Post-Quantum User Authentication and Key Exchange Based on Consortium Blockchain

Shiwei Xu†
College of Informatics
Huazhong Agricultural University
Wuhan, P. R. China
xushiwei@mail.hzau.edu.cn

Ao Sun
College of Informatics
Huazhong Agricultural University
Wuhan, P. R. China

Xiaowen Cai
College of Informatics
Huazhong Agricultural University
Wuhan, P. R. China

Zhengwei Ren
School of Computer Science and Technology
Wuhan University of Science and Technology
Wuhan, P. R. China

Yizhi Zhao
College of Informatics
Huazhong Agricultural University
Wuhan, P. R. China

Jianying Zhou†
Singapore University of Technology and Design
Singapore
jianying_zhou@sutd.edu.sg

ABSTRACT

Consortium blockchain has been widely used in many application scenarios, where there is the demand for a universal user authentication and key exchange mechanism for all the application users in the system like Know Your Customer. Since current solution heavily rely on traditional public-key cryptosystems that are vulnerable to attacks from quantum computers, we design and implement the first post-quantum (PQ) user authentication and key exchange system for consortium blockchain, which is integrated with all the PQ public-key (i.e., signature and encryption/KEM) algorithms in the current round of NIST call for national standard. Furthermore, we also provide chaincodes, related APIs together with client codes for further development. Last but not least, we perform a systematic evaluation on the performance of the system including the consumed time of chaincodes execution and the needed on-chain storage space. Based on the experiment results, we discuss the implications of our findings, which are helpful for the PQ blockchain-based application developers, the undergoing NIST call and the developers of the PQ algorithms.

CCS CONCEPTS

• **Security and privacy → Systems security → Distributed systems security**; • Security and privacy → Cryptography → Public key (asymmetric) techniques.

KEYWORDS

Post-quantum cryptography, Consortium blockchain, User authentication, Key exchange, Performance evaluation

†Corresponding author

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

1 INTRODUCTION

As a distributed system, blockchain has gained much attention from researchers and industries in recent years due to enhanced data integrity, better transparency and greater traceability. Unlike public blockchain that are publicly accessible, consortium blockchain is managed by several organizations and only the organization members in the system are allowed to access the blockchain. Benefited from this characteristics, consortium blockchain has been widely used as a backbone for cross-company and cross-organization scenarios like medical information sharing [1], energy trading [2], e-health [3], smart home [4], etc.

To provide enhanced data integrity and traceability, the underlying consortium blockchain system heavily uses the public-key/asymmetric cryptography (e.g., RSA, ECDSA) and hash functions (e.g., SHA-256) to separately authenticate consortium members and hash/record the transactions between the members. While at the application layer, the consortium blockchain users also interact securely with each other by leveraging public-key/asymmetric cryptography (e.g., RSA, ECDSA, ECDH), which is essential for authenticating the users and exchanging keys that are used to further encrypt/decrypt private data stored on-chain.

However, since large-scale quantum computer is coming closer [5], the related Shor's algorithm [6] is posing a serious threat to the traditional public-key/asymmetric cryptography (e.g., RSA, ECDSA, ECDH) that is based on mathematics problems of integer factorization (IF), discrete logarithm (DL) or elliptic curve discrete logarithm (ECDL). If a practical quantum computer is available, then the Shor's algorithm would solve the IF, DL and ECDL problems in polynomial time. In that case, the security footstone of consortium blockchain would be badly broken.

To address the issue, PQ cryptography has been actively developed by research projects (e.g., PQCrypto [7], SAFEcrypto

[8], CryptoMathCREST [9], PROMETHEUS [10]) and some standardization initiatives [11-14] have also been launched. Among plenty of standardization initiatives, it is worth noting the National Institute of Standards and Technology (NIST) [15] call for proposals of PQ public-key cryptosystems including digital signature algorithms and Key Encapsulation Mechanism (KEM) algorithms, which is currently in its third round [16] and expected to deliver the first national standard drafts between 2021 and 2022. For future consideration, it is meaningful to fully evaluate the security and performance of these candidate algorithms in the consortium blockchain.

Many studies [17-21] have been done to develop PQ signature algorithms that are claimed to be suitable for blockchain systems and then build blockchain systems based on their PQ signature algorithms. While other works [22-23] apply the PQ candidate signature algorithms involved in the current round of NIST call to the existing consortium blockchain systems. These efforts mostly try to integrate PQ signature algorithms with the underlying consortium blockchain systems to provide PQ membership authentication and data traceability, which are considered more urgent as the security footstone of blockchains. However, to the best of our knowledge, there is no work focusing on exploring application-level PQ security mechanisms based on both digital signature and KEM algorithms in the NIST call in order to perform user authentication and preserve user data secrecy, which are also crucial when blockchain-based applications are developed, not even to mention a full evaluation on the security and performance of each candidate algorithm at the application layer of consortium blockchain.

1.1 Our Contributions

Therefore, we propose a chaincode-based PQ user authentication and key exchange system on top of consortium blockchain, which provides fundamental PQ security mechanisms that most blockchain-based applications need. Furthermore, based on our system, we also conduct a systematic evaluation on the performance of each PQ signature and KEM algorithms, which is helpful for the PQ blockchain-based application developers, the undergoing NIST call and the developers of the PQ algorithms.

- We design a chaincode-based CA providing post-quantum 3-layer (i.e., CA, user ID and user KEM) certificate services for blockchain-based application users to perform user authentication, based on which the users could further exchange keys by using the KEM certificate provided by the CA. It should be mentioned that the certificate services are cross-channel and could be provided to all the application users.
- We implement our system on top of Hyperledger Fabric and integrate the system with all the PQ public-key algorithms in the current (final) round of NIST call. Furthermore, we also provide chaincodes, related APIs together client codes, which allow developers to create their PQ applications based on our system services.
- We perform a systematic evaluation on the performance of our system including the consumed time of chaincodes execution and the needed on-chain storage space. Based on the experiment results, we discuss the implications of our findings, which are helpful for the PQ blockchain-based application developers, the undergoing NIST call and the developers of the PQ algorithms.

1.2 Paper Outline

The rest of the paper is structured as follows. In Section 2, we give an overview of the consortium blockchains and the PQ algorithms in current-round of NIST call. And then we present the design and implementation of our system based on consortium blockchain in Section 3. After that, in Section 4 we comprehensively evaluate the performance of the system based on each PQ candidate signature and KEM algorithm and discuss the implications of our findings. In Section 5, we introduce the related works. Finally, we draw a conclusion in Section 6.

2 BACKGROUND

In this section, we take an overview of consortium blockchains and Hyperledger Fabric. And then, we review all the candidate signature and KEM algorithms in the current round of NIST call and introduce their classifications based on the quantum-resistant problems.

2.1 Consortium Blockchain and Hyperledger Fabric

Consortium blockchain operates under a set of organizations, and provides a way to secure the interactions among the organizations that have a common goal but do not fully trust each other. In the organizations, some users want to keep their data public, while other users may only intend to share their data with specific users, for which user authentication and key exchange are quite essential.

Among current existing consortium blockchains, Hyperledger Fabric is one of the most successful projects, and it supports executing distributed applications (i.e., smart contracts, aka chaincodes) on a consortium of *organizations* and *peers* in one *channel*, whose business logic is programmed as *chaincodes* written in standard, general-purpose programming languages (e.g., Go, Node.js, Java).

Hyperledger Fabric introduces an *execute-order-validate* architecture, where distributed system nodes such as endorsement peer, commitment peer, orderer and client are deployed, and the peers are grouped by the organizations they belong to. All the chaincodes should be installed on specific peers before the execution of Hyperledger Fabric. The work flow of Hyperledger Fabric is introduced as follows. Firstly, the client invokes the execution of chaincode by submitting a transaction proposal to the endorsement peers. Secondly, after executing chaincode and generating signature for the execution result, the endorsement peers send the endorsed proposals back to the client. Thirdly, the client then broadcasts the endorsed proposals to the orderers. Fourthly, the orderers establish a total order on all submitted transactions in one channel, batch these transactions into a new block, and distribute both to all the commitment peers in the

channel. Finally, the commitment peers validate each transaction in the new block, and append the block to the distributed ledger.

During the execution of Hyperledger Fabric, peers can access on-chain data by *keys*, which are like member variables bound to the installed chaincodes, and a *key* can be a single datum or a tuple of data. All modifications and updates to the data and installed chaincodes are recorded by Hyperledger Fabric and therefore modifications can be easily detected.

Based on the architecture and work flow, Hyperledger Fabric provides certificate-based account for the users to login the system via client (nodes) and submit transaction proposals to peers. Just like peers, the user accounts are also grouped by the organizations and one user account is bound to only one organization. In one organization, there are two types of user accounts namely *ADMIN* and *USER*, the former of which acts like the organization administrator that can perform privileged operations such as adding new peers to the organizations, while the latter type is just normal user account, which is responsible for invoking the chaincodes. In essence, the certificate-based user accounts provided by Hyperledger Fabric are like Linux user accounts, which are used to login and operate the system rather than provide user personal information (e.g., name and locality) for the applications (e.g., Know-Your-Customer financial application) running in the system. Additionally, since user account and certificate are bound to only one organization, users in different organizations and channels cannot authenticate each other. Therefore, this drives the development of application-level user authentication and key exchange system for the smart contracts running on Hyperledger Fabric.

