Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

The Performance Impact of Zero-Knowledge Proof on Blockchain Frameworks in Terms of Asset Lending Management System

Author: Rui Pan (VU: 2717917 UvA: 13676105)

1st supervisor:	Dr. Adam S. Z. Belloum	University of Amsterdam
2nd reader:	Dr. Zeshun Shi	Delft University of Technology

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

March 24, 2023

"I am the master of my fate, I am the captain of my soul"
from Invictus*, by William Ernest Henley*

Abstract

Privacy-preserving has been a popular concern in the field of blockchain due to the broad need for anonymity and confidentiality during transactions. The goal of this thesis is to investigate the performance impact of using zero-knowledge proof (ZKP) on two mainstream blockchain frameworks for asset lending management systems and incidentally, exploring viable solutions for measuring the performance of blockchain under privacy-preserving settings. The study focuses on evaluating the scalability, efficiency, and security of various blockchain platforms that incorporate ZKP technology. Through extensive experimentation and analysis, the research finds that the use of ZKP in blockchain frameworks results in worse performance in terms of transaction rate and latency, while maintaining the privacy of user personal information. The results of this research have significant ramifications for asset lending management systems built on blockchain platforms, as well as their design and implementation. The findings of this research point to the inclusion of ZKP technology as a potential means of resolving the problems with existing blockchain-based asset lending systems' ability to protect user privacy, but at the expense of blockchain performance.

Keywords: *Blockchain, Privacy-Preserving, Anonymity, Performance Evaluation, Hyperledger Fabric, Ethereum*

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Glossary

Caliper	Hyperledger Caliper; a generic benchmark tool for mainstream blockchain frameworks
gRPC	gRPC Remote Procedure Calls; a cross-platform open source high-performance remote procedure call framework developed by Google
Idemix	Identity Mixer; a cryptographic protocol suite used to provide anonymity and confidential features, such as ZKP, to Hyperledgre Fabric framework
JMeter	Apache JMeter; a generic benchmark and measurement tool used to load test functions and measure performance
PoA	Proof-of-Authority
PoS	Proof-of-Stake
PoW	Proof-of-Work
Solidity	a programming language used to create smart concerts for Ethereum
zk-SNARKs	zero-knowledge succinct non-interactive argument of knowledge
ZKP	Zero-Knowledge Proof; a cryptography protocol used to prove the validity of a statement while the prover doesn't need to reveal the additional information about the statement
ZoKrates	a toolbox on Ethereum, It enables verifiable computation in DApp, from the specification of program in a high-level language to generating proofs of computation to verifying those proofs in Solidity

1 Introduction

Today's decentralisation applications, in particular those based on the Ethereum framework, have emerged with a large number of them every year, which have also become a pillar of the Web3 world. The motivation behind this phenomenon is the more advanced and mature Blockchain technology, which acts as a backbone, has been validated by the academic community and recognised by the industry (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13). People have realised its potential through its adoption in various fields, including but not limited to digital currency, anonymous voting and property management, where the traditional centralised issues such as lack of privacy, opaque market and cost fees can be mitigated by blockchain due to its decentralised and immutable nature (14).

Given the vast market and opportunity, many researchers are concerned with improving the security and privacy of existing blockchain frameworks, which will store countless transaction records and personal data in ledgers and frequently exchange digital assets between participants (15). And this is where zero-knowledge proof (ZKP), which can make transactions unlinkable and provide anonymous transactions for privacy concerns, is gaining more attention. However, although several mainstream blockchain frameworks have proposed relevant solutions, for example Hyperledger Fabric offers Idemix to encrypt transactions using ZKP and Ethereum has Zokrates, it has rarely been applied and evaluated in real-world scenarios (16, 17). Furthermore, a few studies try to find out the impact of ZKP on blockchain in specific systems, which only focus on comparing the performance difference of blockchain frameworks. Therefore, it's essential to understand the cost of applying privacy-preserving solutions on blockchain frameworks, especially since ZKP is a computationally intensive workload that could consume additional computational resources for each transaction, it's crucial to find out whether ZKP will affect the overall performance of the blockchain and make it less attractive to the industry.

As a result, we choose a system for managing asset lending as our test case. This is because there are a lot of businesses need to keep track of the equipment belonging to the company that employees borrow, such as laptops, tablets, smartphone, VR headsets, or even bicycles. Since each lease record is a transaction containing the asset's information, owner, transfer record, and damage report, which is maintained in the ledger and cannot be changed by anybody, every new transaction must be based on the most recent transaction record. By doing this, the business can manage its assets with ease and determine who is liable for any damaged or missing assets.

We explore this niche by deploying ZKP on two popular blockchain frameworks, creating a control and experimental group to observe several key performance metrics to see if ZKP negatively affects blockchain performance. Finally, there will be a conclusion on the basis of quantitative results.

This thesis investigates and describes the performance impact of the privacy-preserving technique, ZKP, on two mainstream blockchain frameworks that are being tested in the asset management system scenario. The rest of the thesis is structured as follows: Chapter 2 first introduces the basic blockchain background and two tested blockchain frameworks, then explains the principle, advantages and challenges in ZKP, in addition, the real scenario design is also explained, and finally we state the problems encountered and research questions. Chapter 3 presents the design of the solution to integrate the ZKP technique into blockchain frameworks, and the workflow of benchmarking the blockchain with and without ZKP. Chapter 4 details the implementation of the blockchain-based asset management system and the solution for benchmarking the blockchain frameworks, including two types of benchmark architectures for different frameworks, how to integrate Apache JMetric [1] with Fabric-SDK-Java [2], etc. Chapter 5 discusses the experiments, comparison results and evaluation. Finally, Chapter 6 presents the discussion, answers to the research questions, and future work.

[1]https://jmeter.apache.org/

[2]https://github.com/hyperledger/fabric-sdk-java

2

Background

Chapter 2 provides an introduction to the basic background of blockchain technology and presents two commonly used blockchain frameworks. The chapter then delves into the principle, advantages, and challenges of zero-knowledge proof (ZKP) and its applications in real-world scenarios. The chapter concludes by highlighting the problems encountered in implementing ZKP and the research questions that will be addressed in the subsequent chapters.

2.1 Blockchain

2.1.1 Overview

Blockchain arose from the classic problem of how to make it easier for users to quickly create a verifiable time stamp on digital documents, given the low cost of counterfeiting (18). In addition, the real-world scenario is a distributed network of users who also need methods to repeatedly create and verify timestamps. Stuart Haber and Scott Stornetta initially came up with a native solution that required the documentation's x hash value to be calculated y=*hash(x)*, then sent y to a third-party provider called a time-stamping service (TSS), which appends a timestamp. By comparing the documentation to the TSS copy and adding a signature and date, users may verify that TSS got the hash and that the information has not been changed.

In the meanwhile, the authors noted that this system depends on a centralised TSS, which is difficult to trust. Given the dispersed network, the authors suggest using a network of computers to timestamp the document so that ensure the network integrity. Overall, this decades-old research provides today's blockchain with a sound theoretical foundation, in particular, it defines the nature of blockchain as a chronological chain of hashed data and maintained by all nodes in the network. Whereas, another key concept is consensus, as multiple transactions need to submit changes simultaneously, requiring a consensus mechanism to determine the order of submission. PoW and PoS are the most widely used techniques. The previous is referred to as mining, and it calls on all nodes to solve a problem; the one who does so the quickest gets to submit the transaction first, and so on. While the latter is a PoW alternative where validators are selected based on the amount of stake they possess in the network rather than mining to validate transactions. Bitcoin continues to employ PoW, but Ethereum has switched to PoS due to PoW's excessive energy usage (19).

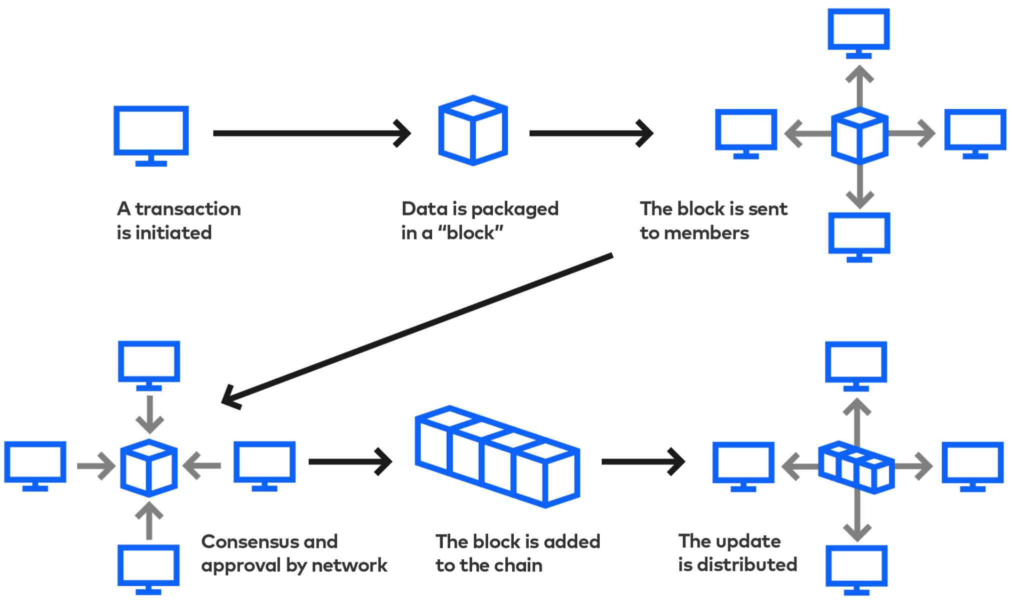

A transaction is initiated
Data is packaged in a "block"
The block is sent to members
Consensus and approval by network
The block is added to the chain
The update is distributed

Figure 2.1: Major steps of submitting a transaction to the blockchain.

Figure 2.1 shows blockchain transactions to illustrate how its components interact, first, a transaction can be created by an individual who signs the transaction with a private key. Then, the transaction will be packaged to a block with other transactions and broadcast to all members in the blockchain network. Upon receiving the incoming block, other user can verify and consensus to decide whether it can be approved and added to the blockchain. If approved, the blockchain gets updated and all distributed ledger also will be updated with the latest block. Blockchain transactions record value exchanges like bitcoin transfers or asset transfers. Adding a blockchain transaction usually involves these steps: The sender produces a digital signature using their private key to identify and authorise the transaction. The sender then composes a digital message with the recipient's address, the amount of bitcoin or other assets to be transferred, and their digital signature for confirmation. The sender must publish the transaction to the blockchain network for all nodes to acknowledge it. Logging onto a network node, using a wallet, or using network-connected apps can do this. After receiving the transaction, these nodes validate it by verifying the sender's digital signature and funds. After validation, nodes must agree to accept the transaction.

2.1.2 Category

While making transaction public and transparent is what people strives for and the essence of blockchain, it becomes the noticeable obstacle to adoption for those people or organizations who care about their user personal privacy. Therefore, the development of blockchain is gradually evolving into two categories today: Permissionless blockchain and Permissioned blockchain:

- **Permissionless blockchain frameworks.** Also known as public blockchain, are open to anyone and do not require any form of authentication or authorization to participate in the network. The most well-known example of a permissionless blockchain is Bitcoin and Ethereum. In these networks, anyone can participate as a node, validate transactions and create blocks.
- **Permissioned blockchain frameworks.** As opposite to permissionless framework, known as private blockchain, are restricted networks where access is controlled by a central authority(CA) or a group of authorized participants. In these networks, only authorized participants are allowed to participate as nodes and validate transactions. Permissioned blockchains are often used for enterprise use cases, where a closed group of participants need to share sensitive data and need to be sure of who is accessing the network. Examples of permissioned blockchain include Hyperledger.

