Operating ZKPs on Blockchain

A Performance Analysis Based on Hyperledger Fabric

Pan, Rui; Shi, Zeshun; Belloum, Adam; Zhao, Zhiming

DOI
10.1109/DAPPS61106.2024.00018

Publication date
2024

Document Version
Final published version

Published in
Proceedings - 2024 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2024

Citation (APA)
Pan, R., Shi, Z., Belloum, A., & Zhao, Z. (2024). Operating ZKPs on Blockchain: A Performance Analysis Based on Hyperledger Fabric. In *Proceedings - 2024 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2024* (pp. 69-78). (Proceedings - 2024 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2024). IEEE. https://doi.org/10.1109/DAPPS61106.2024.00018

2024 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS) | 979-8-3503-6295-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/DAPPS61106.2024.00018

Operating ZKPs on Blockchain: A Performance Analysis Based on Hyperledger Fabric

Rui Pan*, Zeshun Shi†, Adam Belloum*, Zhiming Zhao*✉

*Multiscale Networked Systems (MNS) Group, University of Amsterdam, 1098 XH, the Netherlands
†Cyber Security Group, Delft University of Technology, 2628 XE Delft, the Netherlands
Email: panrui19980316@gmail.com, z.shi-2@tudelft.nl, { a.s.z.belloum,z.zhao}@uva.nl

***Abstract*—Preserving privacy in blockchain-based systems is crucial for ensuring anonymity and confidentiality during transactions. While cryptographic solutions can address on-chain privacy concerns, their implementation on blockchains may introduce performance overhead, which remains unclear to researchers and practitioners. This paper investigates the performance impact of integrating zero-knowledge proofs (ZKPs) into the widely adopted permissioned blockchain framework called Hyperledger Fabric. The study focuses on evaluating the scalability and bottleneck aspects of blockchain platforms incorporating ZKPs. Through comprehensive experimentation and analysis, the study reveals that the integration of ZKPs compromises performance in terms of transaction rates and latency, while effectively safeguarding users' personal information. Implementing on-chain ZKP feature would result in a performance loss of 30% to 87.5% in various experimental configurations in Hyperledger Fabric. The findings presented in this paper are informative for the design and implementation of blockchain-based systems with strict privacy requirements.**

***Index Terms*—Blockchain, Zero-Knowledge Proofs, Performance Analysis, Hyperledger Fabric.**

I. Introduction

In recent years, the decentralization of applications has witnessed a significant surge, establishing itself as a fundamental pillar in the Web 3.0 ecosystem. This phenomenon can be attributed to the maturation and advancement of blockchain technology, which has garnered recognition from both academia and industry [1], [2], [3], [4]. The potential of blockchain has been realized through its adoption in diverse domains, including digital currency, anonymous voting, and property management. It effectively addresses inherent drawbacks of centralized systems, such as privacy limitations, market opacity, and excessive fees [5].

Preserving transaction privacy on the blockchain assumes paramount importance given the fundamental contradiction between its public verifiability and the imperative for privacy protection. The transparency inherent in blockchain's transaction history exposes the potential risks of privacy infringements. Consequently, the adoption of suitable encryption algorithms and privacy-preserving techniques becomes imperative to ensure transaction confidentiality and anonymity. By ensuring secure operations within a blockchain framework, the introduction of such privacy protection mechanisms holds critical significance, facilitating widespread blockchain adoption and fostering its implementation in sensitive domains [6]. 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, such as Hyperledger Fabric's Idemix and Ethereum's Zokrates, the practical application and evaluation of these methods in real-world scenarios have been limited [7], [8]. 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.

In this context, this paper investigates the performance impact of the privacy-preserving technique ZKP on a commonly used permissioned blockchain framework under various experimental scenarios. The rest of the paper is structured as follows: Section II first introduces the basic blockchain background and the tested blockchain framework, then explains the principle, advantages and challenges of using ZKPs on blockchain. Section III 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. Section IV details the implementation details of the benchmark. Section V discusses the experiments, comparison results and evaluation. Section VI introduces the related work. Finally, Section VII presents the conclusion, discussion, and future work.

II. Background Knowledge

In this section, we will explain the background knowledge. The section is started with an introduction to the blockchain and Hyperledger Fabric. Next, zero-knowledge proofs and their applications in blockchain are described in detail.