2.2 PQ Algorithms in the NIST Call

In the third (current) round of NIST call for PQ public-key algorithms, there are three digital signature (i.e., CRYSTALS-DILITHIUM [24], FALCON [25], Rainbow [26]) and four KEM (i.e., Classic McEliece [27], CRYSTALS-KYBER [28], NTRU [29], SABER [30]) finalist algorithms, which will continue to be reviewed for consideration for standardization at the conclusion of the third round. While at the same time, there are also three digital signature (i.e., GeMSS [31], Picnic [32], SPHINCS+ [33]) and five KEM (i.e., BIKE [34], NTRU Prime [35], FrodoKEM [36], SIKE [37], HQC [38]) alternate candidate algorithms, which are potentially standardized but most likely will not occur at the end of the third round. These alternate candidate algorithms may be reconsidered for various reasons (e.g., better performance, higher security level, broader range of hardness assumptions) in a fourth round of evaluation held by NIST.

It is worth mentioning that in order to eliminate the complexity of the padding scheme and the proofs needed to show the padding is secure, NIST currently only announces the KEM algorithms rather than the public-key encryption/decryption algorithms. Unlike one public-key encryption/decryption algorithm, one KEM algorithm can only be used for key exchange between network peers and users. In brief, in a KEM-based protocol, Alice generates and encapsulates a shared secret SS to get a ciphertext CT by using the public key of Bob, and Bob decapsulates SS from CT by using his private key. The shared secret SS is further used as a symmetric key to encrypt/decrypt the exchanged secret data between Alice and Bob.

2.3 PQ Algorithms Classification Based on the Quantum-Resistant Problems

To avoid the Shor's attack, researchers have developed PQ public-key algorithms relying on different mathematics problems other than the IF, DL or ECDL problems. Based on the mathematics problems, the PQ algorithms can be classified into different families introduced as follows.

2.3.1 PQ Digital Signature Algorithms. In the current round of NIST call, there are six PQ digital signature algorithms, which can be divided into three following groups.

- **Lattice-based cryptosystems.** This kind of cryptographic schemes are based on lattices, which are sets of points in n-dimensional spaces with a periodic structure. Lattice-based security schemes rely on presumed hardness of basic lattice problems like the Shortest Vector Problem (SVP) (i.e., find the shortest non-zero vector within a lattice), the Closest Vector Problem (CVP) (i.e., find a lattice vector that minimizes the distance from another target lattice) and the Shortest Independent Vectors Problem (SIVP) derived from SVP and CVP. In the current round, FALCON and CRYSTALS-DILITHIUM are lattice-based candidate algorithms and separately based on the Short Integer Solution problem (SIS) [39] over NTRU lattices and "Fiat-Shamir with Aborts" technique [40]. Lattice-based algorithms are promising quantum-resistant solutions with relatively efficient implementations, balanced key/signature sizes and strong security properties.
- **Hash-based cryptosystems.** The security of these schemes depends on the security of underlying hash function instead of on the hardness of a mathematical problem. Currently, Picnic is a hash-based algorithm and makes use of hash-based zero-knowledge proof technique [41], while SPHINCS+ is a stateless hash-based signature algorithm built on Merkle tree. Normally, hash-based schemes have very small key sizes but suffer from large signature sizes.
- **Multivariate-based cryptosystems.** Another family of problems that are used by some NIST PQ signature algorithms is related to solving multivariate quadratic equations over finite fields which is an NP-hard problem. Multivariate PQ schemes often lead to excessive key sizes, and the NIST multivariate-based signature algorithms are Rainbow and GeMSS.

As shown in Table 1, we summarize the details of all the six PQ digital signature algorithms in the current round of NIST call. By using different key sizes and parameters, all the algorithms can achieve different NIST security levels (i.e., bits-of-security level), which is defined as the effort required by a classical computer to perform a brute-force attack on a given-length cryptographic key. Normally NIST security levels 1~5 approximately imply 128/160/192/224/256-bits-of security levels.

Table 1: Details of PQ digital signature algorithms in the current round of NIST call for national standards.

Alg. types	Algorithms names	Alg. subtypes	Claimed NIST security level	Public key size (bytes)	Private key size (bytes)	Signature size (bytes)
Digital signature	CRYSTALS-DILITHIUM	Lattice-based	2, 3, 5	1312~2592	2528~4864	2420~4595
	FALCON	Lattice-based	1, 5	897~1793	1281~2305	690~1330
	Rainbow	Multivariate-based	1, 3, 5	60192~1930600	64~1408736	66~212
	Picnic	Hash-based	1, 3, 5	33~65	49~97	34036~209510
	SPHINCS+	Hash-based	1, 3, 5	32~64	64~128	7856~49856
	GeMSS	Multivariate-based	1, 3, 5	352190~3135590	128~256	258~600

2.3.2 PQ KEM Algorithms. In the current round of NIST call, there are nine PQ KEM algorithms, which can be divided into three following groups.

- **Lattice-based cryptosystems.** The lattice problems that NIST PQ KEM algorithms depend on are Learning With Errors (LWE) [42], Ring Learning With Errors (Ring-LWE) [43] and Module Learning with Rounding (MLWR) problem [44] with different lattices. And the list of NIST PQ KEM algorithms includes KYBER, NTRU, SABER, FrodoKEM and NTRU Prime, which have similar advantages just like lattice-based PQ signature algorithms.
- **Code-based cryptosystems.** This family of algorithms are essentially based on the theory that supports error-correction codes. In the current round of NIST call, Classic McEliece is an example of code-based cryptosystem that dates back from 1970s and whose security is based on the syndrome decoding problem [45]. Classic McEliece provides fast encapsulation and relatively fast decapsulation, but is burdened with huge sizes of public and private key sizes. To ease this issue, the PQ KEM algorithms BIKE and HQC based on different codes have been proposed and selected as the alternate candidate algorithms in the current round of NIST.
- **Supersingular Elliptic Curve Isogeny (SECI) cryptosystems.** The SECI cryptosystems are based on the isogeny protocol for ordinary elliptic curves and enhanced to withstand the quantum attack detailed in [46]. The only one isogeny-based PQ KEM algorithm passed to the current round of NIST call is SIKE, which is founded on pseudo-random walks in supersingular isogeny graphs.

As shown in Table 2, we summarize the details of all the PQ KEM algorithms in the current round of NIST call. Just like the PQ signature algorithms, all the PQ KEM algorithms can achieve different NIST security levels by using different key lengths and related parameters.

Table 2: Details of PQ KEM algorithms in the current round of NIST call for national standards.

Alg. types	Algorithms names	Alg. subtypes	Claimed NIST security level	Public key size (bytes)	Private key size (bytes)	Ciphertext size (bytes)	Shared secret size (bytes)
KEM	Classic McEliece	Code-based	1, 3, 5	261120~1357824	6452~14080	128~240	32
	CRYSTALS-KYBER	Lattice-based	1, 3, 5	800~1568	1632~3168	768~1568	32
	NTRU	Lattice-based	1, 3, 5	699~1138	935~1450	699~1138	32
	SABER	Lattice-based	1, 3, 5	672~1312	1568~3040	736~1472	32
	BIKE	Code-based	1, 3	2542~6206	3110~13236	2542~6206	32
	FrodoKEM	Lattice-based	1, 3, 5	9616~21520	19888~43088	9720~21632	16, 24, 32
	HQC	Code-based	1, 3, 5	2249~7425	2289~7285	4481~14469	64
	NTRU Prime	Lattice-based	2, 3, 4	897~1322	1125~1999	897~1184	32
	SIKE	SECI-based	1, 2, 3, 5	197~564	350~644	236~596	16, 24, 32

3 SYSTEM DESIGN, IMPLEMENTATION AND SECURITY ANALYSIS

This section presents the details regarding the design and implementation of chaincode-level PQ user authentication and key exchange system on Hyperledger Fabric. Based on the design and implementation, we introduce the threat model of the system and how we protect the on-chain data and chaincodes from attacks in the threat model.

3.1 System Design

Figure 1 shows the architecture of chaincode-level PQ user authentication and key exchange system, where we design a smart contract-based CA for cross-channel application users to request a 3-layer chain of PQ certificates (i.e., self-signed CA certificate, user ID certificate signed by the CA and KEM certificate signed by the user with given ID) with their personal information, authenticate each other based on the generated ID certificates, and perform key exchange by using the KEM certificates.