2.1.3 Hyperledger Fabric

Hyperledger Fabric [1] is an open source blockchain platform designed for enterprise use cases. It is one of the Hyperledger projects hosted by the Linux Foundation. One of its biggest advantages is that it provides a modular architecture, which allows for flexibility and customisation in the implementation of blockchain-based decentralised applications

[1] https://www.hyperledger.org/use/fabric

(dAPP). In addition, it is a permissionless blockchain framework with a membership service provider (MSP) mechanism, which allows only users or organisations that have permission to access the blockchain and perform transactions - it's like a shop that only accepts credit cards from a few banks.

It is composed of six key components:

- **Channels**: Hyperledger Fabric channels protect transactions. Channels enable a set of people to trade without other network members seeing the specifics unless authorised.
- **Smart Contracts (Chaincode)**: Smart contracts govern the network transactions. Chaincode that can be implemented in Go, Node.js, or Java, is used to implement business logic, access control, and more sophisticated operations.
- **Membership Services Provider (MSP)**: Membership services providers manage network participants' identities. MSPs authenticate and authorise subscribers to use the network.
- **Ledger**: The distributed ledger contains all transactions and network status. The ledger is copied to all network nodes to provide everyone a consistent representation of the network.
- **Node Types**: Hyperledger Fabric supports various types of nodes, including peers, ordering services and certificate authorities (CAs).
- **Consensus:** A consensus mechanism ensures that all nodes in Hyperledger Fabric agree on the ledger state. The platform offers pluggable consensus algorithms including Kafka-based, Raft-based and more.

2.1.4 Ethereum

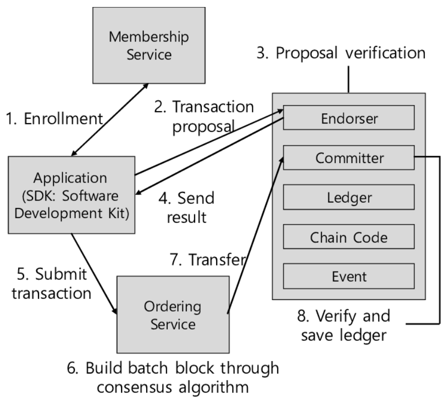
Ethereum [1], another open-source proposed by Vitalik Buterin in 2013 and started in 2015, decentralised blockchain system, supports smart contracts and dApps. It is a *world computer* means that it executes code on a wide-range of nodes spreading the world. Unlike Hyperledger Fabric, Ethereum is a permissionless network and anybody may observe and access the blockchain network without restriction.

It consists of seven key components:

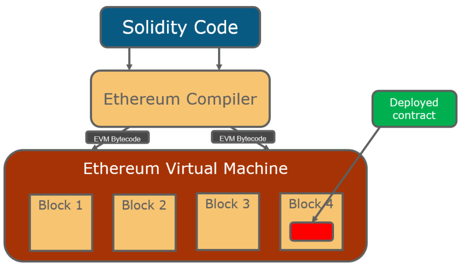
[1]https://ethereum.org/en/

- **Ethereum Virtual Machine (EVM)**: All Ethereum nodes run the EVM. It executes smart contracts and maintains the Ethereum blockchain.
- **Smart Contracts**: Smart contracts of Ethereum are self-executing agreements with terms encoded directly into the code. However, compared to Hyperledegr Fabric's highly customisable development environment, Ethereum only supports Solidity [1].
- **Gas**: Everything takes price, and Ethereum has no exception. Gas is an internal exchange for executing smart contracts on the Ethereum network. Each operation performed on-chain requires a certain amount of Gas to be paid in terms of Ether as long as it updates EVM states, which can effectively prevent malicious hacking.
- **Ether (ETH)**: Ether is the cryptocurrency used to pay for the execution of smart contracts and transactions on the Ethereum network.
- **Dapps**: Dapps are Ethereum network-based, decentralized applications. They are constructed on top of smart contracts and have a wide range of uses, including as in social media, gaming, and banking.
- **Solidity**: The Ethereum network uses the computer language Solidity to create smart contracts. For developers with experience in object-oriented programming, it is comparable to JavaScript and is intended to be simple to learn.
- **Consensus**: Ethereum adopted a consensus algorithm Proof of Work (PoW) in the past, a computationally demanding consensus process that forces miners to solve challenging mathematical problems in order to validate transactions and produce new blocks, was the consensus mechanism that Ethereum formerly employed. Because PoS is a more energy-efficient mechanism, Ethereum switched from PoW to PoS as its consensus method. With PoS, validators are selected to construct blocks based on the amount of bitcoin they have "staked," or locked up as collateral. As a result, validators have a financial interest in the network and an incentive to behave honestly. It is anticipated that the switch to PoS would make it more ecologically friendly.

[1] https://docs.soliditylang.org/en/latest/



(a) Hyperledger Fabric (20) assigns different duties to different nodes.



(b) Ethereum is a monolithic design in which all functionalities are hosted in the Ethereum virtual machine.

Figure 2.2: Comparison of Hyperledger Fabric and Ethereum architecture

2.2 Privacy-Preserving Technique - Zero-knowledge Proof

A major issue in the blockchain sector is privacy. The need to secure sensitive personal and financial data from unauthorized access and exploitation grows as more sensitive data is being kept on blockchain networks. However, owing to the public nature of transaction data on the blockchain, the usage of conventional blockchain systems might leave people and organizations open to data breaches.

We therefore explore a more intriguing privacy-preserving mechanism called Zero-Knowledge proof that is crucial in addressing these issues. This section provides the background for the later part, which introduces ZKP and explains how it can be used to enhance the privacy and security of blockchain-based systems.

2.2.1 Definition

ZKP is a method of proving the possession of some information, such as a private key, without revealing the actual information (21). It allows one party (the prover) to prove to another party (the verifier) that a statement is true without revealing any additional information. Specifically, it has a formal definition. Let's say there are two participants, prover P and verifier V. V is able to check if the result computed by P's program C is correct, which can be represented as y=C(x), it must fulfil two properties:

- **completeness**: P must prove the result y to V. If y is true, V can always believe it.

- **Soundness**: P cannot prove the result to V as long as y is false, except for small probability events.

2.2.2 Application

ZKP is utilized in the blockchain to give transactions anonymity and secrecy. All transactions on a public blockchain like Ethereum are accessible on the ledger, making it challenging to protect the privacy of sensitive data. Without disclosing the specifics of the transaction, ZKP can be used to demonstrate the authenticity of a transaction. This may be helpful when a user has to demonstrate their identification without disclosing any personal information, for example. We then explore two cryptography toolkits designed for blockchain frameworks:

- **Idemix**:

  Overview: Identity Mixer is an anonymous certificate solution proposed by IBM in 2009 that implements the underlying ZKP. The motivation behind this is that traditional X.509 adopted by Hyperledger is prone to the issue of over-exposure of all attributes, which can lead to the information leakage, because users have to present all attributes on the certificate during authentication. Therefore, anonymous authentication techniques are needed to minimise the exposure of user attributes. Idemix can solve the problem of over-exposure of information when the user presents the certificate in the traditional solution by allowing the user to selectively present the attribute information in the certificate (22).

  How Idemix works: As shown in Figure 2.3, the Idemix process requires three participants, namely the issuer, the user and the verifier. In the begining, the user or peer will generate a request for certificate and send it to issuer. Issuer then returns a certificate in form of Idemix credential containing user's attributes to user. If the verifier requires the user to present the certificate of attribute 1, the user can convert the certificate into a valid unlinkable token of any pseudonym of the user, containing only attribute 1 of the original credential and hiding other attributes. Verifying the token can be achieved through leveraging the CA's public key to check token validity.

  Architecture: MSPs from trustworthy authorities like CAs or tool *idemixgen* may validate a person or organization's identification in the Idemix architecture. MSPs employ features like organisation and position to build a unique digital identity for

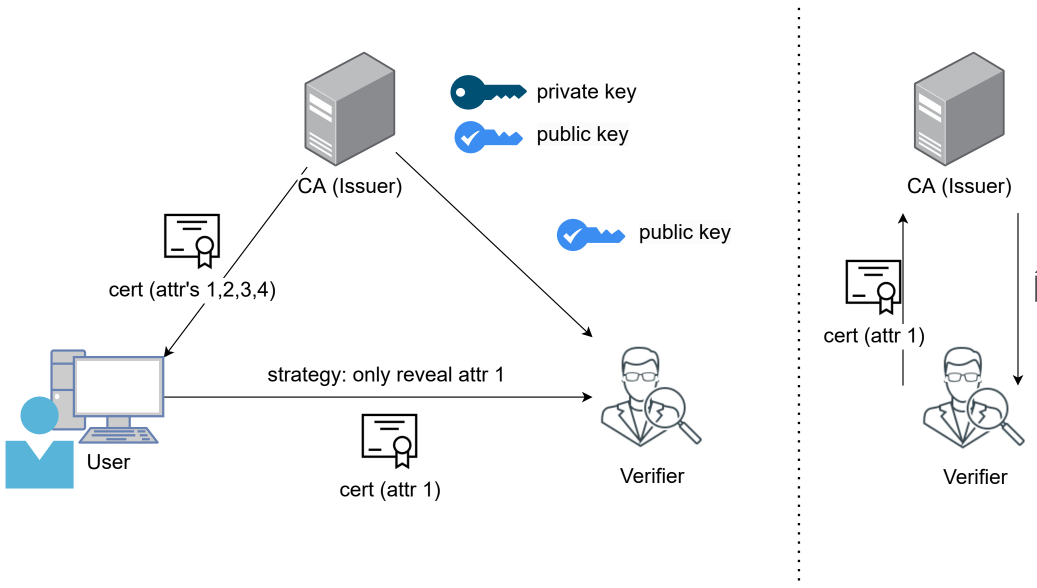


Figure 2.3: Idemix Overview

the person or organisation. These digital identities validate blockchain transactions without disclosing personal information.

Limitation: The limitation of Idemix, however, is that it is currently only available on the Hyperledger blockchain platform and is only supported by a specific Java SDK, making it incompatible with other blockchain networks.

- **Zokrates**:

  Overview: Zokrates is an open source toolbox including a language and related tools for instantiating ZKP on Ethereum. It allows developers to create zero-knowledge smart contracts on Ethereum that can perform computations on private inputs without revealing them.

  How Zokrates works: As shown in Figure 5.2, In the first step, the programmer uses the Zokrates language to encode an off-chain calculation. The program is compiled and produces a binary file that needs to be used with Zokrates CLI to generate the proof key and verification key. The next step is to use the CLI command to generate a Solidity verification smart contract, which contains the verification key that was generated earlier. In addition, the contract contains a verification function that receives the proof and returns the result of whether the proof is true or false.As a result, this contract can be used to validate users' proof but conceal the real value. More detailed explanation is discussed in Appendix 8.1.

