned.

Node A waits until the transfer agreement key is received before they can know Node B's confirmation prior to finalizing the asset transfer. Upon confirming the terms through the *ReadTransferAgreement* chaincode method, Node A proceeds to complete the transaction by calling the *AssetTransfer* chain-code method which updates the asset ownership in the *AssetCollection* to reflect Node B's ownership. Additionally, Node A updates its organization-specific collection to reflect the asset's new quantity post-transaction. After finalizing a transaction all objection within the *QueryCollection* and

AgreementCollection relating to this specific transaction are removed. These operations are also illustrated in Figure 5.1 below.

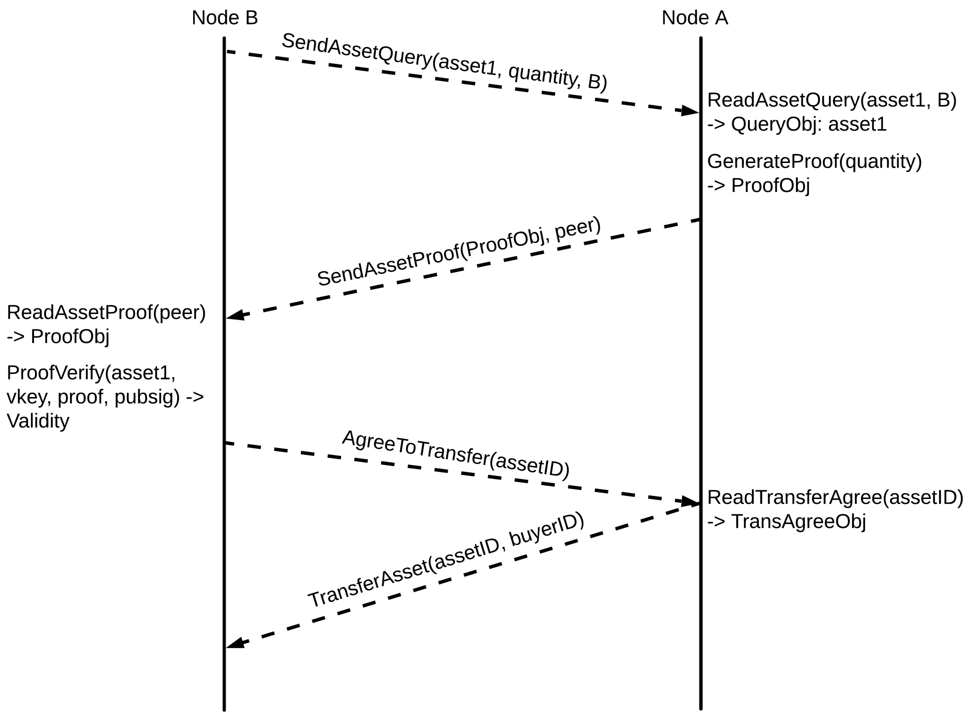

Node B
Node A
SendAssetQuery(asset1, quantity, B)
ReadAssetQuery(asset1, B)
-> QueryObj: asset1
GenerateProof(quantity)
-> ProofObj
SendAssetProof(ProofObj, peer)
ReadAssetProof(peer)
-> ProofObj
ProofVerify(asset1, vkey, proof, pubsig) ->
Validity
AgreeToTransfer(assetID)
ReadTransferAgree(assetID)
-> TransAgreeObj
TransferAsset(assetID, buyerID)

Figure 5.1: Sequence of operations to complete a transaction between Node A (sender) and Node B receiver.

5.0.2 Multi-Chain Integration

In the context of multi-chain integration using PAVE, its incorporation into a multi-enterprise or multi-chain scenario leverages Hyperledger Fabric's foundational features, such as channels and Private Data Collections. These mechanisms ensure privacy and scalability while facilitating collaborative interactions across organizations.

Each channel in Hyperledger Fabric represents a private sub-network where participating organizations maintain a shared ledger. Transactions within a channel are isolated, ensuring data privacy and integrity. PDCs enhance this model by enabling finer-grained privacy controls, allowing only specific organizations to access sensitive data while others see only its cryptographic hash. This design aligns seamlessly with PAVE's objectives by safeguarding confidential asset-related data during cross-organizational transactions.