A. Blockchain

Blockchain is a decentralized and immutable digital ledger that securely records and verifies transactions across multiple participants. In blockchain, 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 of the blockchain

979-8-3503-6295-4/24/$31.00 ©2024 IEEE
DOI 10.1109/DAPPS61106.2024.00018

network. Upon receiving the incoming block, other users 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 ledgers 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 authorize 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.

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

- **Permissionless Blockchain.** Also known as public blockchains, 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.** As opposed to a permissionless framework, known as a private blockchain, are restricted networks where access is controlled by 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. Some of the most successful examples of permissioned blockchains include Hyperledger Fabric and Sawtooth [9].

B. Hyperledger Fabric

Hyperledger Fabric[1], an open-source permissioned blockchain platform, has emerged as a prominent solution for enterprise use cases within the Hyperledger projects hosted by the Linux Foundation. Its modular architecture allows for enhanced flexibility and customization in developing decentralized applications (dApps) on the blockchain. Through its membership service provider (MSP) mechanism, Fabric operates as a permissioned blockchain framework, limiting access and transaction capabilities to authorized users or organizations. Supporting multiple programming languages, offering scalable consensus mechanisms, and boasting a robust governance framework, Fabric has garnered significant popularity as a preferred platform for constructing blockchain-based systems in the enterprise realm [10]. Comprising six key components, Fabric provides a reliable foundation for building secure and efficient blockchain networks.

- **Channels**: Hyperledger Fabric channels protect transactions. Channels enable a set of people to trade without other network members seeing the specifics unless authorized.
- **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 authorize 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 with 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.

C. Zero-knowledge Proofs

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 (ZKP) that is crucial in addressing these issues. ZKP is a method of proving the possession of some information, such as a private key, without revealing the actual information [11]. 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.

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

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 the cryptography toolkits designed for blockchain frameworks:

- ***Idemix:*** Identity Mixer is an anonymous certificate solution proposed by IBM in 2009 that implements the underlying ZKP. The motivation behind this is that the traditional X.509 certificate adopted by Hyperledger is prone to the issue of over-exposure of all attributes, which can lead to information leakage because users have to present all attributes on the certificate during authentication. Therefore, anonymous authentication techniques are needed to minimize 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 [12].
- ***How It Works:*** As shown in Figure 1, the Idemix process requires three participants, namely the issuer, the user and the verifier. In the beginning, the user or peer will generate a request for a certificate and send it to the issuer. The issuer then returns a certificate in the form of an Idemix credential containing the user's attributes to a 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 by leveraging the CA's public key to check token validity.
- ***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 [12].

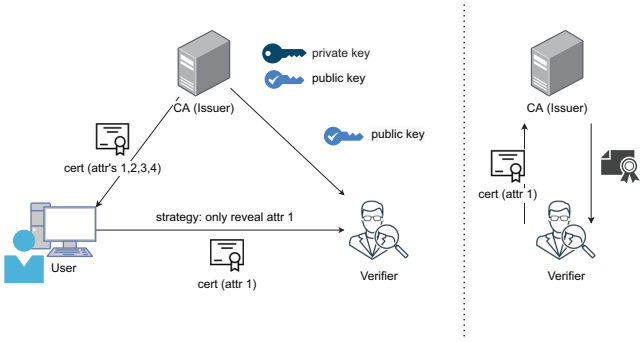


Fig. 1. Idemix System Overview

III. Model Design & Implementation

This section describes the model design and implementation choices of our performance study. Specifically, we first describe the configuration of the blockchain network, followed by the benchmark workflow for Hyperledger Fabric. Finally, we present the details of operating zero-knowledge proofs on Hyperledger Fabric using Idemix.

A. Blockchain Network

We build a Hyperledger Fabric network consisting of two separate organizations for testing purposes (as shown in Figure 2). In this case, an anonymous transaction is sent from one organization to another to test ZKPs. Each organization has peer nodes, Certificate Authority (CA) nodes, and 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) 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.