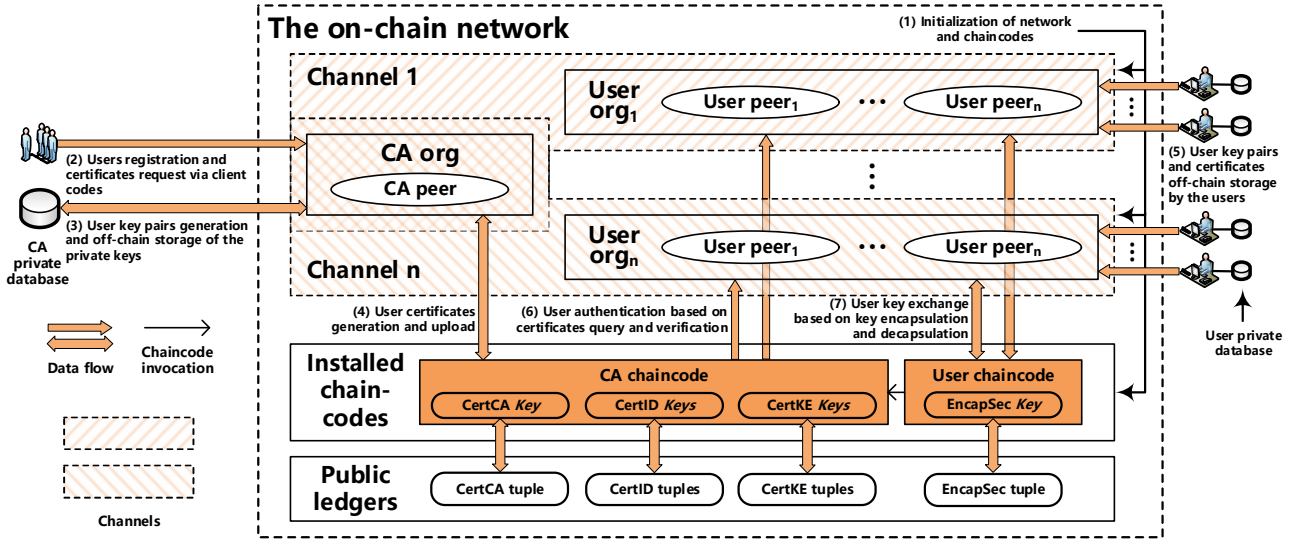

Figure 1: The architecture, work flow and data flow of the PQ user authentication and key exchange framework based on Hyperledger Fabric.

In the architecture, one CA organization/peer is responsible for providing the PQ certificate services (including generating certificate chains, querying, verifying and revoking the generated certificates), and can be added to multiple channels so as to provide the cross-channel certificate services for users executing chaincodes on the other organizations/peers in these channels.

To support the PQ certificate services, we redesign the X.509 certificate so as to include the information about the PQ algorithms and keys. Figure 2 shows the format of the PQ X.509 certificate, where the fields about PQ algorithms are added. The new certificate includes the PQ public key of the subject and the related algorithm as well as the specific PQ signature used to create the certificate. The certificate will be signed by the issuer, and the generated PQ signature is placed at the end of the certificate.

As shown in Figure 1, the execution of the chaincode-level PQ user authentication and key exchange system consists of seven phases: (1) *Initialization*, (2) *User registration and certificates request*, (3) *User key pairs generation and off-chain storage*, (4) *User certificate generation and on-chain storage*, (5) *User key pairs and certificates off-chain storage by the users*, (6) *User authentication based on certificates*, (7) *User key exchange based on key encapsulation/decapsulation*.

(1) *Initialization*: First of all, we deploy all the channels, organization and peers, and then install (and instantiate) the two chaincodes (i.e., `CA` chaincode and `User` chaincode) on all the peers. To provide the certificate services, we initialize the CA by invoking the `CA` chaincode to generate the PQ key pair and the related root certificate. The root CA certificate is stored on-chain under the CertCA *key* while the related key pair together with certificate information are stored off-chain in the private database maintained by the CA administrator. Since the on-chain data tuples are accessed via *keys*, users need to know the certificate *key* if one certificate query/verification is needed. For simplicity, we use the serial number of one certificate as its certificate *key* on the public ledger. To further enable access control, the CA administrator (and the user) need to provide one password for the generated key pair, and the password is also stored in the off-chain private database of the CA. We will detailedly introduce the security mechanisms of our system in Section 3.3.

(2) *User registration and certificates request*: After the initialization, the users provide their personal information (e.g., country, organization, locality and name) together with the password via client codes to register and request ID certificates based on the provided information. Moreover, the users can also request KEM certificates for their KEM public keys via client codes.

(3) *User key pairs generation and off-chain storage*: Upon receiving one certificate request, the CA peer generates the PQ key pair (i.e., ID or KEM) for the user and stores the key pair (together with the password) off-chain in the private database of CA.

(4) *User certificate generation and on-chain storage*: Based on the generated key pair, the CA peer signs the user ID public key (resp. user KEM public key) together with the user's personal information by using the private part of the CA root key (resp. user ID private key). Then, the generated user ID certificate (resp. user KEM certificate) is stored under the CertID *Key* (resp. CertKE *Key*) on-chain. In case of unexpected on-chain attacks, the generated certificate is also stored in the off-chain private database of the CA.

(5) *User key pairs and certificates off-chain storage by the users*: After the generation of user key pair and certificate, the generated key pair together with the certificate are returned to the user via client code and then stored in the private database maintained by the user.

(6) *User authentication based on certificates*: Relying on the on-chain certificates and services provided by the `CA` chaincode, the users could invoke the `User` chaincode to (further invoke the `CA` chaincode to) query and verify one user ID and KEM certificates with given serial numbers.

(7) *User key exchange*: Grounded on successful verification of the ID and KEM certificates, the user Alice generates and encapsulates a shared secret by using the public key in the KEM certificate of the user Bob. The encapsulated shared secret is then uploaded to the public ledger and stored under the EncapSec *Key*, which is concatenation of serial numbers of Alice and Bob ID certificates. After the encapsulation, the user Bob downloads the encapsulated shared secret, retrieves the KEM private key from his private database and finally decapsulates the shared secret.

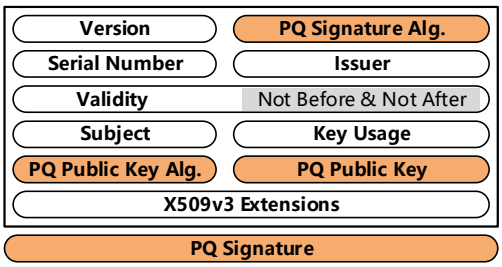

Figure 2: The format of PQ X.509 certificate.

3.2 System Implementation

The main implementation of the PQ user authentication and key exchange system consists of two chaincodes, namely `PqCa` and `PqUser`. As shown in Table 3, the `PqCa` chaincode provides APIs to generate the PQ key pairs together with the related certificates (for the CA, the user identity key and the user KEM key), query one certificate (based on a given serial number), verify one certificate (and its parent certificates in the certificate chain), revoke one certificate, encapsulate and decapsulate the shared secret between users. While the `PqUser` chaincode invokes the corresponding APIs in `PqCa` to query/verify one certificate and provides APIs to perform the encapsulation and decapsulation operations for the users.

The data structure in the off-chain private databases is designed relatively simply for further development. In the off-chain private databases (e.g., MySQL), one key pair, the corresponding certificate and related information (e.g., password) are all stored in one line, and the certificate serial number is set as the key of the line.

Besides the chaincodes, we also develop client codes to help developers create their applications based on the PQ user authentication and key exchange system. As shown in Table 3, there are two client codes, namely `PqCa_Cli` and `PqUser_Cli`, which are written in Go and used to invoke the APIs in the corresponding chaincodes. The configuration files for the client codes to invoke the chaincodes are also provided, and all of the chaincodes, client codes as well as the configuration files are available on [62].

3.3 Security Analysis

In this section, we introduce the threat model of our system, and then we present how we design and implement our system to protect on-chain data and chaincodes from attacks in the threat model.

Table 3: The chaincodes and client codes of PQ user authentication and key exchange system on Hyperledger Fabric.

Chaincodes/ Client codes	APIs	Descriptions
`PqCa/ PqCa_Cli .go`	Gen_ CertCA	Generate the key pair and certificate (self-signed) for the CA
	Gen_ CertID	Generate the key pair and ID certificate (signed by the CA) for each user
	Gen_ CertKE	Generate the KEM key pair and KE certificate for each user
	Query_ Cert	Query one certificate based on a given serial number
	Verify_ Cert	Verify one certificate and its parent certificates in the certificate chain
	Revoke_ Cert	Revoke one certificate based on a given serial number
`PqUser/ PqUser_ Cli.go`	Query_ Cert	Invoke chaincode `PqCa` and query one certificate with a given serial number
	Verify_ Cert	Invoke chaincode `PqCa` and verify one certificate (together with the related certificate chain)
	Encap	Perform the encapsulation operation and upload encapsulated shared secret
	Decap	Download the encapsulated share secret and perform decapsulation operation

3.3.1 Threat Model. In the system, all the private data (i.e., private key and password) are stored off-chain, and only certificates are stored in the public ledgers on-chain, therefore the on-chain attacker cannot violate the secrecy of the on-chain data.

Besides secrecy, the malicious peers may also want to violate other security properties that the system intends to preserve. Possible attacks from malicious peers are listed as follows.

- **Malicious user peer** may want to impersonate the CA peer and invoke `CA` chaincode to modify/update the certificates stored under CertCA, CertID, and CertKE *keys* (bound to the `CA` chaincode) belonging to CA, him/herself or other users.
- Both **malicious CA peer** and **malicious user peer** may want to invoke the `User` chaincode in order to modify/update the encapsulated shared secret stored under the EncapSec *keys* (bound to the `User` chaincode) owned by other user peers.
- **All malicious peers** may want to modify/update one chaincode to bypass the security check in the original chaincode.

3.3.2 System Security Mechanisms. Based on the threat model, we develop security mechanisms deployed in the chaincodes and client codes of our system to protect the integrity of chaincodes and on-chain data.