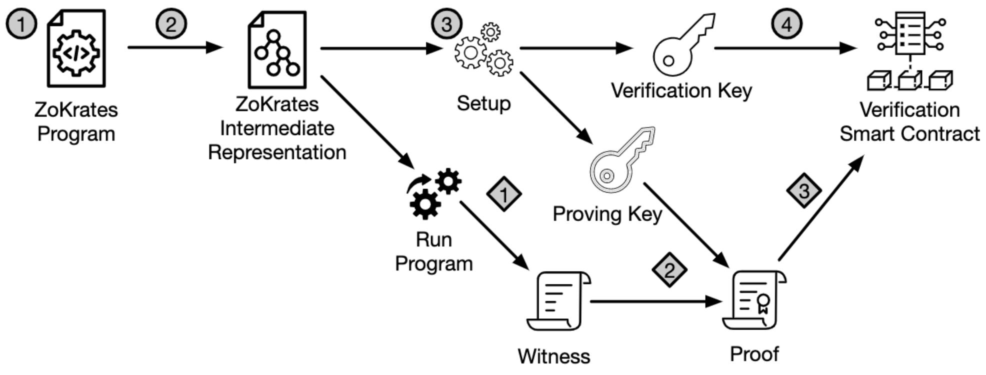


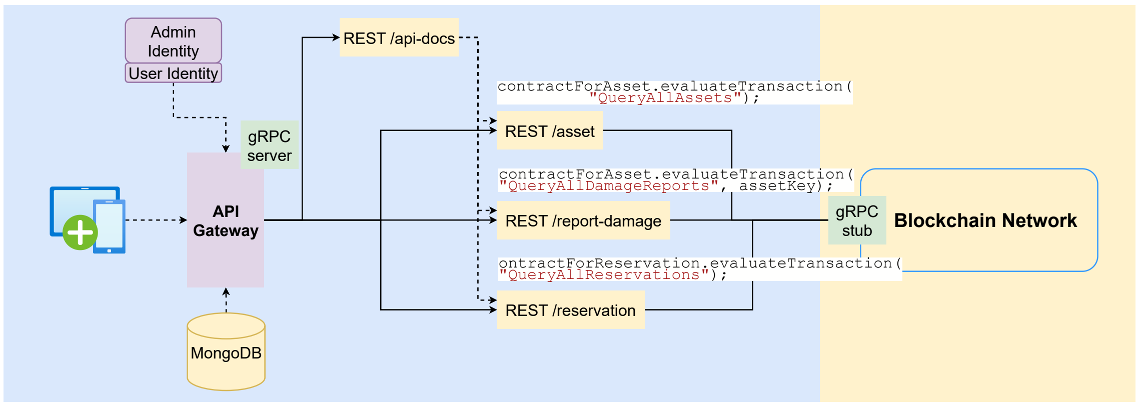
Figure 2.4: Major steps of Zokrates to generate the ZKP proof and witness and verification smart contract.

Architecture: Zokrates has a built-in language used to build a proof of computation that is confirmed on the Ethereum blockchain (23). ZKP technology lets Zokrates verify data without disclosing it. Hence, transactions and identities may be verified without violating privacy.

Limitation: The limitation of Zokrates is that it consumes extra computation resource, which leads to the increasing cost of ZKP off and on the blockchain.

2.3 Asset Lending Management System

We choose asset lending system as our underlying scenario because corporations often involve lending valuable assets, such as smartphones, laptops, hololenses and rasberries, to employees for business reasons. To protect company's benefit reducing the loss of public properties, an asset lending system is inevitable useful for employers to keep track of the asset status and find the responsible person in case happening unpleasant situations such as damaged or loss assets. The system should provide a safe and dependable capability to track and manage firm assets. However, compared to classic database-based management system, utilising blockchain technology can be an up-to-date alternative. Since the assets' status could be real-time updated and maintained by smart contracts that store them in a distributed ledger equally owned by every participants in the blockchain network. And credited to blockchain's immutable trait, committed transaction data could hardly be modify arbitrarily, instead, any modifications is permanently saved the ledger. Thus, company can safely monitor and check out who are are the owners of lent assets and what is the asset condition. The high-level view of our system architecture is illustrated in Figure 2.5, the back-end of the system can be powered by a various blockchain frameworks, while the front-end operates as an API-gateway that receives and processes transaction requests before passing them on to the blockchain. Additionally, it is responsible for managing user identities. The communication between both sides is based on the gRPC [1] protocol that can allow components developed by different tech-stacks to exchange information seamlessly. Build API gateway is to simplify gRPC endpoints offered by the blockchain.



```
contractForAsset.evaluateTransaction(
        "QueryAllAssets");
```

```
contractForAsset.evaluateTransaction(
"QueryAllDamageReports", assetKey);
```

```
ontractForReservation.evaluateTransaction(
"QueryAllReservations");
```

Figure 2.5: The asset-lending management system is mainly consisted of two parts, back-end and front-end.

2.4 Problem Statements

The implementation of ZKP in blockchain-based applications (dAPP) is gaining momentum, owing to the elevated security and privacy it offers. Yet, the ramifications of ZKP on the functioning of dAPP remain largely obscure, inciting the skepticism regarding to the optimal utilization of ZKP in blockchain-based systems. With the intent to bridge this knowledge gap, this research endeavors to shed light on the impact of ZKP on the performance among two mainstream blockchain frameworks — Hyperledger Fabric and Ethereum. By delving into the intricacies of how it affects the performance of various blockchain frameworks, the study aims to provide organizations and developers with generic but insightful information to make informed decisions on the deployment of ZKP in their systems, ensuring maximum performance and security.

[1]https://grpc.io/

2.5 Research Questions

According to the problems encountered with performance impact of ZKP on blockchain framework in terms of asset lending management system, the research questions defined are described as follows:

- **RQ 1**: Does ZKP have significant impact on the performance of a blockchain-based application?

  - The aim of this question is to probe into the influence of ZKP on the functioning of blockchain-based applications, as the integration of ZKP into blockchain frameworks is rapidly gaining popularity that incubates a number of cryptography tools such as Idemix, Zokrates, and Bulletproofs (24), credited to the heightened security and privacy it guarantees. Yet, the relationship between ZKP and the blockchain performance remains somewhat of a mystery due to lacking of a comparison of various frameworks to conclude a generic conclusion, leading to indecision about the best utilization of ZKP in these systems.

- **RQ 2**: Which blockchain framework has better performance with ZKP enabled?

  - The purpose of this question is to conduct a comparison of the performance differences between Hyperledger Fabric and Ethereum with ZKP enabled to see how the off-chain and on-chain operations will take effect. This study aims to determine which framework is more suitable for incorporating ZKP.

3

Related Work

This chapter provides a thorough summary of earlier research on the use of ZKP approaches, including a wide range of blockchain framework modifications. The objective is to comprehend how implementing ZKP would affect crucial performance indicators including transaction throughput, latency, and resource use. We can learn more about the trade-offs associated with deploying ZKP in blockchain systems as well as the possible effects on the overall functionality of blockchain-based apps by analyzing these indicators. By doing this, we offer a useful resource for academics and programmers exploring the area where blockchain technology and ZKP converge.

3.1 Hyperledger-Based Solutions

Li et al. introduce a blockchain-based ZKP technique-based identity verification system for ride-sharing platforms. The platform is built by the authors using Hyperledger Fabric, while the ZKP module is implemented using Hyperledger Ursa [1], which is developed using Rust and provides a set of APIs for use. By confirming users' identities without disclosing their personal information to the platform or other users, the proposed solution aims to increase ride-sharing safety and address trust and privacy issues. The authors assess the system's performance and demonstrate that Proof generation takes place off-chain on average in 39 ms while Proof verification takes place on-chain on average in 239 ms. Regular operation and ZKP verification cause an average transaction latency of 500 ms. Although the authors show how effective and secure the suggested method is, they do not directly compare it to other identity verification systems' performance. ZKP's effect on system performance may therefore rarely be inferred directly. Overall, the authors

[1]https://www.hyperledger.org/use/ursa

demonstrate the potential of ZKP in protecting privacy and enhancing the security of such platforms, which is an important contribution to the field of identity verification in ride-sharing systems (25, 26).

Similarly, Bai et al. propose a ZKP-based healthcare identity system called Health-zkIDM, built on the Hyperledger Fabric framework, to protect the identities of patients in various healthcare fields. The authors opt for client-chaincode separation, off-chain computation and on-chain verification paradigm, in the ZKP implementation with Go-snark, a low-cost computation-based zk-SNARK, to generate and verify the proof. The evaluation results show that the system provides efficient identity verification with an average verification time of 2.11 seconds and a total time of 3.16 seconds (27).

In general, the authors aim to measure the performance of the proposed system with ZKP feature enabled, where the different phase of ZKP computation time is measured and the average throughput with different transaction send rate is measured. However, due to the different implementation of the blockchain network with ZKP libraries, where the authors adopt an off-chain computation and on-chain verification paradigm that is contrast to our Hyperledger solution taking place all ZKP operations on-chain. Therefore, it is not the objective to make a direct comparison between two results, but we can draw a common empirical conclusion from both our studies that the overhead of ZKP is typically more than 500 ms.

3.2 Ethereum-Based Solutions

Gabay et al. present a privacy-preserving authentication scheme for connected electric vehicles (EV) using blockchain and ZKP. The authors aim to address the privacy concerns associated with sharing personal information in traditional centralised authentication schemes for electric vehicles. To do so, the proposed scheme uses blockchain to securely store EV data and ZKP to verify the authenticity of the data without revealing it to the authenticating entity. The evaluation results show that the proposed scheme is computationally efficient and outperforms traditional authentication methods in terms of privacy protection. The authors used Ethereum as the blockchain framework and Zokrates to generate witnesses and proofs. All operations except verification were performed off-chain, and the total time overhead was 37.60 seconds. The authors also measured the benchmark gas, cost fee, and time to compute the witness and generate the proof. The proof was generated using SHA256 and took 15 seconds (28). Given that real-world EV charging is not a heavy workload scenario, the authors conclude that the cost of ZKP is affordable and can be used effectively to protect privacy in connected vehicle networks.

Another study tries to address privacy issues with conventional digital identity management system (DIMS), Yang et al. introduce a blockchain-and-zero-knowledge proof based DIMS (BZDIMS). The suggested system also implements ZKP procedures through Zokrates on the Ethereum to enable users to authenticate their identity without disclosing their personal information to authorized parties. The approach enables secure and effective digital identity management, according to the authors' security, efficiency, and privacy evaluations. On the private blockchain, the scheme was estimated to have a verification proof function throughput of about 170 TPS, compared to 15 TPS on the public blockchain, where the lower throughput is caused by the consensus process. According to the authors, the effectiveness of Zokrates is not satisfactory, but the overall performance of the proposed scheme is still feasible (29).

Overall, the existing studies of Ethereum-based applications with ZKP are adapted to low-load scenarios, such as EV charging and DIMS, the additional computational overhead of ZKP is under-tolerated. Moreover, since most Ethereum studies rely on Zokrates to implement anonymous and unlinkable transactions, where the part of ZKP pressure is distributed to clients, it is helpful to reduce the blockchain network load. In our study, we also benchmark the Ethereum network with ZKP under high pressure, while only measuring on-chain performance metrics, which also show a similar tendency that smart contract functions with the proof verification function aren't affected.

4

Design

In this chapter, we present a high-level design logic behind the blockchain-based asset lending management system and the benchmark workflow. As the research aims to investigate a general impact of ZKP on blockchain frameworks, we implement the same application context on two mainstream frameworks, namely Hyperledger and Ethereum. Because the constraints and different underlying infrastructure frameworks, we need a separate design for each. In addition, the derivative tools and development ecology, such as the benchmark tool, software development kit (SDK) and documentation, are maintained by different communities, which also requires us to choose different tools and languages to implement benchmark workflows.

4.1 Blockchain Network Design

4.1.1 Hyperledger Fabric

We have set up a Hyperledger network consisting of two separate organisations, which can be seen in Figure 4.1. Each organisation has two peer nodes, two Certificate Authority (CA) nodes, and two CouchDB nodes to store the ledger data generated by the peer nodes. The network also includes an ordering node and a command line interface (CLI) node that receives and executes user transactions and commands. In addition, a chaincode node, responsible for executing and maintaining the smart contract logic, is created on-the-fly via the CLI node.