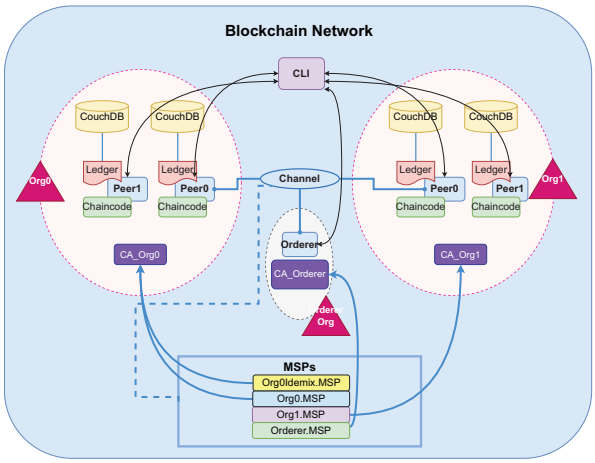


Fig. 2. The Topology of a Two-Organizations Hyperledger Fabric Network

By having a network with two organizations, we have simulated a scenario where only users with identities from the owning organization 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 the order of transactions, and the CLI provides an interface for users to interact with the network and deploy the chaincode.

B. Benchmark Workflow

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

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

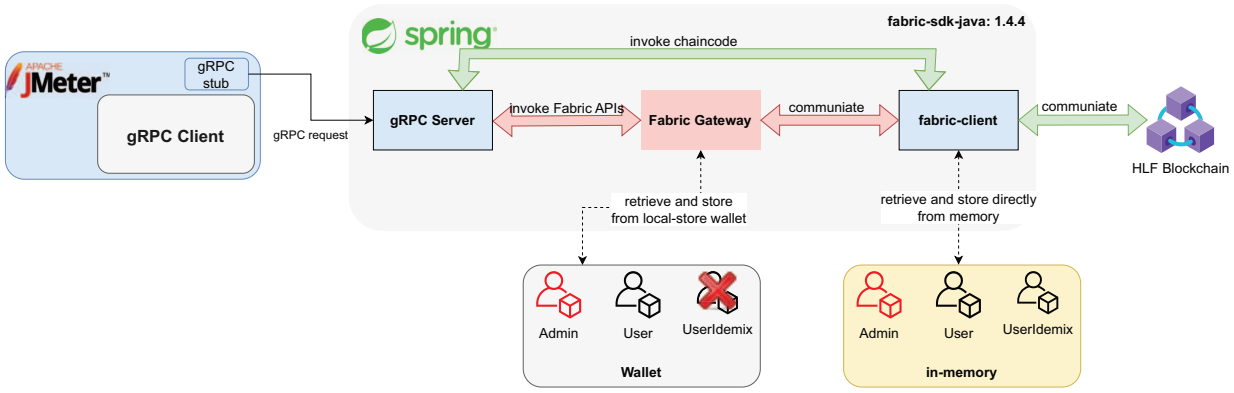


Fig. 3. Hyperledger Fabric Benchmark Workflow

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 transaction throughput. In addition, JMeter supports third-party plug-ins[3] 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.

As shown in Figure 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, where we opted for fabric-sdk-java to interact with the blockchain and manage user identities in memory after encountering compatibility issues with fabric-gateway, which prevented us from storing the Idemix identity in the wallet. And our backend is a two organizations Fabric network.

To allow gRPC requests in JMeter, we installed a third-party plugin named jmeter-grpc-request[4]. 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 *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, 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.

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.

C. Idemix in Hyperledger Fabric

Because Hyperledger Fabric natively supports ZKP during authentication to prevent transactors from overly exposing their personal attributes when signing transactions with traditional X.509 [13] 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 Section II-C.

The interaction among three components - CA, SDK, and Idemix-MSP - involving the Idemix process is illustrated in Figure 4. Here, CA refers to the authority that issues ZKP credentials to SDK (the user), who can then use the credential for making transactions based on ZKP. The Idemix-MSP acts as a verifier and intercepts new transactions to verify the validity of the credential. To enable ZKP, we need to leverage CA to add an Idemix MSP configuration of an organization 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 anonymous

[3]https://github.com/undera/perfmon-agent
[4]https://github.com/zalopay-oss/jmeter-grpc-request

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.