- **Integrity of certificates under certificate *keys*.** We perform access control in the `CA` chaincode, to which the certificate *keys* (i.e., CertCA, CertID and CertKE *keys*) are bound, by using the *GetCreator*() API (in the *shim* package) in order to make sure that only the CA peer can invoke the certificate generation and revocation APIs in the `CA` chaincode.
- **Integrity of encapsulated shared secret under EncapSec *key*.** To protect the integrity of the encapsulated shared secret, we again check the identity of the `User` chaincode invoker by using the *GetCreator*() API in order to make sure that only the designated user peer can get access to specific EncapSec *key* bound to the User chaincode. Furthermore, one developer can calculate one Message Authentication Code (MAC) based on the shared secret together with timestamp and also store the MAC in the EncapSec *key*, in order to further protect the integrity of the encapsulated shared secret. For sake of simplicity, we deploy no MAC mechanism.
- **Integrity of chaincodes.** To prevent any modification to or substitution of the installed chaincodes by malicious peers, it is needed to pre-record all the version numbers of the chaincodes, and check the version numbers during the execution of PQ-KES4Chain in client codes, which developers use to invoke the chaincodes. If a mismatch of the chaincode version numbers is detected, then the invoking of the chaincodes and the execution of PQ-KES4Chain should be ceased.

4 PERFORMANCE EVALUATION

In this section, we evaluate the performance of essential operations in our system focusing on the execution time and on-chain storage. During the analysis, we highlight the performance of each PQ signature and KEM algorithms on the consortium blockchain. Finally, we discuss the implications of our findings based on the experiment results.

The experiments involve three VMs on the local server, and each VM is based on Ubuntu 18.04 configured with eight CPU cores of Intel Xeon Gold 6131 throttled to 2.6GHz and 16GB memory, and we let one CA peer (for CA administrator) and two user peers (for users Alice and Bob) in one channel separately running on the three VMs. The system is built on Hyperledger Fabric 2.2.3, and the chaincodes together with the client codes are written in Go (1.15.7). We utilize the latest *liboqs* 0.6.0 library [63] and its Go wrapper [64] to generate PQ key pair, sign/verify the PQ signature and encapsulate/decapsulate the shared secret. Since the *liboqs* library is incompatible with the native Hyperledger Fabric docker image for chaincode execution, we build a new docker image integrated with *liboqs* based on Ubuntu 18.04 and use it as the execution environment of our chaincodes. The versions of docker and docker-compose we use are 19.03.13 and 1.25.0-rc1. For developers, the configuration files for the client codes, the docker file to generate the new docker image and information about how to use the new image are also available on [62].

4.1 PQ Algorithms and Parameter Sets Studied

To achieve different NIST security levels (i.e., level 1~5), all the PQ algorithms offer different key lengths and parameter sets. In our study, we divide all the PQ algorithms into three groups, namely level 1~2, level 3, level 4~5, which separately indicate low, medium and high security levels. We summarize the specified PQ algorithm names by groups in Table 4.

Table 4: Specific PQ Algorithm names in our analysis grouped by NIST security levels.

Levels	Sig. Alg. Names	KEM Alg. Names
Level 1~2	(1). Dilithium2-AES (2). Falcon-512 (3). picnic3_L1 (4). SPHINCS+-SHA256-128f-robust (5). SPHINCS+-SHA256-128f-simple	▪ Kyber512-90s ▪ NTRU-HPS-2048-509 ▪ LightSaber-KEM ▪ BIKE1-L1-FO ▪ FrodoKEM-640-SHAKE ▪ HQC-128 ▪ ntrulpr653 ▪ sntrup653 ▪ SIKE-p434 ▪ SIKE-p503
Level 3	(6). Dilithium3-AES (7). picnic3_L3 (8). SPHINCS+-SHA256-192f-robust (9). SPHINCS+-SHA256-192f-simple	▪ Kyber768-90s ▪ NTRU-HPS-2048-677 ▪ Saber-KEM ▪ BIKE1-L3-FO ▪ FrodoKEM-976-SHAKE ▪ HQC-192 ▪ ntrulpr761 ▪ sntrup761 ▪ SIKE-p610
Level 4~5	(10). Dilithium5-AES (11). Falcon-1024 (12). picnic3_L5 (13). SPHINCS+-SHA256-256f-robust (14). SPHINCS+-SHA256-256f-simple	▪ Kyber1024-90s ▪ NTRU-HPS-4096-821 ▪ FireSaber-KEM ▪ FrodoKEM-1344-SHAKE ▪ HQC-256 ▪ ntrulpr857 ▪ sntrup857 ▪ SIKE-p751

In our experiments, we choose one PQ signature algorithm and one KEM algorithm at the same security level (i.e., in the same line of Table 4). It should be mentioned that we exclude the PQ signing algorithms Rainbow, GeMSS and the PQ KEM algorithm Classic McEliece from our analysis, because these three algorithms (with any key length and parameter set) cause much more execution time and on-chain storage than other algorithms, make the figures almost unreadable and are not suitable for the consortium blockchains. In case of any reader who is interested in the excluded experiment results, we provide a full experiment results list which is also available on [62].

4.2 Performance Analysis - Level 1~2

Firstly, we study the performance of each operation in our system based on the studied PQ algorithms at NIST security level 1~2. The user operations in the analysis include CertCA/ID generation, CertKE generation, certificate chain verification, CertCA/ID query, CertKE query, certificate revoke, encapsulation and decapsulation, which are all essential operations of our system, and we believe that the readers can easily find the correspondence between these user operations and the seven steps of our system depicted in Figure 1 and Section 3.1. Since the certificates CertCA

and CertID are both generated based on signing private key and signed public key of the same PQ signature algorithm, the generation and query time of these two certificates should be the same, and therefore we only consider the average generation/query time of CertCA and CertID in the analysis.

4.2.1 Execution Time. For quick understanding, we summarize the execution time of all the user operations in Figure 3, where one PQ KEM algorithm name is prefixed with one PQ signature algorithm number that is enumerated in Table 4. As shown in Figure 3, the execution time of all the user operations is within 100 milliseconds and the operations based on some specific algorithms even finish execution within 20 milliseconds. This means that the chaincode-based PQ user authentication and key exchange system is time-efficient for other blockchain-based applications relying on our system.

Based on the execution time in Figure 3, one can easily find that our system based on PQ signature algorithm Dilithium2-AES enumerated as (1) (resp. Falcon-512 enumerated as (2)) has the shortest (resp. the second shortest) execution time of the certificate-related operations. On the other hand, the PQ KEM algorithms Kyber512-90s, NTRU-HPS-2048-509, LightSaber-KEM, HQC-128, ntrulpr653 and sntrup653 have better performance on the encapsulation and decapsulation time than the other PQ KEM algorithms in our system.

However, the execution time we summarized in Figure 3 is the total time of one user operation execution, which mainly consists of basic PQ public-key cryptographic operations (e.g., key pair generation, signing/verification and encapsulation/decapsulation) and public ledger access operations (e.g., uploading/downloading the generated certificates and encapsulated shared secrets by using $PutState()$ and $GetState()$ in Hyperledger Fabric).

In order to look into one user operation execution, we further test the execution time of the basic PQ public-key cryptographic operations (i.e., signing/KEM key pair generation, CertCA/ID/KE signing and verification) in the corresponding user operations (i.e., CertCA/ID/KE generation and certificate chain verification), and one can roughly get the execution time of public ledger access operations (i.e., storing/retrieving certificates) by a subtraction. For simplicity, we do not test the execution time of the basic encapsulation/decapsulation operations, because the patterns can be summarized (as follows) based on what we have tested.

Again, for quick understanding, we summarize the related results in Figure 4, where the basic PQ public-key cryptographic operations execution time are compared with the total execution time of the corresponding user operation. To enable a clear look at the comparison, we do not include the test results based on the PQ signature algorithm SPHINCS+. As shown in Figure 4, with a given certificate size (i.e., thousands of bytes) based on most PQ signature algorithms (detailed in the following subsection), the execution time of one certificate storage/retrieval operation is around 1 millisecond, which takes a big part of the user operations execution time based on fast PQ signature algorithms (e.g., Dilithium2-AES and Falcon-512) but can be ignored in the user operations based on slow PQ signature algorithms (e.g., picnic3_L1 and so on).

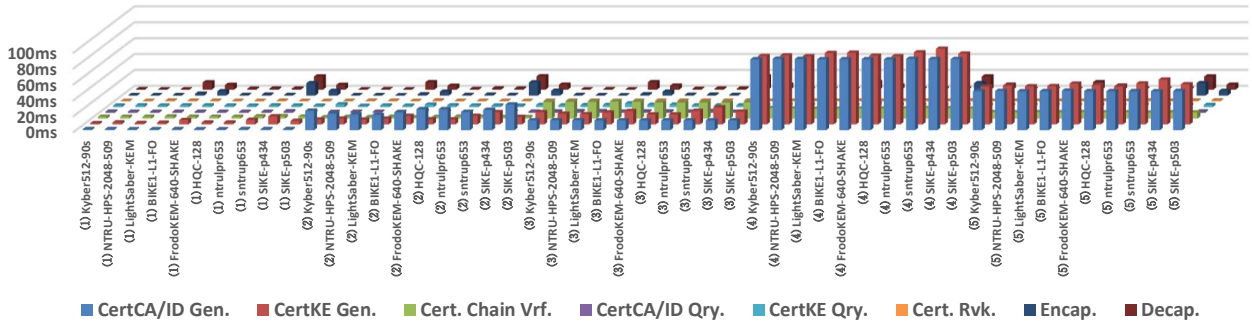

Figure 3: The execution time of each operation in our system based on the studied PQ algorithms at NIST security level 1~2.