To integrate PAVE in a multi-chain scenario, consider a system involving O_1, O_2,..., O_m, representing different organizations. On-boarding a new organization in a Fabric network involves several steps to maintain the system's security and integrity. When adding a new organization, such as O_{m+1}, the network administrator first updates the system configuration to include O_{m+1}'s identity and policies in the channel configuration block. This update is executed using Fabric's ConfigTxlator tool. Once updated, O_{m+1} deploys its peers and joins the channel by synchronizing its ledger with the existing one. Once joined the privacy-preserving and verifiability that PAVE provides can be utilized.

In this system each organization operates its own peers and contributes to private or shared ledgers. For example, an asset owner in O_1 can use PAVE's proof aggregation mechanism to consolidate multiple asset verification requests from O_2, O_3,..., O_k (a subset of m) into a single zero-knowledge proof. This proof is shared only with peers within the same channel, ensuring that only the requesting organizations can verify the proof without exposing sensitive data, such as asset quantities or owner identity.

In a supply chain involving O_1 as a supplier, O_2 as a distributor, and O_3 as a retailer, O_4 (a logistics partner) might join the system to provide delivery operations. Upon joining the channel, O_4 is granted access only to relevant logistics data within a specific PDC while remaining excluded from sensitive financial or asset-related information. This granular control is facilitated through the policies put in place for each private data collection, ensuring that O_4 can operate without compromising other participants' data privacy.

By integrating PAVE with Hyperledger Fabric's capabilities, the system not only supports secure and efficient interactions among existing participants but also provides a scalable framework for expanding the network while maintaining robust privacy guarantees.

5.0.3 Security Analysis

To address the identified threats in our attacker model, the system incorporates several strategies. First of which being ensuring confidentiality between organizations. Hyperledger Fabric's [5] PDC mechanism enables private data sharing with selected peers, while other participants see only a cryptographic hash of the transaction. Communication channels are secured using Transport Layer Security (TLS), ensuring authenticated and encrypted data transmission that prevents unauthorized interception.

Preserving integrity is achieved through endorsement policies that require multiple independent organizations to approve a transaction. This prevents any single entity from unilaterally manipulating the transaction process. Consensus algorithms like Raft [20] or Practical Byzantine Fault Tolerance (PBFT) [15] further reinforce integrity by achieving agreement on transaction validity across a majority of nodes, even in the presence of malicious actors. Hash-based transaction validation and time-stamped logs add additional layers of protection, creating an auditable trail for detecting and resolving anomalies.

Network security is enhanced by authenticated communication channels and the use of cryptographic techniques such as digital signatures and PKI. These measures ensure message authenticity and integrity, making unauthorized modifications challenging. Additionally, Fabric's use of time-stamping and transaction ordering prevents replay or delay attacks by external entities.

By combining these mechanisms provided by Hyperledger Fabric, PAVE effectively mitigates threats from external actors, preserves data integrity, and ensures secure and trustworthy interactions between organizations in a collaborative environment.

Chapter 6

Evaluation

6.0.1 Implementation

HyperLedger Fabric [5] utilizes Docker containers power each component, such as peer nodes, ordering services, and chaincode, running them in isolated environments. Within the Docker network, each organization and node operates on a unique port number, which helps with identification and communication between nodes. Each organization and node is configured using custom YAML files. These YAML files define environment details, such as peer and orderer node configurations, identity credentials, network addresses, and permissions. This setup establishes how each node connects and operates within the Docker network, coordinating the components of the Fabric network.

For managing ledger data, CouchDB was used as our state database. CouchDB, a NoSQL document-oriented database, allows JSON-based queries that support the efficient management of private data collections [1]. For private data collections, CouchDB handles the private states similarly to public states but ensures that private data is only accessible to authorized organizations by encrypting the data before storage.

To develop chaincode, Fabric supports several programming languages, including Go, Java, and TypeScript. We chose TypeScript along with snarkjs [4], a library for zk-SNARKs [16]. This library enabled us to generate and verify cryptographic proofs that confirm the validity of data without revealing it. Integrating snarkjs with our node application and chaincode allowed us to handle proof generation and verification smoothly.