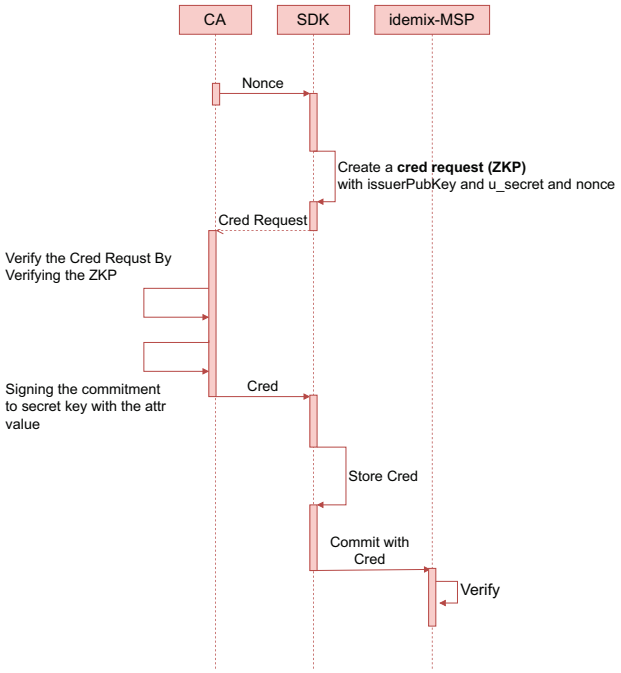


Fig. 4. The Sequence Diagram of Interaction Between Components in Idemix

IV. Experiment Setup

This section presents the experiment parameters and benchmark metrics for our experiments.

A. Experiment Parameters

The environment for the experiment is set up by installing and configuring a 2-organization Hyperledger Fabric network with and without *Idemix* and JMeter on the same machine, but hosted in different Docker containers to ensure resource isolation (as shown in Figure 5). The configuration is shown in Table I, 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. Specially, four different operations were tested: query without Idemix credential, query with Idemix credential, add without Idemix credential, and add with Idemix credential.

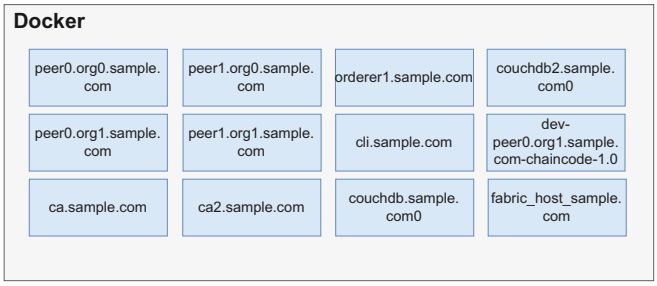


Fig. 5. Two Organization Hyperledger Fabric Network Container Overview

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)[5] 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 II, 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 Section II-C to observe the performance effect of ZKP. Idemix automatically hides the identity of the actor, and verifies the required attributes at the Idemix MSP, all in the background as long as the Idemix credential is used. We thus can control the variable easily.

TABLE I
JMeter Configuration

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, PerfMon Metrics Collector	Measure needed metrics: throughout, avg latency, cpu and memory

TABLE II
Hyperledger Fabric Control Experiments Setup

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

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

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

TABLE III
EVALUATION METRICS AND DESCRIPTIVE STATISTICS

Metric	Unit	Descriptive Statistics
Latency	millisecond	mean, standard deviation
Throughput	transactions per second	mean, standard deviation
Error rate	percentage	mean
CPU use	percentage	average, maximum
Memory	megabyte	average, maximum

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 (TPS), 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 Fabric.

C. Use Cases

We have selected an asset lending management system as our designated test case due to the widespread necessity for businesses to effectively track and manage equipment lent to employees, including but not limited to laptops, tablets, smartphones, VR headsets, and bicycles. Within this system, each lending transaction represents a comprehensive record encompassing essential asset details, owner information, transfer history, and any reported damages. These transaction records are maintained in an immutable ledger, ensuring their integrity and preventing unauthorized modifications. To ensure the accuracy of subsequent transactions, each new entry is based on the latest recorded transaction. By adopting this blockchain-based approach, businesses can streamline their asset management processes, facilitating efficient monitoring and enabling the identification of responsible parties in cases of asset damage or loss. The high-level view of the system architecture is illustrated in Figure 6; the back-end of the system can be powered by 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.