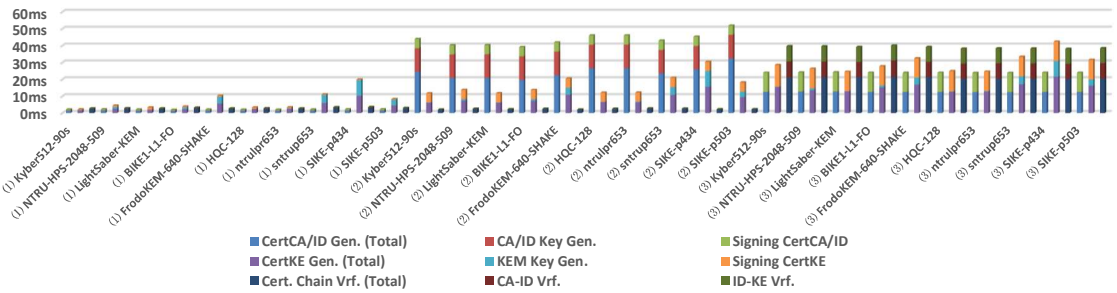

Figure 4: The execution time of basic PQ public-key cryptographic operations compared with the total execution time of one user operation in our system based on the studied PQ algorithms at NIST security level 1~2.

We record all the execution time and on-chain storage spaces in Table 5 attached as Appendix A. In the three pair of brackets at the same line of the table, we highlight the consumed time of one PQ signature key pair generation operation together with one signing certificate (for PQ signature public key) operation, one PQ KEM key pair generation operation together with one signing certificate (for PQ KEM public key) operation, and one verifying certificate (for PQ signature public key) operation together with one verifying certificate (for PQ KEM public key) operation in boldface.

4.2.2 On-chain Storage. We calculate the on-chain storage needed by our system based on different PQ public-algorithms at NIST security level 1~2. The on-chain storage space consists of three types of data, namely CertCA/ID (certificate for CA and user ID), CertKE (certificate for user key exchange) and EncapSec (the encapsulated shared secret) stored under the *keys* with the same name. A full list of the on-chain storage space sizes can also be found in Table 5 attached as Appendix A.

To enable fast understanding, we summarize the on-chain storage space of our system based on different studied PQ public-key algorithms at NIST security level 1~2 in Figure 5. As we can see in Figure 5, the certificates based on the PQ signature algorithm Falcon-512 have the smallest sizes (i.e., mostly thousands of bytes) followed by the certificate sizes based on Dilithium2-AES. On the other hand, the storage-efficient PQ KEM algorithms include Kyber512-90s, NTRU-HPS-2048-509, LightSaber-KEM, ntrulpr653, sntrup653, SIKE-p434 and SIKE-p503, which provide almost equally small size (i.e., hundreds of bytes) of the encapsulated shared secrets stored on-chain and further affect the CertKE sizes due to their small size (i.e., less than 5000 bytes) of KEM public keys as a part of the CertKE.

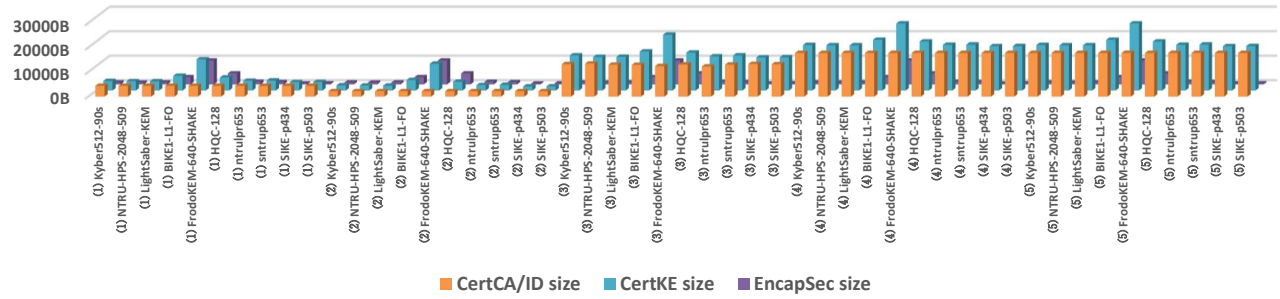

Figure 5: The on-chain storage space of our system based on the studied PQ algorithms at NIST security level 1~2.

4.3 Performance Analysis - Level 3 and Level 4~5

Due to page limit and to avoid repetition, we do not detail the performance analysis for level 3 and level 4~5, because the performance pattern of the system based on PQ algorithms at level 3 and 4~5 is quite similar to the pattern based on algorithms at level 1~2. However, it should be mentioned that FALCON only provides parameter sets to support NIST security level 1 and 5, so if one intends to develop applications using mediate-security-level (i.e., level-3) PQ signature algorithm, Dilithium3-AES seems to be only choice owing to the bad performance of the other PQ signature algorithms at level 3. All the execution time and the on-chain storage space based on PQ public-key algorithms at NIST security level 3 and 4~5 is separately recorded in Table 6 and Table 7 attached as Appendix B and Appendix C.

4.4 Discussion about Experiment Results

Based on the performance experiment results and intuitive analysis, we discuss and summarize the implications of our findings, which could be helpful for the PQ blockchain-based application developers, the undergoing NIST call and the developers of the PQ algorithms involved in the call.

- **For PQ application developers.** Since on-chain storage size would affect the execution time of user operations, PQ algorithms (e.g., Rainbow, GeMSS and Classic-McEliece) with huge size of public key or signature/encapsulated shared secret (which are normally stored on-chain) cause an overload of public ledger uploading/downloading time included in the user operation execution time, even if some algorithm (e.g., Rainbow) has pretty good execution time of basic cryptographic operations (e.g., signature generation/verification) when being tested on a local machine without public ledger [61]. Therefore, based on our overall test results of user operations on consortium blockchain, PQ application developers should choose the PQ signature algorithm CRYSTALS-DILITHIUM (resp. FALCON) if the application is time-sensitive (resp. on-chain storage-sensitive), while four PQ KEM algorithms (i.e., CRYSTALS-KYBER, NTRU, SABER and NTRU Prime) provide almost equally good overall performance on execution time and on-chain storage for the application developers.
- **For the undergoing NIST call.** As mentioned in the above item, the PQ algorithms that we recommend for blockchain-based applications are all lattice-based. To avoid putting all the eggs in one basket, NIST is also planning to take an extra (fourth) round of evaluation to standardize some PQ alternative

candidate algorithms for various reasons (including a broader range of hardness assumptions). Based on the overall performance on consortium blockchain, the hash-based signature algorithm Picnic is competitive on both execution time and on-chain storage. On the other hand, the code-based KEM algorithm HQC has better performance on execution time but worse performance on on-chain storage size, while the performance of the SECI-based KEM algorithm SIKE is just the reverse. Therefore, if the security rationales hold, NIST could standardize two PQ KEM algorithms (i.e., HQC and SIKE) for different application scenarios (e.g., blockchain-based) in the fourth round.

- **For PQ algorithm developers.** Based on the characteristics of blockchain-based applications, we make some suggestions for the PQ algorithm developers to help them improve their algorithms to meet the requirements of the blockchain-based systems. (1) *Small size of encapsulated shared secret is more favorable*. In most application scenarios only one CertID and one CertKE are needed for one user but the user needs to encapsulate/decapsulate many times to exchange keys with other users (during the validity period of the certificates). The number of the encapsulated shared secrets stored on-chain grow exponentially based on the user numbers, and hence small encapsulated shared secret size greatly helps reduce the total on-chain storage space of the system. (2) *Longer shared secret length is more favorable*. Since the shared secret (normally 128~256 bytes) is further used as the symmetric key (e.g., AES key) to encrypt/decrypt on-chain data, to provide enough post-quantum security level, the length of the symmetric key (i.e., the shared secret) must meet corresponding requirements (e.g., 256-bit AES key). Therefore, to achieve the same post-quantum security level, a longer shared secret length can efficiently reduce the number of KEM sessions in the system. (3) *Private key size could be designed longer for better performance*. Normally, private key is not stored on the public ledger, so it has no impact on the on-chain storage. The PQ algorithm developers could upgrade their algorithms by increasing the private key size in return for shorter public key/signature size, shorter encapsulated shared secret size, longer shared secret length or better execution time. In sum, our suggestions could also help the improved PQ algorithms to be more competitive in the extra round of NIST evaluation for standardization.

5 RELATED WORK

5.1 Post-quantum Blockchains

Commercial blockchains and related initiatives have analyzed and addressed the impact of quantum computers. As an example, Bitcoin PQ [47] is an experimental branch of Bitcoin's main blockchain that makes use of the PQ digital signature algorithm XMSS [48], which fails to advance to the third (current) round of NIST call. Just like Bitcoin, another commercial blockchain (i.e., Quantum-Resistant Ledger [49]) also utilizes XMSS as its signature algorithm. Another example is Ethereum 3.0, which plans to utilize PQ zero-knowledge proof technique zk-STARKs (zero-knowledge Scalable Transparent ARguments of Knowledge) to improve its scalability and data privacy [50].

Many researchers have also developed various PQ signature algorithms (e.g., Blockchained PQ Signature [17], Smart Digital Signature [18], Unconditionally Secure Signature [19], PQ Adaptor Signature [20], PQ Linkable Ring Signature [21]) that are claimed to be suitable for blockchains and then build blockchains based on their PQ signature algorithms.