To evaluate our simulated environment, we generated six different organizations with their own organization specific private data collection. All organizations have full access to query, agreement, and asset collections. In this evaluation we tested to see the average latency to generate a single proof as well as the latency to verify a proof. Following this we then recorded the average latency to generate sequential proofs for five separate recipient organizations compared to using our aggregated proof generation optimization. Along with this we also analyzed the space overhead of sending proofs for both the sequential and aggregated simulations.

6.0.2 ZKP Latencies

To understand the ideal zero-knowledge proof to use we evaluated various types of implementations, such as Plonk2 (a zk-SNARK implementation) [24] [3] [17], a Rust-based implementation of Bulletproofs [14], and Rust's Winterfell [6] module which provides a zk-STARK implementation [10]. We use Plonk2 as our zk-SNARK example here since the implementation used within our work is less optimized compared to that of Winterfell. We used snarkjs within our HyperLedger Fabric solution for ease of integration within both node application and chain-code, given the programming language limitation of HyperLedger Fabric.

In our first evaluation of these ZKPs we developed programs that would compute a simple comparison operations to identify if one value is greater than another, similar to that of this papers solution. From the results of this evaluation it was found that zk-SNARKs significantly outperformed Bulletproofs in terms of latency for proof generation and verification as seen in Table 6.1. Bulletproofs, while versatile and not requiring a trusted setup, use an interactive process that depends on more computationally intensive operations, especially in generating proofs. The lack of a trusted setup in Bulletproofs does provide an advantage in some contexts, as it offers a higher level of transparency and flexibility. However, this comes at the cost of speed making the protocol inherently more time-consuming than zk-SNARKs for both proof generation and verification.

On the other hand when we compare the performance of zk-SNARK to zk-STARK we can see that zk-SNARK outperforms zk-STARK in proof generation, while performing slightly worse during proof verification. The logic performed in this test is simplistic, which results in less complex circuits or arithmetic operations. Therefore we see a slight loss in performance as this specific example does not thoroughly evaluate the difference between zk-SNARK and zk-STARK. We perform a more thorough benchmark between the two solutions using fibonacci sequences below.

ZKP	**Gen Latency (ms)**	**Verify Latency (ms)**	**Proof Size (Bytes)**
Bulletproofs	933.05	109	616
zk-STARK	217.55	1.22	4240
zk-SNARK	105.80	2.19	6712

Table 6.1: Latencies for various Zero-Knowledge Proof implementations, showing zk-SNARK provides better time to generate and verify compared to Bulletproofs.

To gain deeper insights into the differences and benefits of zk-SNARK and zk-STARK, we conducted a detailed evaluation by generating zero-knowledge proofs for identifying the n-th Fibonacci number in a sequence. This benchmark was chosen due to its computational complexity and its relevance in highlighting the scalability of these proof systems. The evaluation aimed to identify the trade-offs in proof generation latency, verification latency, and proof size between the two solutions.

Our results revealed that zk-SNARK consistently outperformed zk-STARK in proof generation latency as the n-value in the Fibonacci sequence increased. This observation aligns with existing research by Matter Labs [7], which attributes the difference to the prover algorithmic complexities of the two solutions. zk-SNARK employs a prover with an algorithmic complexity of O(n logn), while zk-STARK operates at a higher complexity of O(n poly-log(n)). These findings, illustrated in Figure 6.1, confirm that zk-SNARK offers superior efficiency in proof generation, particularly as the computational scale grows.

When evaluating proof verification latency (Figure 6.2), we observed a significant divergence in behavior between zk-SNARK and zk-STARK at higher n-values. zk-SNARK verification latency remained relatively constant, even for very large n-values, consistent with Matter Labs' findings of a verifier complexity of ∼O(1). In contrast,