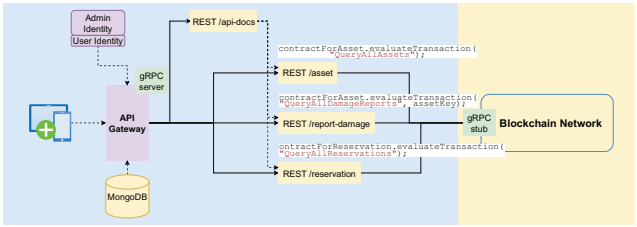


Fig. 6. The Asset Lending Management System Mainly consists of Two Parts, Back-End and Front-End

V. EXPERIMENT RESULT

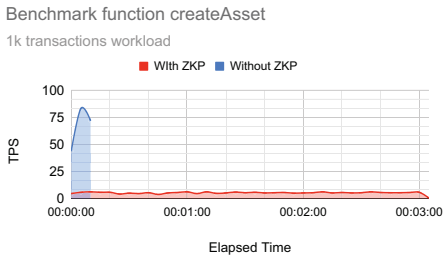
In this section, we present our experimental result. The experiments were conducted to evaluate the impact of ZKP on the performance of the Hyperledger Fabric blockchain framework.

A. Performance Bottleneck

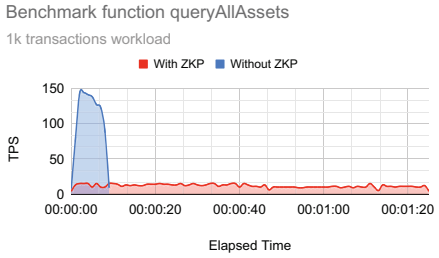
Four different operations were tested: query without Idemix credential, query with Idemix credential, add without Idemix credential, and add with Idemix credential. The results showed in Table IV, where two different scenarios: with and without ZKP for two functions: *queryAllAssets* and *addAsset* are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

It can be seen that operations with ZKP had significantly higher latencies and lower throughput compared to 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.

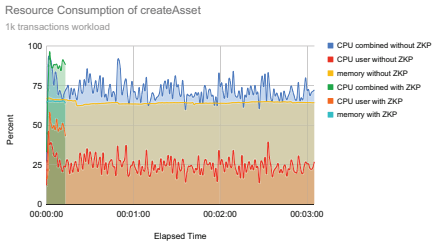
For the throughput of the query operation with ZKP, it was much lower at 8 transactions per second compared to 55.87 transactions per second for the one without ZKP. Meanwhile, the add operation with ZKP had a lower throughput of 9.9 transactions per second compared to 158.28 transactions per second for the add operation without ZKP, hence we notice that the average difference is 7 to 15 times. Finally, the



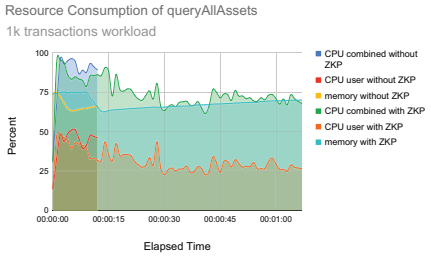
(a) Results of createAsset elapsed time versus transaction per second ZKP performance comparison results



(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

Fig. 7. Results of Control Experiments with Hyperledger Fabric and ZKP: Part 1

error rate of all operations was low, with the query operation with ZKP having the highest error rate of 0.04% which is negligible. Therefore, these results indicate that the operations without ZKP outperformed the operations with ZKP.

We can also see Figure 7 (a) and (b), which are two scatter plots illustrating the comparison of TPS over elapsed time

TABLE IV
THE RESULTS OF A CONTROL EXPERIMENT FOR HYPERLEDGER FABRIC BENCHMARK WITH DESCRIPTIVE STATISTICS

Label	queryAllAssets (no zkp)	queryAllAssets (with zkp)	add (no zkp)	add (with zkp)
#Sample	1000	1000	1000	1000
Avg	82	**591**	23	**341**
Min	37	0	11	0
Max	168	**1003**	80	**1005**
Std. Dev	20.43	**153.68**	8.91	**332.97**
Err %	0.00	**0.04**	0.00	**0.45**
Throughput	55.87	**8.00**	158.28	**9.90**
Recvd KB/sec	127.73	**17.52**	9.12	**0.68**
Avg Bytes	2341.00	2241.83	59.00	70.81