While other works apply the PQ candidate signature algorithms involved in the current round of NIST call to the existing consortium blockchain systems. Blockchain platforms like Abelian [51] have suggested using lattice-based PQ cryptosystems (i.e., CRYSTALS-DILITHIUM together with other PQ algorithms) to prevent quantum attacks, while Corda [52] is experimenting with PQ algorithms like SPHINCS+. [33] firstly tries to build PQ PKI for Hyperledger Fabric based on the lattice-based digital signature algorithm qTESLA that also fails to advance to the third (current) round of the NIST call, and then [23] proposes a redesign of the underlying credential-management procedures of Hyperledger Fabric in order to incorporate hybrid digital signatures (i.e., protection against both classical and quantum attacks using two signature schemes) that include both classical signature and multiple PQ signature algorithms (i.e., CRYSTALS-DILITHIUM, FALCON, qTESLA) involved in the current or previous round of the NIST call.

In sum, most of the efforts try to integrate PQ signature algorithms with the underlying consortium blockchain systems in order to provide PQ membership authentication and data traceability, which are considered more urgent as the security footstone of blockchains.

5.2 Post-quantum Applications and Evaluation Based on Blockchains

Researches have been done to develop applications by utilizing PQ digital signature algorithms to provide user identities and keys management service running on blockchain. [53] proposes an e-voting system based on blockchain providing transparency in the process of voting and adopting the code-based digital signature algorithm Niederreiter [54] to resist quantum attacks. In [55], the authors prototype a blockchain-based PKI system for users by making use of the PQ lattice-based digital signature scheme GLP [56]. Shen et al. [57] apply Rainbow to Ethereum platform as the user public/private key pair generation and transaction signing/verification algorithm, and further compare the signature efficiency of Rainbow with the original algorithm ECDSA.

A few researchers try to deploy PQ key exchange algorithms to blockchains in order to provide key agreement mechanisms for blockchain users to resist quantum attacks. The authors in [58] use the Supersingular Isogeny Diffie-Hellman Key Exchange (SIDH) to protect the sensitive industrial data on the security framework proposed in the paper for distributed systems like blockchain. In [59], the authors propose a PQ solution, which provides end-to-end encryption for freely shared and published online digital data

on blockchain (Ethereum), by using the PQ Password-based Authenticated Key Exchange (PAKE) [60] protocol over rings and ideal lattices.

Very little research work [61] tries to make a systematic evaluation and comparison on the performance of each PQ digital signature and KEM algorithms in the current/previous round of the NIST call. However, its evaluation is not performed on the blockchain and it is strange that different PQ algorithms are evaluated on computers with various CPUs rather than with uniform hardware configuration.

To sum up, there is a lack of work focusing on exploring application-level PQ security system based on both digital signature and KEM algorithms in the NIST call in order to perform user authentication and preserve user data secrecy, which are also crucial when blockchain-based applications are developed, not even to mention a full evaluation on the security and performance of each candidate algorithm at the application layer of consortium blockchain

6 CONCLUSION

In this paper, we proposed the first (PQ) user authentication and key exchange system for consortium blockchain providing universal PQ certificate and key exchange services for all the users in the system. In our system, we integrated the services with all the PQ public-key (i.e., signature and encryption/KEM) algorithms in the current round of NIST call for national standard. We implemented our proposed system based on Hyperledger Fabric, and provided chaincodes, related APIs together with client codes for further development. Last but not least, we performed a systematic evaluation on the chaincodes execution time and on-chain storage space of the system, and discussed the implications of our findings which are helpful for the PQ blockchain-based application developers, the undergoing NIST call and the developers of the PQ algorithms.

ACKNOWLEDGMENTS

This research is supported by the National Key R&D Program of China (Program No. 2018YFC604005), the National Natural Science Foundation of China (Grant No. 61902285, 62004077) and the Fundamental Research Funds for the Central Universities (Program No. 2662018QD043)

REFERENCES

[1] Mingxiao Du, Qijun Chen, Jieying Chen, and Xiaofeng Ma. 2020. An Optimized Consortium Blockchain for Medical Information Sharing. *IEEE Transactions on Engineering Management* (early access), (2020), 1–13.

[2] Keke Gai, Yulu Wu, Liehuang Zhu, Meikang Qiu, and Meng Shen. 2020 Privacy-Preserving Energy Trading Using Consortium Blockchain in Smart Grid. *IEEE Transactions on Industrial Informatics* 15, 6 (2019), 3548–3358.

[3] Aiqing Zhang and Xiaodong Lin. 2018. Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain. *Journal of medical systems* 42, 8 (2018), 140.

[4] Shaoming Zhang, Jieqi Rong, and Baoyi Wang. 2020. A privacy protection scheme of smart meter for decentralized smart home environment based on consortium blockchain. *International Journal of Electrical Power & Energy Systems* 121, (2020), 1–10.

[5] Michele Mosca. 2018. Cybersecurity in an era with quantum computers: will we be ready? *IEEE Security & Privacy* 16, 5 (2018), 38–41.

[6] Peter W. Shor. 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing* 26, 5 (1997), 1494–1509.

[7] [n.d.]. PQCrypto. http://pqcrypto.eu.org.

[8] [n.d.]. SAFEcrypto. https://www.safecrypto.eu.

[9] [n.d.]. CryptoMathCREST (Mathematical Modelling for Next-Generation Cryptography). http://crypto.mist.i.u-tokyo.ac.jp/crest/english/index.html.

[10] [n.d.]. PROMETHEUS. https://www.h2020prometheus.eu.

[11] [n.d.]. Terms of Reference for ETSI TC Cyber Working Group for Quantum-Safe Cryptography. https://portal.etsi.org/TB-SiteMap/CYBER/CYBER-QSC-ToR.

[12] [n.d.]. Crypto Forum Research Group. https://irtf.org/cfrg.

[13] [n.d.]. ISO/IEC JTC 1/SC 27 (Information security, cybersecurity and privacy protection). https://www.iso.org/committee/45306.html.

[14] [n.d.]. IEEE 1363.1-2008 (IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices). https://standards.ieee.org/standard/1363_1-2008.html.

[15] [n.d.]. Post-Quantum Cryptography. https://csrc.nist.gov/Projects/post-quantum-cryptography.

[16] [n.d.]. Post-Quantum Cryptography (Round 3 Submissions). https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[17] Konstantinos Chalkias, James Brown, Mike Hearn, Tommy Lillehagen, Igor Nitto, and Thomas Schroeter. 2018. Blockchained Post-Quantum Signatures. In *iThings/GreenCom/CPSCom/SmartData*. Halifax, NS, Canada, 1196–1203.

[18] Furqan Shahida and Abid KhanSmart. 2020. Smart Digital Signatures (SDS): A Post-quantum Digital Signature Scheme for Distributed Ledgers. *Future Generation Computer Systems* 111, (2020), 241–253.

[19] Xin Sun, Mirek Sopek, Quanlong Wang, and Piotr Kulicki. 2019. Towards quantum-secured permissioned blockchain: Signature, consensus, and logic. *Entropy* 21, 9, (2019), 887.

[20] Muhammed F. Esgin, Oğuzhan Ersoy, and Zekeriya Erkin. 2020. Post-Quantum Adaptor Signatures and Payment Channel Networks. In *ESORICS*. Guildford, UK, 378–397.

[21] Wilson Alberto Torres, Ron Steinfeld, Amin Sakzad, and Veronika Kuchta. 2020. Post-Quantum Linkable Ring Signature Enabling Distributed Authorised Ring Confidential Transactions in Blockchain. *IACR Cryptology Eprint Archive* 2020 (2020), 1121.

[22] Robert Campbell. 2019. Transitioning to a Hyperledger Fabric Quantum-Resistant Classical Hybrid Public Key Infrastructure. *The Journal of British Blockchain Association* 2, 2, (2019), 1–11.

[23] Amelia Holcomb, Geovandro Pereira, Bhargav Das, and Michele Mosca. 2021. PQFabric: A Permissioned Blockchain Secure from Both Classical and Quantum Attacks. In *ICBC*. Virtual Conference.

[24] [n.d.]. CRYSTALS-DILITHIUM. https://pq-crystals.org/.

[25] [n.d.]. FALCON. https://falcon-sign.info/.

[26] [n.d.]. Rainbow. https://www.pqcrainbow.org/.

[27] [n.d.]. Classic McEliece. https://classic.mceliece.org.

[28] [n.d.]. CRYSTALS-KYBER. https://pq-crystals.org/kyber/index.shtml.

[29] [n.d.]. NTRU. https://ntru.org/.

[30] [n.d.]. SABER. https://www.esat.kuleuven.be/cosic/pqcrypto/saber/.

[31] [n.d.]. GeMSS. https://www-polsys.lip6.fr/Links/NIST/GeMSS.html.

[32] [n.d.]. Picnic. https://microsoft.github.io/Picnic/.

[33] [n.d.]. SPHINCS+. https://sphincs.org/.

[34] [n.d.]. BIKE. https://bikesuite.org.

[35] [n.d.]. NTRU Prime. https://ntruprime.cr.yp.to.

[36] [n.d.]. FrodoKEM. https://frodokem.org.

[37] [n.d.]. SIKE. https://sike.org.

[38] [n.d.]. HQC. https://pqc-hqc.org.