By having a network with two organisations, we have simulated a scenario where only users with identities from the owning organisation have access to the blockchain. This protects ledger transactions and data. The CouchDB nodes ensure that the ledger data is stored persistently and is readily available for querying. The ordering node helps maintain

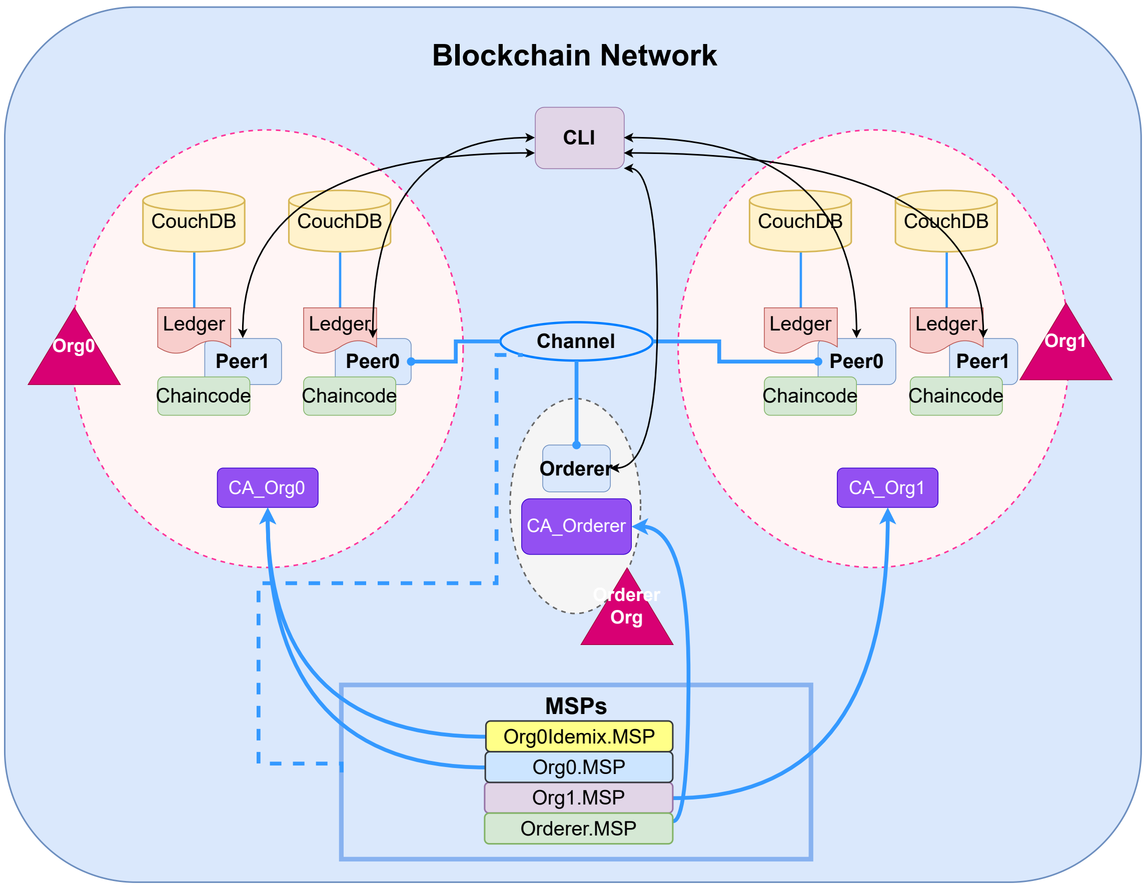


Figure 4.1: The topology of a 2-organisations (the orderer organisation is not included) Hyperledger Fabric network

the order of transactions, and the CLI node provides an interface for users to interact with the network and deploy chaincode.

4.1.2 Ethereum

In our study, we choose to build a single node private Ethereum network with clique consensus mechanism, which is shown in Figure 4.2. In contrast to the very free configuration of Hyperledger, which requires the user to assign and maintain peer, cli, orderer, chaincode and DB nodes in the network, the configuration of Ethereum only needs to deal with an execution client called *geth* [1]. This client, acting as an Ethereum node, combines the main functionalities of the various Hyperledger nodes mentioned above, such as transaction handling and gossip, state management and support for the EVM. In addition, geth provides

[1]https://geth.ethereum.org/

RPC methods so that users are able to query and perform transactions using libraries such as Web3.js [1]. Since the study focuses on security and privacy, we switch to a private Ethereum network that doesn't connect to the main network and uses a rather special consensus mechanism called Clique (30), a proof-of-authority (PoA) algorithm where only verified signers can create new blocks, and the signer can also be replaced by other users through voting. Therefore, it requires less resource consumption compared to PoW and is suitable for private and test networks.

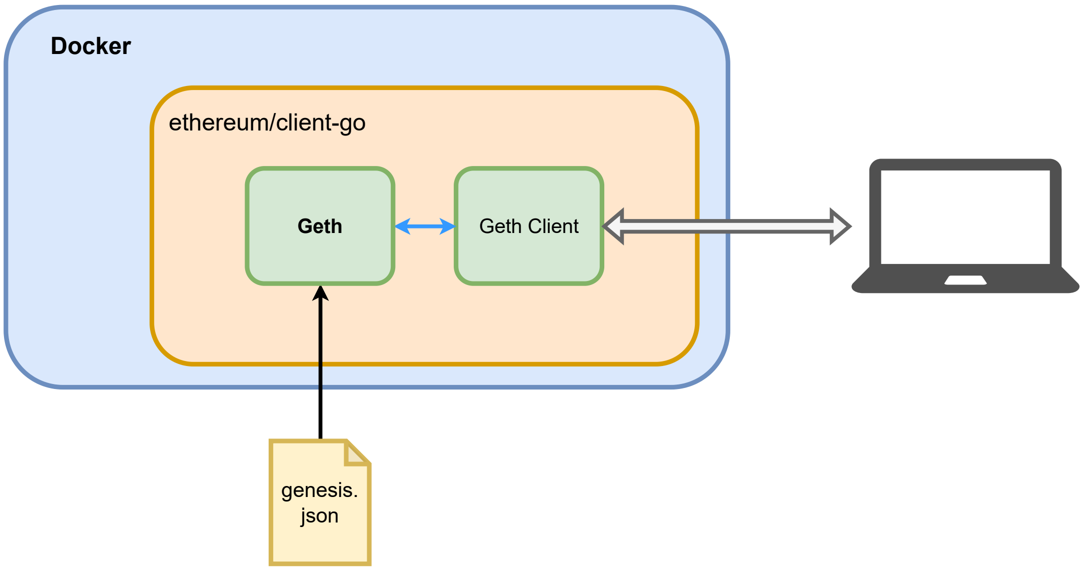


Figure 4.2: The topology of a 1-node private Ethereum network

4.2 Benchmark Workflow Design

4.2.1 Hyperledger Fabric

To effectively benchmark the performance of the Hyperledger Fabric framework, we designed a benchmark workflow consisting of three main components: a load generator using JMeter [2], a Fabric client implemented using Fabric-SDK, and a backend blockchain network containing the chaincode and ledger.

The load generator, JMeter, will simulate transactions and send them to the Fabric client. To evaluate the performance of the system, we added synchronous listeners to JMeter to monitor key metrics such as average response time, standard deviation of metrics, and

[1]https://web3js.readthedocs.io/en/v1.8.2/
[2]https://jmeter.apache.org/

transaction throughput. In addition, JMeter supports third-party plug-ins [1] that allow us to collect resource usage data from the host machine during the experiment, including CPU and memory usage. This data will allow us to determine the impact of ZKP on blockchain performance.

The Fabric client is a critical component of the benchmark workflow, as it provides two important functions. Firstly, it enables the registration and enrolment of new users and administrators to the blockchain, which is necessary to grant permissions to access the blockchain and process ZKP transactions. Second, the Fabric client abstracts the logic of the blockchain and provides simplified backend APIs for front-end users to interact with.

4.2.2 Ethereum

We will utilize Caliper [2], an open-source tool that offers a standardized way to test the performance of various blockchain systems, to benchmark the performance of a single-node private Ethereum network. Caliper works by submitting a set amount of transactions to the blockchain and tracking how long it takes for those transactions to be completed. The goal of Caliper is to simulate a real-world scenario and use that simulation to gauge a blockchain network's performance. Three input files—*a benchmark configuration, a network configuration, and a workload configuration*—must be given to Caliper before an experiment may be executed. These files specify the quantity of transactions to transmit, their size, and the amount of time between transactions, as well as the topology and network certification necessary for the tool to generate the load and gain access to the blockchain. Caliper will connect to the geth container and publish the required amount of transactions to the network in our single node private Ethereum network. The time it takes to complete each transaction will then be measured by Caliper, which will also offer important performance data including transaction throughput, latency, and resource use. Caliper offers extensive statistics on the network's behavior in addition to these performance indicators, including the typical processing time per transaction and the typical resource usage.

[1] https://github.com/undera/perfmon-agent

[2] https://hyperledger.github.io/caliper/

5

Implementation

In this chapter, we present a thorough implementation of the benchmark procedures described in Chapter 4. Our approach concentrates on the supporting infrastructure's underpinning elements, such as transaction processing and application context. We go over the application's setting and implementation. We also describe two approaches to deploying and configuring ZKP in two different blockchain frameworks. Additionally, we discuss the benchmarking tools we have selected and the setup process. Lastly, we introduce the assembling of the benchmarking tool, blockchain middleware, and blockchain network. This chapter contains a workflow implementation guide and process component justifications.

5.1 Infrastructure

5.1.1 Hyperledger Fabric

We create a shell script to automatically launch the Hyperledger network. The script starts by building a docker image named `fabrichost_sample` from the official Golang image. This image acts as the host for the network and downloads the necessary Hyperledger binaries and images from the official repository and registry. As shown Figure 5.1, the sequence workflow demonstrates the main steps of the script:

- The script generates crypto files using the *cryptogen* binary for the CA and organization,
- It launches the CA node to generate Idemix keys for ZKP and creates channel artifacts using *configtxgen.*
- The script uses a docker-compose file to start peer, CA, CLI, orderer, and database nodes.

- After waiting 3 seconds for the containers to start, it invokes the CLI node to create the channel, install and instantiate the chaincode on the channel.