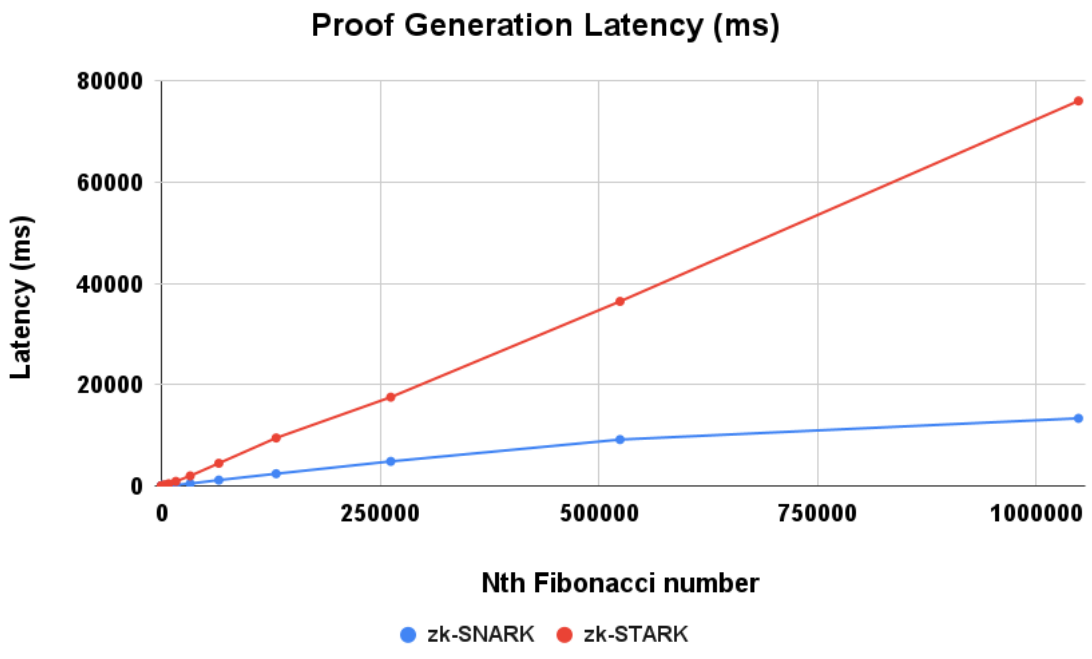


Figure 6.1: Proof generation latency comparing Plonk2 (zk-SNARK) and Winterfell (zk-STARK)

zk-STARK verification latency exhibited a gradual increase, corresponding to its verifier complexity of O(poly-log(n)). This distinction supports zk-SNARK's suitability for applications that are dependent on verification speed.

Finally, our analysis of proof size demonstrated that zk-SNARK tends to produce larger proofs compared to zk-STARK (Figure 6.3). This outcome reflects the inherent design philosophies of the two systems: zk-SNARK is optimized for speed as circuit complexity increases, whereas zk-STARK excels in handling larger computations and constraints as complexity increaases. The compact nature of zk-STARK proofs highlights its advantage in scenarios requiring minimal storage or transmission overhead. Also mentioned in Matter Labs evaluation was the difference in gas cost, which