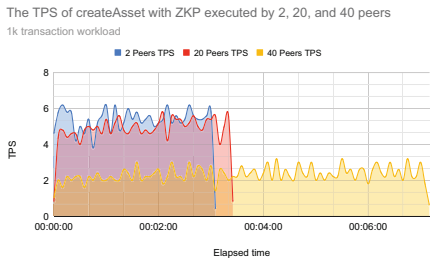
between two series: red with ZKP and blue without, with 1000 total transactions in the control experiment. With the ZKP function enabled, it is evident that both scenarios take longer to send the same number of transactions. Additionally, we can observe that *createAsset* takes around 3 minutes compared 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 7 (c) and (d) are two area charts, which show the comparison of resource consumption over time of the same functions with and without ZKP, with different colours representing different metrics. We can see a similar trend that the group without ZKP saves more resource consumption by finishing earlier.

B. Performance Scalability

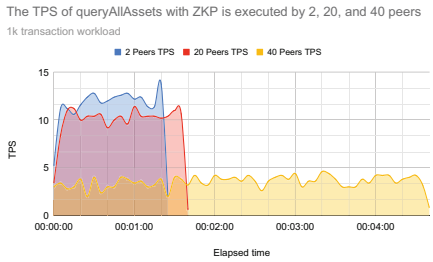
The results depicted in Figure 8 highlight the varying impact of ZKP performance on scalability, revealing a discernible pattern wherein TPS decreases as the network is scaled by extending the number of peers. To explore this trend, we designed three levels of network size with 2, 20 and 40 peers. Unlike the experiment settings in Figure 7, where the independent variable was enabling ZKP or not, here it is the number of peers with enabled ZKP by default. Upon observing Figures 8 (a) and (b), it becomes apparent that TPS declines as more peers are added to the network, where the differences between 2 and 20 peers networks when benchmarking both query and create operations is negligible and 20 peers case has relatively lower TPS and takes extra 10 seconds to finish; this effect is particularly pronounced in benchmarking with 40 peers which take twice as long to complete an equivalent workload at its lowest TPS. However, Figures 8 (c) and (d) demonstrate that all experimental cases consume nearly identical amounts of CPU and memory resources.

VI. RELATED WORK

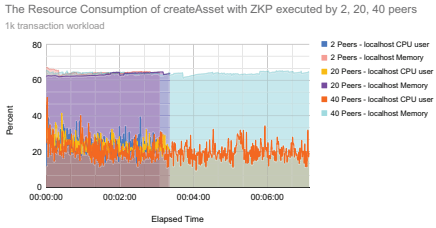
Several studies have proposed to use ZKPs to protect the privacy of applications on the Hyperledger Fabric blockchain. Li et al. [14] introduce a blockchain-based ZKP technique-based identity verification system for ride-sharing platforms.



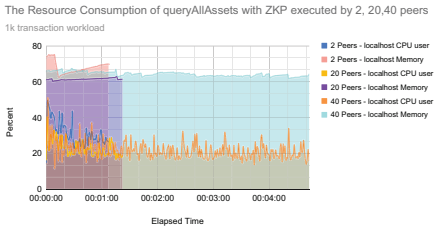
(a) Results of createAsset elapsed time versus transaction per second ZKP performance with varying numbers of peers



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



(c) Results of createAsset elapsed time versus resource consumption ZKP performance comparison results with varying numbers of peers



(d) Results of queryAllAssets elapsed time versus resource consumption ZKP performance comparison results with varying numbers of peers

Fig. 8. Results of Control Experiments with Hyperledger Fabric and ZKP: Part 2

The platform is built by the authors using Hyperledger Fabric, while the ZKP module is implemented using Hyperledger Ursa, 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 [14], [15].

Similarly, Bai et al. [16] 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.