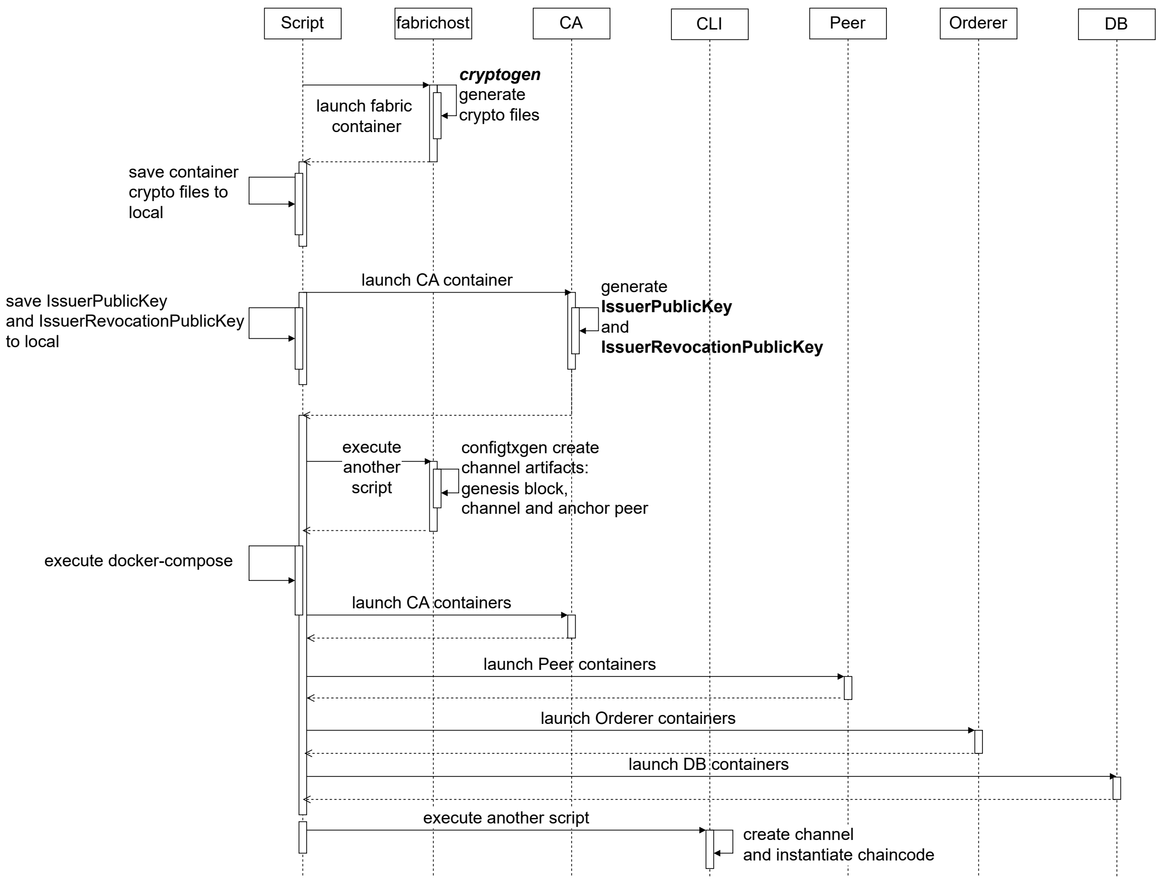

Script
fabrichost
CA
CLI
Peer
Orderer
DB
launch fabric container
cryptogen generate crypto files
save container crypto files to local
launch CA container
generate IssuerPublicKey and IssuerRevocationPublicKey
save IssuerPublicKey and IssuerRevocationPublicKey to local
execute another script
configtxgen create channel artifacts: genesis block, channel and anchor peer
execute docker-compose
launch CA containers
launch Peer containers
launch Orderer containers
launch DB containers
execute another script
create channel and instantiate chaincode

Figure 5.1: The sequence flow of the script to launch Hyperldger Fabric network

5.1.2 Ethereum

Compared to the complex setup of Hyperledger, creating an Ethereum network is much simpler. We leverage docker-compose to bring the network to life and also create a Dockerfile to build the network image. The image is based on the official Ethereum client-go image [1], which includes an execution client geth and all the necessary dependencies and environment. The Dockerfile also sets the options for geth to start the network with a specific configuration, where we are going to create a one-node private network. For efficiency and energy consumption consideration, the configuration adopts the Clique PoA

[1]https://hub.docker.com/r/ethereum/client-go

consensus mechanism and sets up a pre-defined account to sign new blocks (30). Finally, the docker-compose file starts a container running geth, which exposes a port to interact with the private Ethereum network.

5.2 Zero-Knowledge Proof

5.2.1 Idemix in Fabric

Because Hyperledger Fabric natively supports ZKP during authentication to prevent transactors from overly exposing their personal attributes when signing transactions with traditional X509 (31) authentication, which exposes all attributes to other users, we only need to take additional steps during Fabric network configuration to enable ZKP. Hyperledger implements ZKP through a cryptographic protocol suite called Idemix, as introduced in the Chapter 2. To enable ZKP, we simply need to add an Idemix MSP configuration of an organisation that needs to hide its identity to *configtx.yaml*, which will take effect when the blockchain network is spun up. In our experiment, we enable Organisation 0 to issue an anomynous transaction to Organisation 1, where the Org0Idemix MSP verifies the ZKP proof sent by Organisation 0. To register and enroll an Idemix user identity, the developer only needs to use a single API provided by the `fabric-sdk-java`, which can store the credential and interact with the Fabric network. In addition to the Idemix MSP, Chaincode can also act as a verifier by checking the attributes of the credential. Since only the Idemix credential can support checking the specified attributes, using *x509Enrollment* to propose a transaction will otherwise fail. We provide a detailed code example and *configtx.yaml* configuration in Appendix 8.2.

5.2.2 Zokrates in Ethereum

We implement ZKP on the Ethereum network through Zokrates, which allows only legitimate users to generate proofs. Zokrates is a tool for creating zkSNARKs, a type of ZKP that offers several advantages over traditional implementations, such as constant verification time, a constant-size proof regardless of the complexity of the problem, and the ability for the prover to prove knowledge with a single message. Compared to Idemix's high-level abstraction, Zokrates furnishes a finer control to generate witness and proof. As a result, implementing ZKP in Ethereum requires more work, which is depicted in Figure 5.2. To generate a proof of knowledge for a secret function using Zokrates, we first create a .zok file that describes the question and data to be hidden. The function checks if a

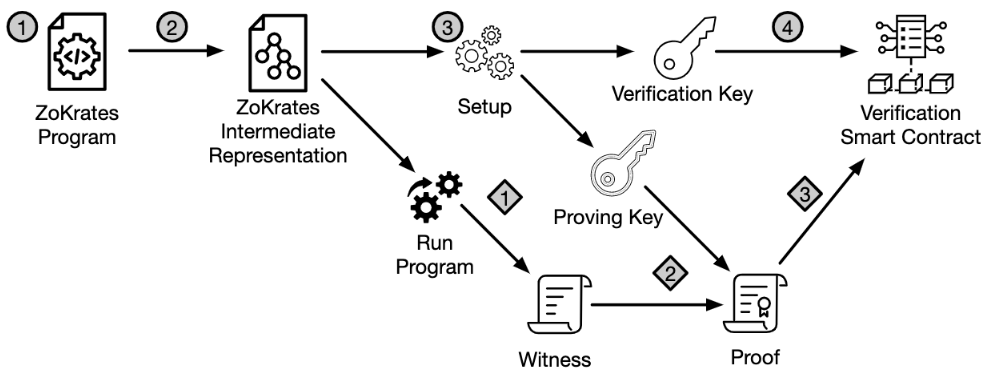


Figure 5.2: Major steps of Zokrates to generate the ZKP proof and witness and verification smart contract.

given ID belongs to a company or organization by verifying the prefix and suffix against a hash value. The function returns a boolean value based on the condition. The next step is to compile the .zok file to get an executable binary that can be used to compute a witness for the compiled program. The witness is a proof of the prover's knowledge of the secret function. Zokrates generates a proof key and a verification key, which are stored in *verification.key* for later use. Then, we provide the witness and proof key as parameters to Zokrates, which generates a *proof.json* consisting of two parts, proof and input. Zokrates also outputs a verification smart contract written in Solidity, which can be deployed on Ethereum to verify the truth of the proof.

5.3 Benchmark Setup

5.3.1 Hyperledger Fabric

5.3.1.1 Component Justifications

Initially, we found that benchmarking a Hyperledger network was more challenging than we first thought. Specifically, we were using Caliper, but it does not support the `fabric-sdk-java`, only the `fabric-sdk-node`, the only SDK that can support Idemix features. To overcome this limitation, we turned to Apache JMeter, which supports benchmarking for most languages and frameworks. We wanted to build a REST API server with `fabric-gateway-java` [1] as a client to communicate with JMeter. Unfortunately, a more serious problem with `fabric-gateway-java` is that it does not support storing Idemix credentials in the wallet that the gateway instance relies on during initiation to establish connection with the Fabric

[1]https://github.com/hyperledger/fabric-gateway-java

network. Therefore, we decided to implement a gPRC API server instead, as the gRPC protocol underlying `fabric-sdk-java` allows the Fabric client to interact directly with the blockchain, such as creating a transaction proposal request and sending the request to the blockchain.

Feature	`fabric-gateway-java`	**fabric-sdk-java**
Purpose	High-level API for interacting with Hyperledger Fabric network	Low-level API for building blockchain-based applications and interacting with Hyperledger Fabric network
Abstraction	Abstraction of network communication, transaction committing and event listening	Low-level interfaces for direct communication with a Hyperledger Fabric network
Ease of use	Easy to use	Requires more code to perform the same actions
Flexibility	Limited flexibility due to high level abstractions	Greater flexibility and control over network

Table 5.1: Compare `fabric-gateway-java` and `fabric-sdk-java`

To do this, we abandon the `fabric-gateway-java` library instead of using `fabric-sdk-java`. The difference between them, as shown in Table 5.1, is that the former is can be regarded as a subset of the latter and offers higher abstraction of Fabric with simple configurations, capable of maintaining the user's identity and reading from the local or in-memory wallet to create a gateway instance to constantly connect to the network, which the developer can call from anywhere in the application. In contrast, the latter gives developers more flexibility and control over how they interact with the Fabric network, and doesn't rely on the wallet to maintain and import the user identity, which can bypass the `fabric-gateway-java` restriction and create transactions directly with Idemix credentials. Lastly, since we require the fabric client to constantly listen to JMeter's external gRPC requests, we need a host environment with both a fabric client and a gRPC server to relay the workload to the client. Finally, we construct middleware using Spring Boot [1], which is a popular Java framework used to simplify the development process.

5.3.1.2 Solution

As shown in Figure 5.3, our benchmark workflow consists of three parts, the JMeter is responsible for generating the workload, the middleware is a Fabric client hosted in the Spring Boot framework, and our backend is a two organisations Fabric network.

To allow gRPC requests in JMeter, we must install a third-party plugin named jmeter-grpc-request[2]. The plugin can test any gPRC server including our middleware, the request will be issued from the gRPC stub. Then we create a *test plan* in JMeter and add a

[1]https://spring.io/

[2]https://github.com/zalopay-oss/jmeter-grpc-request

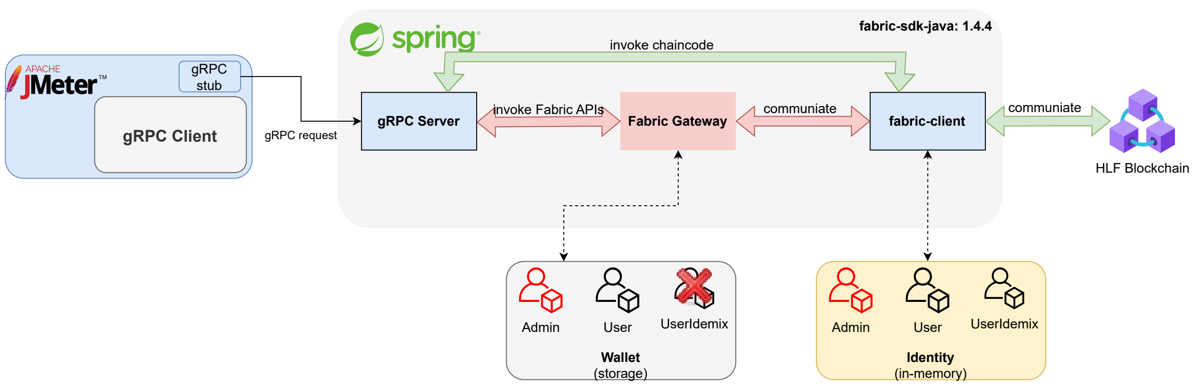


Figure 5.3: Hyperledger Fabric Benchmark Workflow

thread group specifying the number of agents to simulate users sending requests and further configuration. Next, we add a *sampler* with the type of gPRC request to the thread group that will generate the real workload. To capture key metrics such as latency and throughput, we add *listeners* to the sampler, which will listen and format the results. Furthermore, the underlying elements of the gRPC protocol are *service* and *message*, which are generic schemes that define the function passed to the request body and the structure of the request and response bodies. Therefore, in order for the gPRC sampler to successfully produce a workload, we need to specify the directory of a .proto file that defines the service and message. The code can be found in Appendix 8.2.