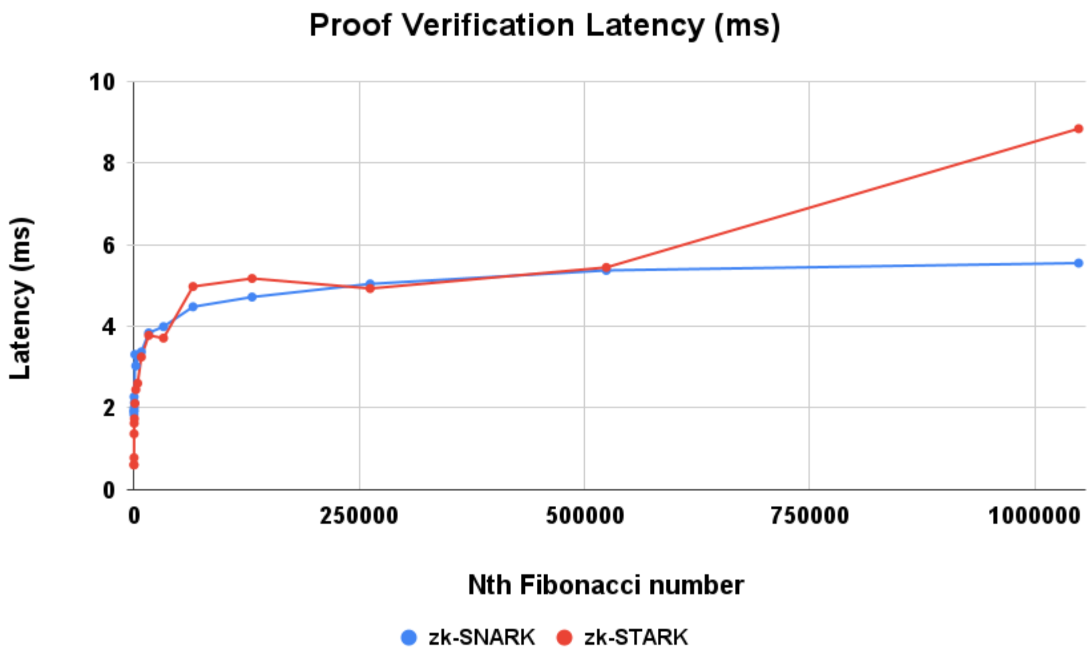


Figure 6.2: Proof verification latency comparing Plonk2 (zk-SNARK) and Winterfell (zk-STARK).

illustrated how zk-SNARKs is far more gas-efficient than zk-STARKs, primarily due to their constant-time verification.

In conclusion, our evaluation highlights the trade-offs between zk-SNARK and zk-STARK across key performance metrics. While zk-SNARK demonstrates lower proof generation latency and constant-time verification, zk-STARK offers more compact proof sizes and scalability for large-scale computations. These findings provide clarification on which solution would fit best our application. Since this solution is latency dependent and the proof generated by each requester is simplistic there would be no need for using zk-STARK, but instead zk-SNARK would be the most ideal solution to use.

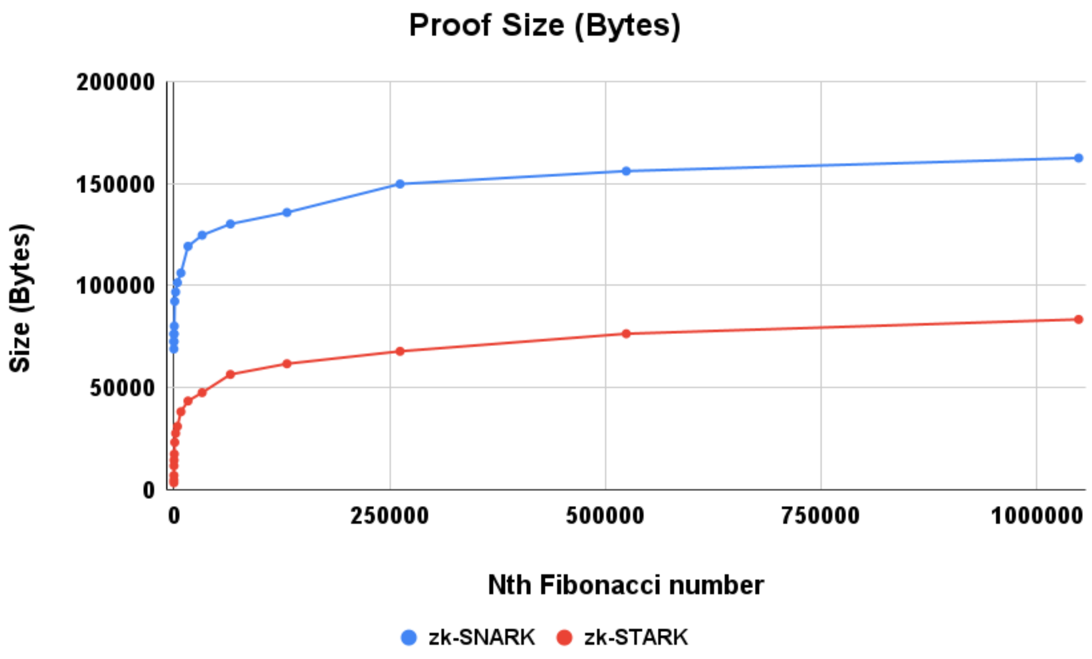


Figure 6.3: Proof Size comparing Plonk2 (zk-SNARK) and Winterfell (zk-STARK).