[39] M. Ajtai. 1996. Generating hard instances of lattice problems (extended abstract). In *STOC*. Philadelphia, PA, USA, 99–108.

[40] Vadim Lyubashevsky. 2009. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In *ASIACRYPT*. Tokyo, Japan, 598–616.

[41] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Greg Zaverucha. 2017. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In *CCS*. Dallas, Texas, USA, 1825–1842.

[42] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM* 56, 6, (2009), 1–40.

[43] Vadim Lyubashevsky, Chris Peikert, Oded Regev. 2013. On Ideal Lattices and Learning with Errors over Rings. *Journal of the ACM* 60, 6, (2013), 1–35.

[44] Adeline Langlois, Damien Stehle. 2015. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography* 75, (2015), 565–599.

[45] Elwyn R. Berlekamp, Robert J. McEliece, Henk C. A. Van Tilborg. 1978. On the inherent intractability of certain coding problems (Corresp.). *IEEE Transactions on Information Theory* 24, 3, (1978), 384–386.

[46] Andrew Childs, David Jao, Vladimir Soukharev. 2013. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology* 8, (2013), 1–29.

[47] [n.d.]. Bitcoin Post-Quantum. https://bitcoinpq.org/.
[48] Johannes Buchmann, Erik Dahmen and Andreas Hülsing. 2011. XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In *PQCrypto*. Taipei, Taiwan, 117–129.
[49] [n.d.]. The Quantum Resistant Ledger. https://www.theqrl.org/#.
[50] [n.d.]. EthHub zk-STARKs. https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-starks/.
[51] [n.d.]. Abelian. https://www.abelianfoundation.org.
[52] [n.d.]. Cipher suites supported by Corda. https://docs.corda.net/docs/corda-os/4.8/cipher-suites.html.
[53] Shiyao Gao, Dong Zheng, Rui Guo, Chunming Jing and Chencheng Hu. 2019. An Anti-Quantum E-Voting Protocol in Blockchain with Audit Function. *IEEE Access* 7, (2019), 115304–115316.
[54] Harald Niederreiter. 1986. Knapsack-type cryptosystems and algebraic coding theory. *Problems of control and information theory* 15, 2, (1986), 159–166.
[55] Hyeongcheol An, Rakyong Choi and Kwangjo Kim. 2018. Blockchain-Based Decentralized Key Management System with Quantum Resistance. In *WISA*. Jeju Island, Korea, 229–240.
[56] Tim Güneysu, Vadim Lyubashevsky and Thomas Pöppelmann. 2012. Practical lattice-based cryptography: A signature scheme for embedded systems. In *CHES*. Leuven, Belgium, 530–547.
[57] Ruping Shen, Hong Xiang, Xin Zhang, Bin Cai and Tao Xiang. 2018. Application and Implementation of Multivariate Public Key Cryptosystem in Blockchain (Short Paper). In *CollaborateCom*. London, UK, 419–428.
[58] Joe D. Preece and John M. Easton. 2018. Towards Encrypting Industrial Data on Public Distributed Networks. In *BigData*. Seattle, WA, USA, 4540–4544.
[59] Amir Hassani Karbasi and Siyamak Shahpasand. 2020. A post-quantum end-to-end encryption over smart contract-based blockchain for defeating man-in-the-middle and interception attacks. *Peer-to-Peer Networking and Applications* 13, (2020), 1423–1441.
[60] Reza Ebrahimi Atani, Shahabaddin Ebrahimi Atani and Amir Hassani Karbasi. 2019. A New Ring-Based SPHF and PAKE Protocol on Ideal Lattices. *The ISC International Journal of Information Security* 11, 1, (2019), 75–86.
[61] Tiago M. Fernández-Caramès and Paula Fraga-Lamas. 2020. Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks. *IEEE Access* 8, (2020), 21091–21116.
[62] [n.d.]. PqAuthKEM4Chain. https://github.com/sec-conf-auth/PqAuthKEM4Chain.
[63] [n.d.]. Liboqs. https://github.com/open-quantum-safe/liboqs.
[64] [n.d.]. Liboqs-go. https://github.com/open-quantum-safe/liboqs-go.

A Evaluation Results Based on PQ Algorithms at NIST Security Level 1~2

In Table 5, we record all the execution time and on-chain storage spaces of our system based on studied PQ public-key algorithms at NIST security level 1~2. In the three pair of brackets at the same line of the table, we highlight the consumed time of one PQ signature key pair generation operation together with one signing certificate (for PQ signature public key) operation, one PQ KEM key pair generation operation together with one signing certificate (for PQ KEM public key) operation, and one verifying certificate (for PQ signature public key) operation together with one verifying certificate (for PQ KEM public key) operation in boldface.

Table 5: The execution time (in MS) and on-chain certificate sizes (in byte) of the system based on PQ algorithms at NIST level 1~2.
NOTE: (1) denotes Dilithium2-AES, (2) denotes Falcon-512, (3) denotes picnic3_L1, (4) denotes SPHINCS+-SHA256-128f-robust, (5) denotes SPHINCS+-SHA256-128f-simple, and * denotes the same PQ signing algorithm as the above line. In the three pair of brackets at the same line, the consumed time of one PQ signature key pair generation operation, one signing certificate (for PQ signature public key) operation, one PQ KEM key pair generation operation, one signing certificate (for PQ KEM public key) operation, one verifying certificate (for PQ signature public key) operation, and one verifying certificate (for PQ KEM public key) operation are highlighted in boldface.