According to the definition, the RPC method `QueryBlockchain` appears in both the stub and the server, taking as parameter a `BlockchainRequest` message containing a `AppUser` message from the stub and receiving a `BlockchainResponse` message from the server. Next, we use the proto compiler *protoc*, which takes the proto definition and the specified language (Java, which we used in this study) as input, to generate code containing classes and methods that correspond to the messages and services defined in the .proto file. We can then use these generated files in our middleware application along with the gRPC library to implement the server components for communicating using gRPC.

5.3.2 Ethereum

Because Zokrates, which we use in Ethereum, generates witness and zero-knowledge proofs off-chain, integrating it into our application does not affect the original code dependencies, and there is no need to change the architecture of the code to accommodate it. We were therefore able to use Caliper to measure the performance of the blockchain network. As shown in Figure 5.4, the workflow consists of two components: the Caliper itself and the Ethereum network. The Caliper is an easy-to-use tool that requires three configuration files, after which it can automatically run the experiment and output a benchmark report.

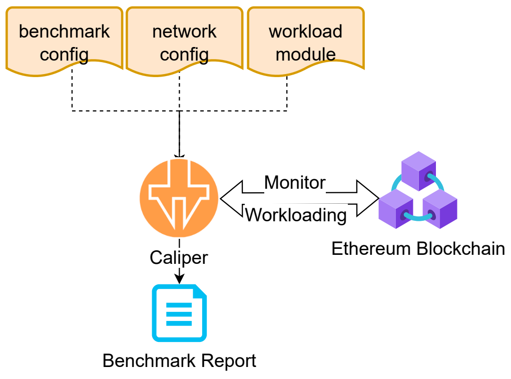


Figure 5.4: Ethereum Benchmark Workflow

First, we need to implement a `networkconfig.json`, which is processed by Caliper and shown in Appendix 8.2, where we define not only the network configurations that allow Caliper to start and connect, but also the account and smart contract references that give Caliper access to interact with the blockchain. This is also the entry point for Caliper to run a benchmark, as we run the command `npx caliper launch manager` followed by the config files, the Caliper would invoke the docker-compose binary to launch the single node Ethereum private network locally, and invoke the other docker-compose file to terminate the whole network when finished. Then the caliper connects to Ethereum via the provided URL and ports and account to generate the workloads defined in `benmarkconfig.yaml`, which acts as a coordinator that specifies the number of workers issuing workloads in parallel to the network to simulate real user behaviour, and defines the execution order and rate. This file can be used to design fine-tuned experiments.

6

Experiment

In this chapter, we describe the experimental strategy and the findings. Given the different types of blockchain frameworks we used and the concurrently supported benchmark tools that provide different performance metrics, we therefore conduct groups of control experiments separately to evaluate the impact of ZKP on the performance of blockchain-based applications and observe which framework representative. To obtain the performance impact of ZKP, we need to design a simulated workload for both frameworks. We also need to fine-tune the parameters for the benchmark tools separately, in order to properly utilise the resources of the host machine. The evaluation of blockchain performance is embodied in a variety of factors, so we need effective metrics to quantify the performance impact of the two types of ZKP tools. The two group outcomes must be compared after the control experiments.

6.1 Design of Experiment

6.1.1 Setup

6.1.1.1 Hyperledger Fabric & JMeter

The environment for the experiment is set up by installing and configuring a 2 organisations Hyperledger Fabric network with and without *Idemix* and JMeter on the same machine, but hosted in different Docker containers to ensure resource isolation, which is shown in Figure 6.1. The configuration is shown in Table 6.2, where the *Average Latency* and *Transaction Throughput* performance metrics are measured under the same load conditions as the *Summary Report* and *Transactions per Second* metrics. JMeter monitors configured with `5 Threads` to simulate users at maximum send rate and `200 Loop Count` to limit the thread to initiate 200 transactions, thereby JMeter will send a total of 1000 transactions to the blockchain for each function. To avoid overloading the system, we set a `1 second ramp-up time` for the thread group so that there is a time gap between each thread when it is cold started, giving it time to gradually increase the load and ensuring that the performance measurements are accurate. The data collected is analysed and compared to observe the impact of ZKP on network performance.

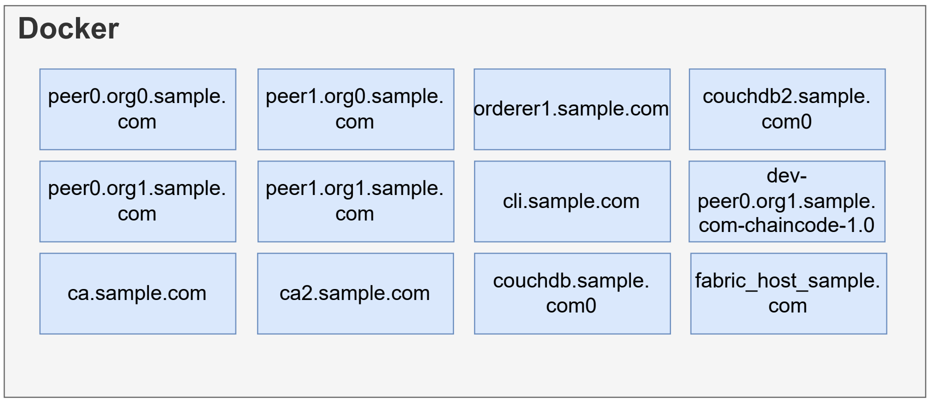


Figure 6.1: 2 Organisation Hyperledger Fabric network container overview

In addition to transaction performance, we also collect resource usage during the experiment to investigate the system load situation, using the JMeter plugin *PerfMon* (Server Performance Monitoring) [1] to set up two monitors to collect CPU and memory metrics. We select two commonly used functions, `createAsset` and `queryAllAssets`, from the smart contract to benchmark their performance. Since we are conducting a control experiment as shown in Table 6.1, we keep all the variable factors except for the identity credential being the same, which is used to create and issue transaction requests from the Fabric client. The independent variable thus can be either X509 credentials or Idemix credentials introduced in Chapter 2.2.2 to observe the performance effect of ZKP. Idemix automatically hides the identity of the actor, and verifies the required attributes at the Idexmi MSP, all in the background as long as the Idemix credential is used. We thus can control the variable easily.

6.1.1.2 Ethereum & Caliper

This experimental setup uses Caliper to benchmark Ethereum. The goal is to measure the performance of Ethereum under the same workload conditions as the Hyperledger Fabric experiment, by running two rounds of tests with 1000 transactions for each round. To

[1]https://github.com/undera/perfmon-agent

Group	Independent Variable	Dependent Variables
Control	Issue transactions with Idemix credential	JMeter configuration, underlying infrastructure
Experimental	Issue transactions with x509 credential	JMeter configuration, underlying infrastructure

Table 6.1: Hyperledger Fabric control experiments setup

Parameter	Value	Description
Number of Threads	5	#simulated user
Ramp-up-Time	1s	Time gap between each thread while starting to avoid a spick
Loop Count	200	Maximum #transaction per thread
Listener	TPS, Summary Report, PerMon Metrics Collector	Measure needed metrics: throughout, avg latency, cpu and memory

Table 6.2: JMeter Configuration

control the independent variable, we keep all variable factors constant while creating two workload configuration files that not only specify the experiment specification, but also pass the proof and witness parameters to the test functions to enable the ZKP feature. The experiment follows control experiment settings in Section 6.1.1.1, where we use 5 worker processes and design 2 test rounds, each benchmarks a single function:

- 1st round executes 1k transactions using the *createAsset* gRPC method with a maximum rate load control, and leave transaction rate to 100 at the start, ramping up to a maximum rate of 20 transactions per second, we also let `sampleInterval` to 1 second to ensure that the transaction rate increases steadily.
- 2nd round runs the *queryAllAssets* gRPC method with the same rate control configuration with 1st round.

Finally, we enable the Docker monitoring module in Caliper to collect Geth container resource usage, which includes average memory and average CPU. The complete list of Caliper configurations can be found in Table 6.3.

6.1.2 Evaluation Metrics

In order to assess the impact of ZKP on the performance of blockchain frameworks, it is necessary to measure various metrics and statistical indicators to evaluate the functioning

Label	Description	Transactions	Rate Control	Workload
create Asset	create a new asset through the deployed contract	1000	tps: 100 step: 20 sampleInterval: 1	benchmarks/scenario/simple/create.js
query All Asset	query all assets through the deployed contract	1000	tps: 100 step: 20 sampleInterval: 1	benchmarks/scenario/simple/queryAll.js with arguments numberOfAsset: 1000

Table 6.3: Caliper Configuration

of each blockchain system in question. Table 6.4 shows that latency is a crucial statistic for measuring blockchain transaction processing time. It determines blockchain network responsiveness and efficiency.

Metric	Unit	Descriptive Statistics
latency	millisecond	mean, standard deviation
throughput	# transactions per second	mean, standard deviation
error rate	percentage	mean
CPU user	percentage	average, maximum
memory	megabyte	average, maximum

Table 6.4: Evaluation metrics and descriptive statistics

Under most circumstances, lower latency means quicker transaction processing. For comparison between control and experimental groups, latency is evaluated in milliseconds with metrics of mean and standard deviation. Throughput, another important metric, measures the number of transactions processed per unit of time and provides valuable information about the capacity of the blockchain network. A higher throughput value is preferable in most use cases, as it indicates a higher transaction processing rate. Throughput is typically measured in transactions per second, with its mean and standard deviation calculated. The error rate metric measures the percentage of transactions that are not processed correctly. Lower error rate values indicate a more reliable blockchain network. Error rate is typically measured as a percentage, with the mean calculated.

CPU usage shows how efficiently the blockchain network uses CPU resources. Calculate average and maximum CPU utilisation percentages. Memory consumption, on the other side, measures the blockchain network's memory usage and efficiency. Megabyte memory consumption averages and maxes. By calculating and analysing these metrics for comparison between the control and experimental groups, it becomes possible to evaluate the impact of ZKP on the performance of the blockchain frameworks and identify the framework that outperforms the others under the ZKP function. The mean and standard deviation of each metric must be calculated and compared between the control experiment (without ZKP enabled) and the experimental group (with ZKP enabled) for both Hyperledger and Ethereum.

6.2 Experiment Result

6.2.1 Hyperledger Fabric

The experiments were conducted to evaluate the impact of ZKP on the performance of the Hyperledger Fabric blockchain framework. Four different operations were tested: query without Idemix credential, query with Idemix credential, add without Idemix credential, and add with Idemix credential. The results showed in Table 6.5, where two different scenarios: with and without ZKP for two functions: *queryAllAssets* and *addAsset* are placed. we could obeserve that the ZKP feature had a significant impact on the performance of Hyperledger Fabric.

Label	# Sample	Avg	Min	Max	Std. Dev	Err %	Throughput	Recvd KB/sec	Avg Bytes
queryAllAssets_no_zkp	1000	82	37	168	20.43	0.00	55.87	127.73	2341.00
queryAllAssets_with_zkp	1000	**591**	0	**1003**	**153.68**	**0.04**	**8.00**	**17.52**	2241.83
add_no_zkp	1000	23	11	80	8.91	0.00	158.28	9.12	59.00
add_with_zkp	1000	**341**	0	**1005**	**332.97**	**0.45**	**9.90**	**0.68**	70.81

Table 6.5: The results of a control experiment for Hyperledger Fabric benchmark with descriptive statistics.