6.0.3 Latency from Generating/Verifying Proofs

We conducted an analysis to understand the latency involved in generating and verifying a single proof without using our simulation. Proof generation occurs within each node's application, entirely off-chain, meaning it does not involve any communication with the chaincode. This decision helps mitigate the overhead associated with executing chaincode during the lifecycle of a write operation in our chaincode API. Each write operation undergoes multiple steps, including proposal endorsement, ordering, validation, and commitment. These steps introduce additional overhead compared to standalone code execution. Chaincode execution, as part of this lifecycle, also requires communication between endorsing peers, consensus among multiple nodes, and state validation, all of which contribute to the overall time taken to complete a transaction. By performing proof generation off-chain, we effectively avoid this overhead.

On the other hand, proof verification tends to be a low-latency operation, typically making it suitable for inclusion in the transaction process within the chaincode. This approach allows the verification process to be recorded on the ledger. Based on our latency evaluations, we observed that generating a single proof off-chain incurs approximately 300 milliseconds of latency, while verifying this proof within the chaincode results in a much lower latency of around 30 milliseconds.

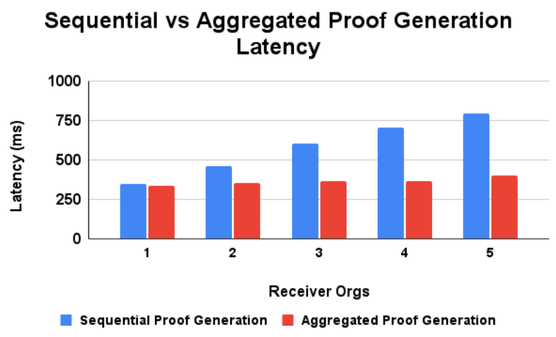
6.0.4 Proof Aggregation Optimization

We conducted simulations to evaluate the optimization of our proof generation process by implementing proof aggregation for multiple recipient organizations requesting portions of the same asset. This optimization allows us to aggregate proofs instead of generating separate proofs for each requester, improving efficiency. The simulation took into account not only the time required to generate the proof but also any preprocessing needed to retrieve a previously generated proof if it had already been created for another party.

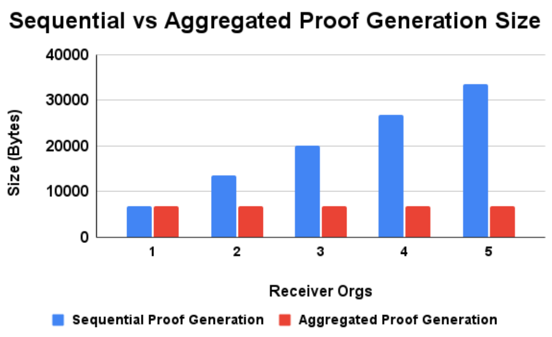
The results of this evaluation, illustrated in Figure 6.4a, demonstrate a clear difference between generating proofs sequentially (without aggregation) and incorporating aggregation. Without aggregation, the generation of proofs follows a linear trend, with each new proof adding a significant amount of time. However, when using aggregation, the additional latency introduced for each extra requester is minimal, increasing by only about 10-30 milliseconds per requester. This highlights the efficiency gains achieved through proof aggregation in reducing the overall time for multiple requesters.

6.0.5 Proof Size Overhead

In evaluating the space efficiency of proof generation, we assessed the total storage overhead in bytes required to accommodate 1 to 5 requests under both sequential and aggregated implementation approaches. Our goal was to compare how each method affects storage requirements, particularly when handling multiple requests for the same asset. This comparison highlights the potential of proof aggregation to reduce the overall space demand significantly.



(a) Proof Generation Latency



(b) Proof Size

Figure 6.4: a) Latency incurred from performing transactions between different organizations. b) Overall proof space overhead for different recipients for the same asset. Simulated for sequential proof generation and using proof aggregation.

In cases where all 1 to 5 requests arrive within the defined observation window for the sender, an aggregated implementation enables the use of a single, fixed-size proof that satisfies multiple requesters. This approach contrasts sharply with the sequential implementation, where each new request triggers an additional proof, resulting in a linear increase in storage size as the number of requests grows as seen in Figure 6.4b. By limiting the need for multiple proofs and instead accommodating multiple requesters with a single, aggregated proof, this method effectively mitigates storage expansion, even as requests accumulate.