Our study introduces a significant innovation by focusing on the on-chain performance testing of Zero-Knowledge Proofs (ZKPs), unlike other approaches that predominantly use an off-chain computation and on-chain verification paradigm. This shift to executing all ZKP operations on-chain, as demonstrated in our Hyperledger solution, underscores the necessity and advantages of on-chain ZKPs. A comparison of our research with existing ZKP-based systems is shown in Table V. By implementing and evaluating ZKPs entirely on-chain, we highlight the potential for enhanced security and efficiency in blockchain networks. This methodology not only sets our work apart from existing studies that utilize mixed off-chain and on-chain operations but also emphasizes the critical role of on-chain ZKPs in advancing blockchain technology. Through measuring the computation times of different ZKP phases and the average throughput across various transaction rates on-chain, our study provides vital insights into the performance and scalability of blockchain systems equipped with ZKP capabilities.

VII. Conclusion

In this paper, 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 one popular framework called Hyperledger Fabric. Finally, we conclude a general conditional result, namely that implementing on-chain ZKPs would result in a performance loss of 30% to 87.5% in various experimental configurations in Hyperledger Fabric.

For further research, although the results of our experiments provide valuable insights, there are still limitations and areas for improvement. For example, the test functions we used

TABLE V
COMPARISON OF ZKP-BASED SYSTEMS ON HYPERLEDGER FABRIC

Aspect	Li et al.	Bai et al.	Our Work
Blockchain Platform	Hyperledger Fabric	Hyperledger Fabric	Hyperledger Fabric
ZKP Library	Hyperledger Ursa	Go-snark	*Idemix*
Application Field	Ride-sharing	Healthcare	*Asset lending management*
Proof Generation Location	Off-chain	Off-chain	On-chain
Proof Verification Location	On-chain	On-chain	On-chain
ZKP Operation Mode	Client-side generation, Blockchain verification	Off-chain computation, On-chain verification	All operations on-chain
Innovation	Fast off-chain proof generation	Efficient identity verification	On-chain ZKP operations, Enhanced on-chain performance testing

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. In addition, other encryption algorithms, such as secure multi-party computation (MPC), will also be considered for testing and comparison.

ACKNOWLEDGMENT

This research was made possible through partial funding from several European Union projects: ENVRI-Hub Next (101131141), EVERSE (101129744), BlueCloud-2026 (101094227), OSCARS (101129751), LifeWatch ERIC, BioDT (101057437, through LifeWatch ERIC), and Dutch NWO LTER-LIFE project.

REFERENCES

[1] R. Taş and O. O. Tanrıover, "Building a decentralized application on the ethereum blockchain," in *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2019, pp. 1–4.

[2] Q. Nguyen, *'Blockchain—A financial technology for future sustainable development,'*. Develop. (GTSD, 2016.

[3] L. Cocco, A. Pinna, and M. Marchesi, "'banking on blockchain: Costs savings thanks to the blockchain technology,'," *Future Internet*, vol. 9, no. 3, pp. 25,, 2017.

[4] Z. Shi, S. Farshidi, H. Zhou, and Z. Zhao, "An auction and witness enhanced trustworthy sla model for decentralized cloud marketplaces," in *Proceedings of the Conference on Information Technology for Social Good*, 2021, pp. 109–114.

[5] P. Tasatanattakool and C. Techapanupreeda, "Blockchain: Challenges and applications," in *2018 International Conference on Information Networking (ICOIN)*, 2018, pp. 473–475.

[6] S. Muralidhara and B. A. Usha, "Review of blockchain security and privacy," in *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, 2021, pp. 526–533.

[7] S. G. S. Team and R. Switzerland, "Specification of the identity mixer cryptographic library," 2010. [Online]. Available: https://dominoweb.draco.res.ibm.com/reports/rz3730_revised.pdf

[8] J. Eberhardt and S. 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)*, 2018, pp. 1084–1091.

[9] Z. Shi, H. Zhou, Y. Hu, S. Jayachander, C. de Laat, and Z. Zhao, "Operating permissioned blockchain in clouds: A performance study of hyperledger sawtooth," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2019, pp. 50–57.

[10] Z. Shi, C. de Laat, P. Grosso, and Z. Zhao, "Integration of blockchain and auction models: A survey, some applications, and challenges," *IEEE Communications Surveys & Tutorials*, 2022.

[11] U. Fiege, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1987, pp. 210–217.