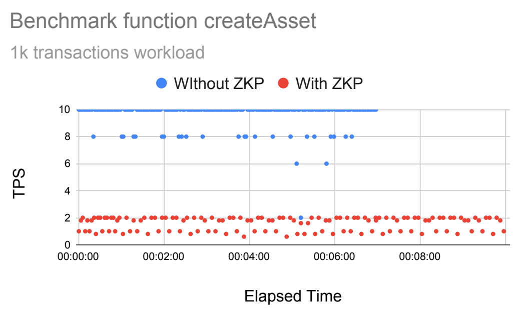
As operations with ZKP had significantly higher latencies and lower throughputs compared to the operations without ZKP. The query operation with ZKP had an average latency of 591 ms, while the query operation without ZKP had an average latency of 82 ms, which is 7 times higher. The add operation with ZKP had an average latency of 341 ms, while the contrary operation had an average latency of 23 ms.

For the throughput of the query operation with ZKP, it was much lower at 8 transactions per second compared to 55.87 transactions per second for the one with ZKP. Meanwhile, the add operation with ZKP had a lower throughput of 9.9 transactions per second compared to 158.28 transactions per second for the add operation without ZKP, hence we notice that the average difference is 7 to 15 times. Finally, the error rate of all operations was low, with the query operation with ZKP having the highest error rate of 0.04% that is negligible. Therefore, these results indicate that the operations without ZKP outperformed the operations with ZKP.

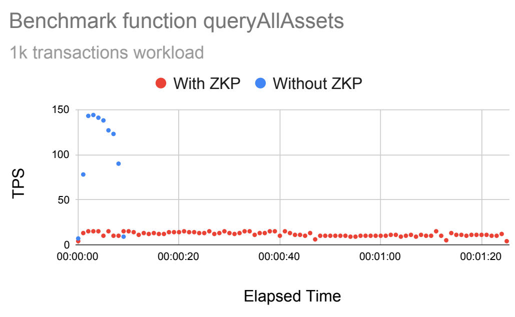
We can also see Figure 6.2 (a) and (b), which are two scatter plots illustrating the comparison of Transaction Processing Speed (TPS) over elapsed time between two series: red with ZKP and blue without, with 1000 total transactions in the control experiment. With the ZKP function enabled, it is evident that both scenarios take longer to send the same number of transactions. Additionally, we can observe that *createAsset* takes nearly 10 minutes compared to 1 minute and 20 seconds spent by *queryAllAssets*. This is because creating an asset involves a more computation-intensive operation in blockchain, which requires writing operations to create new blocks and update ledgers every time. Despite this, it is noteworthy that enabling the ZKP feature would result in a performance loss ranging from 30% to 87.5%. Similarly, Figure 6.2 (c) and (d) are two area charts, which show the comparison of resource consumption over time of the same functions with and without ZKP, different colours representing different metrics. We can see a similar trend that the group without ZKP saves more resource consumption by finishing earlier.

6.2.2 Ethereum

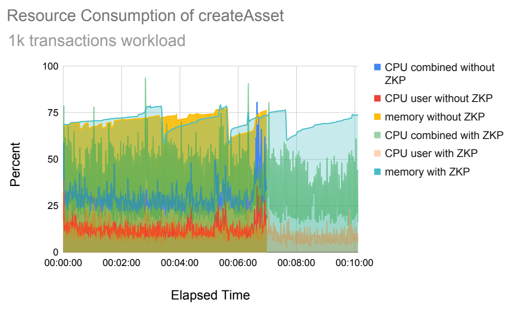
The objective was to test Ethereum's performance and resource usage with and without ZKP, just as earlier studies. Tables 6.6 and 6.7 show Ethereum benchmarking results. In terms of performance, we can see that the throughput of the create operation with and without ZKP is almost the same, while the query operation with ZKP shows a reduction of less than 24 percent. The three latency matrices of Ethereum with and without ZKP, however, barely differ from one another. Table 6.7 demonstrates that Ethereum with ZKP (createAsset_zkp and queryAllAssets_zkp) consumes a little bit more CPU and memory than Ethereum without ZKP (createAsset_no_zkp and queryAllAssets_no_zkp). The additional calculation necessary to verify ZKP could be the cause of the increase in CPU and memory usage. Therefore, due to the separation computation design of Zokrates that only verification is handled on-chain, the performance impact of ZKP on the Ethereum could be negligible.



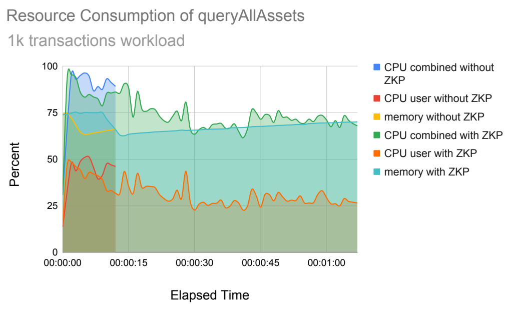
(a) Results of createAsset elapsed time versus transaction per second ZKP performance comparison results



(b) Results of queryAllAssets elapsed time versus transaction per second ZKP performance comparison results



(c) Results of createAsset elapsed time versus resource consumption ZKP performance comparison results



(d) Results of queryAllAssets elapsed time versus resource consumption ZKP performance comparison results

Figure 6.2: Control experiment results of Hyperledger Fabric

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
createAsset_no_zkp	1000	0	92.2	41.34	1.18	20.67	20.1
createAsset_zkp	1000	0	90.5	38.48	0.79	17.43	21.5
queryAllAssets_no_zkp	1000	0	124.3	0.02	0	0	124.3
queryAllAssets_zkp	1000	0	94.5	0.04	0	0	94.5

Table 6.6: Descriptive Statistics of Ethreum Benchmark

Name	CPU%(max)	CPU%(avg)	Memory(max) [MB]	Memory(avg) [MB]
createAsset_no_zkp	0.7	0.7	360	360
createAsset_zkp	0.81	0.81	387	387
queryAllAssets_no_zkp	0.17	0.17	440	440
queryAllAssets_zkp	1.14	1.14	448	448

Table 6.7: Descriptive Statistics of Resource Consumption During Ethereum Benchmark

7

Discussion

In this chapter, we summarise the experimental results under that context of asset-lending management system, meanwhile, we reflect the limitations of the proposed benchmark solution in this study and describe the aspects that can be improved in the future work.

7.1 Discussion of Experiment Results Within the Context

The trials' findings lead us to the conclusion that the performance of the blockchain-based application may be significantly impacted by the adaptation of ZKP. In the case of Hyperledger Fabric, for instance, the query all assets and add a new asset activities with ZKP had a much greater latency and poorer throughput than the operations without ZKP, showing that the usage of ZKP might possibly slow the system down. The studies with Ethereum, on the other hand, demonstrated that ZKP had a negligible effect on performance and resource utilization and that the extra processing needed to produce the ZKP proof and witness could be handled off-chain. Therefore, if the client side is not taken into account, we may argue that ZKP's performance effect on the Ethereum asset loan management system may be minimal.

However, if the burden of the asset-lending management system is taken into account, it's not heavy, and users might not always lend and transfer an asset regularly and seek a quick user experience. Users are ready to wait for a longer transaction response from the blockchain when they lend assets. ZKP may thus have a significant performance impact on blockchain-based applications, although this impact could still be acceptable in the context of an asset lending management system. When considering whether to incorporate ZKP into other situations that call for high performance and huge scale, it is still crucial to take the trade-offs between security and performance into account. While ZKP may offer an extra layer of protection to conceal the user's identity, it may also slow down throughput and increase processing times for transactions.

In conclusion, the design of the system and the particular blockchain-based application will determine the influence of ZKP on performance. According to the results of the control experiments in Chapter 6.2, while ZKP may have little impact on Ethereum, it may have a significant impact on Hyperledger Fabric. When considering whether to include ZKP into a blockchain-based application, it's also important to take into account the system's workload and user expectations. To choose the best way to employ ZKP, security and performance trade-offs must be carefully considered.

7.2 Limitations and Future Works

Although the results of our experiments provide valuable insights, there are still limitations and areas for improvement. First, the test functions we used may not be sufficient to cover all use cases and scenarios that can happen in the asset-lending management system. Therefore, we need to set up more test cases that simulate real-world complex scenarios requiring the execution of multiple functions in a specific order or number of times to emulate the spur of traffic.

Another limitation is that we did not measure the off-chain Zokrates operations, such as computing ZKP proof and witness. These operations require significant time and computing resources (28), which could impact the user experience, especially when users are performing intensive workload computations on various terminal devices. In the future, we could leverage virtual machines to simulate terminal devices used by users and measure the time spent on off-chain calculations.

Moreover, the use of different benchmark tools, Caliper for testing Ethereum and JMeter for testing Hyperledger Fabric, may have introduced inconsistencies into the benchmark results, making it challenging to compare the performance of the two frameworks on the same baseline. Therefore, in future studies, we can use a new and improved Caliper that supports Idemix credentials and enables us to use the same group of test cases and settings to benchmark two different blockchain frameworks.

8

Conclusion

In this study, we conduct an empirical study to investigate the performance impact of ZKP on a blockchain-based asset management system, in order to determine the cost of anonymous transactions in the blockchain. To achieve this goal, we not only design and conduct a series of control experiments, but also implement benchmark workflows for two popular frameworks, Hyperledger Fabric and Ethereum. Finally, we conclude a general conditional result from the experiments that ZKP has a noticeable performance impact on Hyperledger Fabric than on Ethereum due to the different implementation of ZKP. We then draw reasonable conclusions about two proposed research questions:

- **RQ 1**: Does ZKP have significant impact on the performance of a blockchain-based application?

  - Based on the results of our benchmark experiments, we can confirm that ZKP has a performance impact on blockchain-based applications. However, the impact varies depending on the framework used and the type of ZKP cryptosuit employed, which can be categorized as either fully on-chain or partially on-chain. We observed that fully on-chain ZKP calculations had a more significant performance impact than partially on-chain ZKP calculations due to the heavy computational load involved in generating the witness and proof and uploading them to the blockchain for verification. Nonetheless, not all application scenarios require high-performance transactions. In the case of asset-lending management, which is a low-throughput system, the performance impact of ZKP is acceptable.

- **RQ 2**: Which blockchain framework has better performance with ZKP enabled?

  - Based on our experiments, we could not determine which blockchain framework has better performance with ZKP enabled. This is due to the different experiment setups and measurements, with Hyperledger Fabric using its native Idemix feature to realize ZKP on-chain, and Ethereum relying on the external toolbox Zokrates to generate and upload ZKP proof and verify contracts to the blockchain indirectly. Thereby, we could only deduce that Ethereum is less volatile and more stable than Hyperledger Fabric when it comes to ZKP on-chain performance. Also, it is worth noting that our experiments did not take off-chain ZKP operations into consideration for Ethereum, such as generating ZKP witness and calculating proofs, which could exert significant impact on the overall performance of Ethereum-based application.