Chapter 7

Conclusion

In conclusion, PAVE (Privacy-preserving Aggregated Verification for Enterprises) ensures privacy-preserving verifiability and scalability. By integrating zero-knowledge proofs (ZKPs), specifically zk-SNARKs, PAVE enables enterprises to verify asset ownership and transactions without exposing sensitive data, providing trust and collaboration across organizations. Additionally, the introduction of a proof aggregation mechanism reduces computational overhead by consolidating multiple transaction verifications into a single proof, enhancing the system's efficiency in high-volume scenarios.

Through extensive evaluation, PAVE demonstrates significant improvements in both latency and storage efficiency while maintaining robust privacy guarantees. These contributions position PAVE as a scalable and practical solution for supply chain systems, addressing the dual challenges of security and performance in multi-enterprise environments. By laying the groundwork for efficient, privacy-focused blockchain interactions, PAVE offers a pathway to more secure and collaborative supply chain networks.

Bibliography

[1] 1. Introduction — Apache CouchDB® 3.4 Documentation.

[2] google/leveldb: LevelDB is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.

[3] plonky2 1.0.0 - Docs.rs.

[4] Snarkjs. https://github.com/iden3/snarkjs.

[5] Tutorials — Hyperledger Fabric Docs main documentation.

[6] facebook/winterfell, November 2024. original-date: 2021-04-23T19:20:43Z.

[7] matter-labs/awesome-zero-knowledge-proofs, November 2024. original-date: 2018-11-10T05:14:37Z.

[8] Mohammad Javad Amiri, Boon Thau Loo, Divyakant Agrawal, and Amr El Abbadi. Qanaat: a scalable multi-enterprise permissioned blockchain system with confidentiality guarantees. volume 15, pages 2839–2852, July 2022.

[9] Antier_SEO. Decoding ZK-STARK vs ZK-SNARK: An In-Depth Comparative Analysis, April 2024.

[10] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. 2018. Publication info: Preprint. MINOR revision.

[11] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. In 2014 IEEE Symposium on Security and Privacy, pages 459–474, San Jose, CA, May 2014. IEEE.

[12] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-Time Cryptocurrency Exchange Using Trusted Hardware. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pages 1521–1538, London United Kingdom, November 2019. ACM.

[13] Robert Brice. The Top 10 Supply Chain Management Challenges for 2023 & Beyond: An In-Depth Look, March 2023.

[14] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More, 2017. Publication info: Published elsewhere. Minor revision. 39th IEEE Symposium on Security and Privacy 2018.

[15] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance.

[16] Thomas Chen, Hui Lu, Teeramet Kunpittaya, and Alan Luo. A Review of zk-SNARKs, October 2023. arXiv:2202.06877.

[17] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge, 2019. Publication info: Preprint.

[18] Gabriel Kaptchuk, Matthew Green, and Ian Miers. Giving State to the Stateless: Augmenting Trustworthy Computation with Ledgers. In Proceedings 2019 Network and Distributed System Security Symposium, San Diego, CA, 2019. Internet Society.

[19] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System.

[20] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm.

[21] Haochen Ren, Xiulong Liu, Hao Xu, Chenyu Zhang, and Keqiu Li. CubeChain: Generalized Query Framework for Intra- and Cross-Chain Scenarios. In 2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS), pages 345–355, July 2024. ISSN: 2575-8411.

[22] Center for Drug Evaluation and Research. Counterfeit Medicine, August 2024. Publisher: FDA.

[23] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A2L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs. In 2021 IEEE Symposium on Security and Privacy (SP), pages 1834–1851, May 2021. ISSN: 2375-1207.

[24] Polygon Zero Team. Plonky2: Fast recursive arguments with plonk and fri.

[25] Ruping Wang, Siqi Zhong, Qin Zhou, and Jun Tu. A Trustworthy Data Verification Technique for Cross-Chain Data Sharing Based on Merkle Trees. In 2023 International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE), pages 1–6, April 2023.

[26] Dr Gavin Wood. ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER.

[27] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless Cross-chain Bridges Made Practical, October 2022. arXiv:2210.00264 [cs].