[12] "Msp implementation with identity mixer¶." [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/idemix.html#how-to-use-idemix

[13] X. Recommendation, "509: The directory–authentication framework," *ITU*, vol. 8, p. 97, 1988.

[14] X. Yang and W. Li, "A zero-knowledge-proof-based digital identity management scheme in blockchain," *Computers & Security*, vol. 99, p. 102050, 2020.

[15] S. Xu, X. Cai, Y. Zhao, Z. Ren, L. Du, Q. Wang, and J. Zhou, "zkrpchain: Towards multi-party privacy-preserving data auditing for consortium blockchains based on zero-knowledge range proofs," *Future Generation Computer Systems*, vol. 128, pp. 490–504, 2022.

[16] T. Bai, Y. Hu, J. He, H. Fan, and Z. An, "Health-zkidm: A healthcare identity system based on fabric blockchain and zero-knowledge proof," *Sensors*, vol. 22, no. 20, p. 7716, 2022.

APPENDIX A
CONFIGURATION OF HYPERLEDGER FABRIC NETWORK WITH IDEMIX FEATURE ENABLED

This appendix section dedicated to the configuration of a Hyperledger Fabric network with the Idemix feature enabled offers a comprehensive exploration of the mechanisms and protocols necessary for integrating Zero-Knowledge Proofs (ZKPs) to enhance privacy and security. This integration facilitates the verification of transactions in a manner that ensures the anonymity and unlinkability of the user's identity, thereby fostering a more secure and private blockchain environment.

1) Verifier MSP Configuration for Idemix: The configuration of the Membership Service Provider (MSP) for a verifier within the network is a critical step in leveraging the Idemix feature. Listing 1 delineates the specific settings required, including the 'MSPDir' property which directs to the storage location of the 'IssuerPublicKey' and 'IssuerRevocationPublicKey'. These keys are paramount for the MSP's capability to authenticate Idemix credentials, ascertaining that they have been appropriately signed with the issuer's secret key. By designating the MSP type as 'idemix', this configuration underscores the verifier's role in scrutinizing the proofs embedded within Idemix credentials, which are pivotal for the validation of transaction signatures.

2) Enrolling Idemix Credential with Fabric-SDK-Java: The process of enrolling Idemix credentials, as illustrated in Listing 2, is an essential procedure that utilizes a Java method to process a user's 'x509Enrollment' object. This method is ingeniously designed to mask sensitive attributes,

Listing 1 Idemix Verifier MSP Configuration in configtx.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')"
```

subsequently producing an 'idemixEnrollment' object that inherently supports ZKP procedures. This approach is instrumental in preserving user privacy by limiting the visibility of attributes to only those necessary, such as the user's role and organizational unit. Such limitations ensure that the user's identity remains both anonymous and unlinkable, highlighting the method's significance in bolstering privacy within the blockchain network.

Listing 2 Enroll Idemix Credential with `fabric-sdk-java`

```
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);
}
```

3) gRPC Service Definition for Blockchain Queries: The introduction of a '.proto' file for a gRPC service, detailed in Listing 3, marks a pivotal advancement in querying the blockchain. This file outlines the essential structure for a service and messages, enabling effective interaction with the blockchain. The defined service, 'BlockchainService', incorporates a method 'QueryBlockchain' that facilitates the execution of blockchain queries through specified user attributes and function arguments. This configuration is crucial for the gRPC sampler's ability to generate workloads efficiently, showcasing the practical application of gRPC services in blockchain operations.

4) Chaincode for Verifying Idemix Attributes: Listing 4 presents a GoLang implementation of chaincode, specifically crafted to authenticate Idemix attributes during the transaction

Listing 3 The .proto File That Defines a gRPC Service With a Method to Query the Blockchain

```
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;
}

```

process. This implementation demonstrates the retrieval and verification of an organizational unit's attribute ('ou'), a fundamental aspect of transaction privacy and security. The practical application of ZKP within the chaincode is emphasized, highlighting the enhancement of privacy-preserving capabilities within blockchain transactions. This chaincode exemplifies the integration of advanced cryptographic techniques, such as ZKP, to ensure transactional integrity and privacy.

Listing 4 Chaincode Gets Transactor's Attrubutes While Using Idemix Credential to Sign Transctions

```
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
}
```