References

[1] Ruhi Taş and Ömer Özgür Tanriöver. **Building A Decentralized Application on the Ethereum Blockchain**. In *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pages 1–4, 2019. 1

[2] Vitalik Buterin. **Ethereum: A next-generation smart contract and decentralized application platform**. 2014. 1

[3] Q.K. Nguyen. *'Blockchain—A financial technology for future sustainable development,'*. Develop. (GTSD. 1

[4] L. Cocco, A. Pinna, and M. Marchesi. **'Banking on blockchain: Costs savings thanks to the blockchain technology,'**. *Future Internet*, **9**(3):25,. 1

[5] M. Hölbl, M. Kompara, A. Kamišalić, and L.N. Zlatolas. **'A systematic review of the use of blockchain in healthcare,'**. *Symmetry*, **10**(10):470,. 1

[6] C.C. Agbo, Q.H. Mahmoud, and J.M. Eklund. **'Blockchain technology in healthcare: A systematic review,'**. *Healthcare*, **7**(2):56,. 1

[7] L.A. Linn and M.B. Koo. *'Blockchain for health data and its potential use in health it and health care related research,'*. ONC/NIST, Gaithersburg, MD, USA. 1

[8] M. Mettler. **'Blockchain technology in healthcare: The revolution starts here,'**. 1

[9] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang. **'Consortium blockchain for secure energy trading in industrial Internet of Things,'**. *IEEE Trans. Ind. Informat*, **14**(8):3690–3700,. 1

[10] E. Mengelkamp, B. Notheisen, C. Beer, D. Dauer, and C. Weinhardt. **'A blockchain-based smart grid: Towards sustainable local energy markets,'**. *Comput. Sci. Res. Develop*, **33**:1–2, 207–214,. 1

[11] N.Z. Aitzhan and D. Svetinovic. **'Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams,'**. *IEEE Trans. Dependable Secure Comput*, **15**(5):840–852,. 1

[12] G. Perboli, S. Musso, and M. Rosano. **'Blockchain in logistics and supply chain: A lean approach for designing real-world use cases,'**. *IEEE Access*, **6**:62018–62028,. 1

[13] E. Tijan, S. Aksentijević, K. Ivanić, and M. Jardas. **'Blockchain technology implementation in logistics,'**. *Sustainability*, **11**(4). 1

[14] Pinyaphat Tasatanattakool and Chian Techapanupreeda. **Blockchain: Challenges and applications**. In *2018 International Conference on Information Networking (ICOIN)*, pages 473–475, 2018. 1

[15] S Muralidhara and B A Usha. **Review of Blockchain Security and Privacy**. In *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, pages 526–533, 2021. 1

[16] Security Gateway System Team and Rüschlikon Switzerland. **Specification of the Identity Mixer Cryptographic Library Version 2 . 3 . 0 ***. 1

[17] Jacob Eberhardt and Stefan Tai. **ZoKrates - Scalable Privacy-Preserving Off-Chain Computations**. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1084–1091, 2018. 1

[18] Stuart Haber and W Scott Stornetta. *How to time-stamp a digital document*. Springer, 1991. 3

[19] Ethereum. **Proof-of-stake (PoS)**, 2021. 4

[20] Seungeun Kim, Joohyung Kim, and Dongsoo Kim. **Implementation of a blood cold chain system using blockchain technology**. *Applied Sciences*, **10**(9):3330, 2020. 8

[21] Uriel Fiege, Amos Fiat, and Adi Shamir. **Zero knowledge proofs of identity**. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 210–217, 1987. 8

[22] **MSP implementation with identity mixer¶**. 9

[23] Jens Groth. **On the size of pairing-based non-interactive arguments**. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016. 11

[24] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. **Bulletproofs: Short proofs for confidential transactions and more**. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018. 13

[25] Xiaohui Yang and Wenjie Li. **A zero-knowledge-proof-based digital identity management scheme in blockchain**. *Computers & Security*, **99**:102050, 2020. 15

[26] Shiwei Xu, Xiaowen Cai, Yizhi Zhao, Zhengwei Ren, Le Du, Qin Wang, and Jianying Zhou. **zkrpChain: Towards multi-party privacy-preserving data auditing for consortium blockchains based on zero-knowledge range proofs**. *Future Generation Computer Systems*, **128**:490–504, 2022. 15

[27] Tianyu Bai, Yangsheng Hu, Jianfeng He, Hongbo Fan, and Zhenzhou An. **Health-zkIDM: A Healthcare Identity System Based on Fabric Blockchain and Zero-Knowledge Proof**. *Sensors*, **22**(20):7716, 2022. 15

[28] David Gabay, Kemal Akkaya, and Mumin Cebe. **Privacy-preserving authentication scheme for connected electric vehicles using blockchain and zero knowledge proofs**. *IEEE Transactions on Vehicular Technology*, **69**(6):5760–5772, 2020. 15, 37

[29] Xiaohui Yang and Wenjie Li. **A zero-knowledge-proof-based digital identity management scheme in blockchain**. *Computers & Security*, **99**:102050, 2020. 16

[30] Péter Szilágyi. **EIP-225: Clique proof-of-authority consensus protocol**. Ethereum Improvement Proposals, March 2017. [Online serial]. 19, 23

[31] X Recommendation. **509: The Directory–Authentication Framework**. *ITU*, **8**:97, 1988. 23

[32] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. **Noninteractive zero-knowledge**. *SIAM Journal on Computing*, **20**(6):1084–1118, 1991. 43

Appendix

8.1 Explanation of Zokrates Program

Let's assume a scenario, prover P needs to show his social security number to verifier V who likes a company without revealing the real number. Then P can choose to encrypt the value y=$H(x)$ and send y to V and prove x with the knowledge of y. The hash calculation can be implemented using the Zokrates language, as shown in Listing 1, which is a SHA-256 calculation. Because Zokrates defines the size of a field to be only 128 bits, we need 4 fields totaling 512 bits to realise the SHA-256 calculation. The fields a, b, c, d in the method signature are four private fields that coincide with P's secret value. The method returns two hashed values. P only needs to compute a ZK proof of a, b, c, d.

```
import "hashes/sha256/512bitPacked" as sha256packed;

def main(private field a, private field b, private field c, private field d, field h0,
↪   field h1) -> field[2] {
    field[2] h = sha256packed([a, b, c, d]);
    asserts(h[0]==h0);
    asserts(h[1]==h1);
    return h;
}
```

Listing 1: ZoKrates Program Computing SHA-256-Hash

A mathematical demonstration that the calculation carried out by a program is accurate is known as a proof of computation. The proof is generated using a technique called ZK-SNARKs (zero-knowledge succinct non-interactive argument of knowledge). This technique allows the prover (the party who generated the proof) to prove that he has knowledge of a particular proposition without revealing any information about the proposition itself (32). In the case of Zokrates, the proposition being proved is that the computation performed in the program is correct.

The calculated piece of information also known as proof, which can be validated on-chain through invoking the verification function. A smart contract running on the blockchain is implemented with a such function to perform the verification process.

In contrast, the proof of computation is generated off-chain, meaning it is generated outside the Ethereum blockchain. This is done by running the Zokrates toolbox, which takes the program and inputs as input and generates the proof. The proof is then sent on-chain.

8.2 Configuration of 2-organisations Hyperledger Fabric Network with Idemix Feature Enabled

To enable ZKP, we first need to copy and store the 'IssuerPublicKey' and 'IssuerRevocationPublicKey' provided by the CA during startup.This is used by the Verifier (MSP) to verify the user's Idemix credential, as it is signed by the issuer's secret key. The configuration of the Verifier MSP is shown in Listing 2, the 'MSPDir' property specifies the location of 'IssuerPublicKey' and 'IssuerRevocationPublicKey', and the type of MSP is idemix, indicating that Org1Idemix is responsible for verifying the proof contained in the Idemix credential that signs the transaction.

```yaml
    - &Org1Idemix
        Name: idemixMSP1
        ID: idemixMSPID1
        msptype: idemix
        MSPDir: crypto-config/peerOrganizations/idemix-config
        Policies:
            Readers:
                Type: Signature
                Rule: "OR('idemixMSPID1.client')"
            Writers:
                Type: Signature
                Rule: "OR('idemixMSPID1.client')"
```

Listing 2: Idemix Verifier MSP Configuration in configtx.yaml

As shown in Listing 3. This method must take the user's 'x509Enrollment' object as input in order to hide all its sensitive attributes, and produces an 'idemixEnrollment' object that automatically provides ZKP procedures and guarantees that the user's privacy is anonymous and unlinkable, as only limited attributes can be revealed in 'idemixEnrollment‘, such as the user's role and organisational unit, which can't be traced back to the original transactor identity.

```
public HFClient getClient(AppUser appUser, boolean isIdemix) throws Exception {
    if (isIdemix) {
            AppUser newAppUser = new AppUser(
                    appUser.getName(),
                    appUser.getAffiliation(),
                    appUser.getMspId(),
                    caClient.idemixEnroll(appUser.getEnrollment(), mspIdemix)
            );
            return getClient(newAppUser);
    }
    return getClient(appUser);
}
```

Listing 3: Enroll Idemix Credential with `fabric-sdk-java`

As shown in the code in Listing 4.

```
def main(private Id) return (bool result)
    hash_prefix = getPrefix(Id)
    hash_postfix = getPostfix(Id)
    if
    hash_prefix == 1adbc6fcbb71cf6ce97c358aa21e41ab60ccbfd41d95759e941ff1ce9e6fec7e
    ↪  and
    hash_postfix == 5d5dce8b8b874329276826c50744fd4afd1aa5ecf9aaa1c5695d1685835a1b46
            return true
    else
            return false
    terminate
```

Listing 4: Secret Function with Zokrates

As shown in Listing 5, we import the verifier contract previously generated by Zokrates into the asset contract, as we want to verify the user identity via ZKP before querying the blockchain. The user must not only pass the asset id, but also provide proofs and inputs from *proof.json*.

Therefore, in order for the gPRC sampler to successfully produce a workload, we need to specify the directory of a .proto file that defines the service and message, as shown in the listing 6.

Chaincode verify Idemix attributes

```
pragma solidity ^0.8.0;
import "./verifier.sol";

contract asset {
    mapping(string => int) private assets;
    Verifier public ve = new Verifier();
    function queryAsset(string memory asset_id, Verifier.Proof memory proof, uint[2]
↪  memory input) public view returns (Asset asset) {
        ve.verifyTx(proof, input);
        asset = assets[asset_id];
    }
    // ...
}
```

Listing 5: Use verifyTx in Application Smart Contract

```
syntax = "proto3";

package mypackage;

service BlockchainService {
  rpc QueryBlockchain (BlockchainRequest) returns (BlockchainResponse) {}
}

message BlockchainRequest {
  AppUser appUser = 1;
  bool isIdemix = 2;
  string function = 3;
  repeated string args = 4;
}

message BlockchainResponse {
  string response = 1;
}

message AppUser {
  string name = 1;
  string mspid = 2;
  bytes cert = 3;
  bytes privateKey = 4;
}

```

Listing 6: The .proto File That Defines a gRPC Service With a Method to Query the Blockchain

```
    {
    "caliper": {
        "blockchain": "ethereum",
    },
    "ethereum": {
        "url": "ws://localhost:8546",
        "contractDeployerAddress": "0xc0A8e4D217eB85b812aeb1226fAb6F588943C2C2",
        "contractDeployerAddressPassword": "password",
        "fromAddress": "0xc0A8e4D217eB85b812aeb1226fAb6F588943C2C2",
        "fromAddressPassword": "password",
        "transactionConfirmationBlocks": 2,
        "contracts": {
            "simple": {
                "path": "./src/ethereum/simple/simple_zkp.json",
                "estimateGas": true,
                "gas": {
                    "queryAllAssets": 100000,
                    "createAsset": 70000
                }
            }
        }
    }
}


```

Listing 7: The Network Config Json That Defines the Ethereum Blockchain Configuration

```
    func (s *SmartContract) queryAssetsIdemix(APIstub shim.ChaincodeStubInterface)
↪   sc.Response {

        ou, found, err := cid.GetAttributeValue(APIstub, "ou")
        if err != nil {
                return shim.Error("Failed to get attribute 'ou'")
        }
        if !found {
                return shim.Error("attribute 'ou' not found")
        }
        logger.Infof("Organizational unit: '%s'", ou)

        // ... query operations
    }
```

Listing 8: Chaincode Gets Transactor's Attrubutes While Using Idemix Credential to Sign Transctions