Sig. Alg.	KEM Alg.	CertCA/ ID Gen.	CertKE Gen.	Cert. Chain Verification	Cert. Qry.	Cert. Rvk.	Enc.	Dec.	CertCA/ ID size	CertKE size	EncSec size
(1)	Kyber512-90s	1.4 **(0.3, 0.5)**	1.7 **(0.1, 0.4)**	2.4 **(0.2, 0.2)**	1.6, 2.0	0.6	0.5	0.7	4380	3865	768
*	NTRU-HPS-2048-509	1.4 **(0.3, 0.5)**	2.6 **(1.1, 0.7)**	2.5 **(0.2, 0.2)**	1.4, 2.1	0.6	0.5	0.6	4380	3769	699
*	LightSaber-KEM	1.3 **(0.3, 0.5)**	1.9 **(0.1, 1.3)**	2.4 **(0.2, 0.2)**	1.4, 2.0	1.0	0.6	0.5	4380	3739	736
*	BIKE1-L1-FO	1.2 **(0.3, 0.5)**	2.3 **(1.0, 0.6)**	2.8 **(0.2, 0.2)**	1.4, 2.1	0.8	2.3	9.4	4380	6010	2946
*	FrodoKEM-640-SHAKE	1.4 **(0.3, 0.5)**	5.6 **(4.1, 0.6)**	2.4 **(0.2, 0.2)**	1.4, 2.2	0.7	5.5	6.1	4380	12687	9720
*	HQC-128	1.3 **(0.3, 0.5)**	2.4 **(0.2, 0.7)**	2.5 **(0.2, 0.2)**	1.4, 2.0	0.6	0.8	1.2	4380	5309	4481
*	ntrulpr653	1.3 **(0.3, 0.5)**	2.5 **(0.4, 0.5)**	2.4 **(0.2, 0.2)**	1.5, 1.8	0.6	1.2	1.5	4380	3960	1025
*	sntrup653	1.4 **(0.3, 0.5)**	6 **(4.7, 0.4)**	3.0 **(0.3, 0.2)**	1.5, 2.1	0.7	0.8	1.4	4380	4056	897
*	SIKE-p434	1.4 **(0.3, 0.5)**	10.2 **(9.3, 0.4)**	3.1 **(0.3, 0.2)**	1.9, 1.9	0.6	15.6	16.7	4380	3392	346
*	SIKE-p503	1.4 **(0.3, 0.5)**	4.5 **(3.4, 0.4)**	2.7 **(0.2, 0.2)**	1.4, 1.8	0.6	5.9	6.2	4380	3440	402
(2)	Kyber512-90s	24.5 **(14.0, 5.5)**	6.2 **(0.1, 5.4)**	1.9 **(0.1, 0.1)**	1.3, 3.6	0.6	0.5	0.6	2194	2094	768
*	NTRU-HPS-2048-509	20.8 **(14.0, 5.5)**	7.3 **(1.0, 5.5)**	2.3 **(0.2, 0.1)**	1.5, 1.6	0.8	0.5	0.7	2197	2001	699
*	LightSaber-KEM	19.9 **(14.0, 5.5)**	6.2 **(0.1, 5.5)**	2.1 **(0.1, 0.1)**	1.4, 2.0	0.7	0.4	0.5	2191	1971	736
*	BIKE1-L1-FO	19.2 **(14.0, 5.5)**	7.3 **(1.0, 5.4)**	2.4 **(0.1, 0.1)**	1.3, 1.8	0.6	1.5	9.5	2192	4240	2946
*	FrodoKEM-640-SHAKE	22.5 **(14.0, 5.5)**	10.9 **(4.1, 5.5)**	1.9 **(0.1, 0.1)**	1.7, 3.1	1.1	4.9	5.1	2194	10917	9720
*	HQC-128	26.6 **(14.0, 5.5)**	6.5 **(0.2, 5.4)**	2.2 **(0.2, 0.2)**	1.4, 2.2	0.6	0.8	1.0	2193	3540	4481
*	ntrulpr653	26.6 **(14.0, 5.5)**	6.4 **(0.4, 5.4)**	2.6 **(0.1, 0.1)**	1.5, 1.9	0.6	1.2	1.7	2197	2193	1025
*	sntrup653	23 **(14.0, 5.5)**	10.8 **(4.6, 5.5)**	2.5 **(0.1, 0.1)**	1.5, 1.9	0.6	0.8	1.4	2196	2291	897
*	SIKE-p434	25.8 **(14.0, 5.5)**	15.6 **(9.3, 5.5)**	2.1 **(0.1, 0.1)**	1.4, 1.8	0.6	16.5	16.9	2193	1627	346
*	SIKE-p503	32.4 **(14.0, 5.5)**	9.4 **(3.3, 5.5)**	2.1 **(0.1, 0.1)**	1.6, 2.7	0.7	6.1	6.3	2197	1678	402
(3)	Kyber512-90s	12.5 **(0.1, 11.4)**	15.5 **(0.2, 12.8)**	20.9 **(9.7, 9.1)**	1.8, 2.1	0.8	0.7	0.6	13212	14316	768
*	NTRU-HPS-2048-509	12.6 **(0.1, 11.4)**	13.6 **(1.1, 11.6)**	21.2 **(9.3, 9.1)**	2.2, 2.6	0.8	0.5	0.7	13452	13658	699
*	LightSaber-KEM	12.7 **(0.1, 11.4)**	12.9 **(0.1, 11.5)**	21.1 **(9.2, 8.9)**	2.2, 2.7	0.6	0.4	0.6	12988	13712	736
*	BIKE1-L1-FO	12.5 **(0.1, 11.4)**	15.2 **(1.1, 11.5)**	21.5 **(9.7, 8.9)**	1.7, 4.7	0.7	1.5	9.6	12926	15915	2946
*	FrodoKEM-640-SHAKE	12.4 **(0.1, 11.4)**	16.9 **(4.1, 11.5)**	21.2 **(9.1, 8.9)**	1.8, 3.0	0.7	4.9	5.3	12441	22771	9720
*	HQC-128	12.5 **(0.1, 11.4)**	12.6 **(0.3, 12.0)**	20.3 **(9.0, 8.9)**	1.9, 3.2	0.7	0.9	1.1	13096	15438	4481
*	ntrulpr653	12.4 **(0.1, 11.4)**	12.7 **(0.4, 11.5)**	20.4 **(9.1, 8.9)**	1.6, 2.2	0.8	1.1	1.4	12263	13928	1025
*	sntrup653	12.4 **(0.1, 11.4)**	17.1 **(4.7, 11.6)**	20.4 **(9.0, 8.9)**	2.3, 2.1	0.7	0.8	1.5	13022	14283	897
*	SIKE-p434	12.5 **(0.1, 11.4)**	21.6 **(9.3, 11.5)**	20.2 **(9.0, 8.9)**	1.5, 2.2	0.9	15.7	16.7	13322	13442	346
*	SIKE-p503	12.4 **(0.1, 11.4)**	16.2 **(3.7, 11.7)**	20.5 **(9.1, 8.9)**	2.2, 2.4	0.7	5.8	6.3	13179	13522	402
(4)	Kyber512-90s	89.3 **(3.5, 84.8)**	86.1 **(0.1, 85.0)**	13.4 **(5.3, 5.2)**	2.0, 2.8	0.8	0.5	0.6	17792	18546	768
*	NTRU-HPS-2048-509	90 **(3.5, 84.8)**	86.9 **(1.0, 84.6)**	12.7 **(5.2, 5.2)**	2.0, 3.0	1.0	0.5	0.7	17794	18452	699
*	LightSaber-KEM	89.8 **(3.5, 84.8)**	85.8 **(0.2, 85.3)**	12.4 **(5.2, 5.1)**	2.2, 2.8	0.7	0.5	0.7	17794	18422	736
*	BIKE1-L1-FO	89.3 **(3.5, 84.8)**	89.9 **(1.1, 84.8)**	12.6 **(5.1, 5.1)**	2.1, 2.6	1.0	1.6	9.5	17796	20695	2946
*	FrodoKEM-640-SHAKE	89.4 **(3.5, 84.8)**	90.3 **(4.0, 84.9)**	12.7 **(5.3, 4.9)**	2.2, 4.5	0.8	5.0	5.2	17796	27372	9720
*	HQC-128	89.4 **(3.5, 84.8)**	86.5 **(0.2, 85.0)**	12.8 **(5.2, 5.2)**	1.9, 2.9	0.9	0.8	1.1	17794	19992	4481
*	ntrulpr653	89.2 **(3.5, 84.8)**	85.9 **(0.4, 85.1)**	12.7 **(5.2, 5.1)**	1.8, 2.7	0.9	1.2	1.4	17792	18641	1025
*	sntrup653	90 **(3.5, 84.8)**	90.7 **(4.7, 84.8)**	13.3 **(5.3, 5.4)**	2.0, 2.6	0.8	0.9	1.5	17796	18741	897
*	SIKE-p434	89.6 **(3.5, 84.8)**	95.3 **(9.3, 85.4)**	13.2 **(5.0, 5.1)**	1.9, 2.8	1.2	15.6	16.7	17796	18077	346
*	SIKE-p503	89.5 **(3.5, 84.8)**	89.1 **(3.4, 84.6)**	12.6 **(4.9, 5.0)**	2.0, 2.6	0.7	5.9	6.3	17794	18123	402
(5)	Kyber512-90s	49.3 **(1.9, 46.7)**	47.7 **(0.1, 46.5)**	8.3 **(2.7, 2.7)**	1.9, 2.4	0.8	0.5	0.7	17794	18548	768
*	NTRU-HPS-2048-509	49.4 **(1.9, 46.7)**	48.6 **(1.2, 46.6)**	8.0 **(3.0, 2.6)**	1.9, 2.6	0.8	0.5	0.6	17794	18452	699
*	LightSaber-KEM	49.1 **(1.9, 46.7)**	48.1 **(0.1, 46.5)**	8.2 **(2.7, 2.8)**	2.0, 2.6	0.7	0.5	0.8	17794	18422	736
*	BIKE1-L1-FO	49.2 **(1.9, 46.7)**	48.3 **(1.1, 46.3)**	8.0 **(2.8, 2.7)**	2.1, 2.6	0.7	1.6	9.5	17794	20693	2946
*	FrodoKEM-640-SHAKE	49.9 **(1.9, 46.7)**	51.4 **(4.0, 46.2)**	8.2 **(2.7, 2.8)**	2.2, 3.0	0.9	4.9	5.3	17794	27370	9720
*	HQC-128	49.3 **(1.9, 46.7)**	49.6 **(6.2, 46.3)**	8.0 **(2.7, 2.8)**	1.8, 2.6	0.8	0.9	1.2	17794	19992	4481
*	ntrulpr653	49.2 **(1.9, 46.7)**	47.5 **(0.4, 46.3)**	8.0 **(2.8, 2.7)**	2.2, 3.1	1.2	1.1	1.5	17794	18643	1025
*	sntrup653	49.2 **(1.9, 46.7)**	51.6 **(4.7, 46.2)**	7.6 **(2.6, 2.7)**	2.1, 2.8	0.8	0.8	1.5	17794	18739	897
*	SIKE-p434	49.1 **(1.9, 46.7)**	56.4 **(9.3, 46.2)**	8.1 **(2.7, 2.6)**	2.1, 2.9	0.9	15.7	16.8	17794	18075	346
*	SIKE-p503	49.4 **(1.9, 46.7)**	50.5 **(3.3, 47.0)**	7.9 **(2.7, 2.7)**	1.8, 2.8	1.2	5.9	6.3	17794	18123	402

B Evaluation Results Based on PQ Algorithms at NIST Security Level 3

In Table 6, we record all the execution time and on-chain storage spaces of our system based on studied PQ public-key algorithms at NIST security level 3. In the three pair of brackets at the same line of the table, we also highlight the consumed time of one PQ signature key pair generation operation together with one signing certificate (for PQ signature public key) operation, one PQ KEM key pair generation operation together with one signing certificate (for PQ KEM public key) operation, and one verifying certificate (for PQ signature public key) operation together with one verifying certificate (for PQ KEM public key) operation.

Table 6: The execution time (in MS) and on-chain certificate sizes (in byte) of the system based on PQ algorithms at NIST level 3.
NOTE: (6) denotes Dilithium3-AES, (7) denotes picnic3_L3, (8) denotes SPHINCS+-SHA256-192f-robust, (9) denotes SPHINCS+-SHA256-192f-simple, and * denotes the same PQ signing algorithm as the above line. In the three pair of brackets at the same line, the consumed time of one PQ signature key pair generation operation, one signing certificate (for PQ signature public key) operation, one PQ KEM key pair generation operation, one signing certificate (for PQ KEM public key) operation, one verifying certificate (for PQ signature public key) operation, and one verifying certificate (for PQ KEM public key) operation are highlighted in boldface.

Sig. Alg.	KEM Alg.	CertCA/ ID Gen.	CertKE Gen.	Cert. Chain Verification	Cert. Qry.	Cert. Rvk.	Enc.	Dec.	CertCA/ ID size	CertKE size	EncSec size
(6)	Kyber768-90s	2.8 **(0.5, 0.7)**	1.6 **(0.2, 0.7)**	3.1 **(0.5, 0.4)**	2.1, 2.1	0.6	0.6	0.8	5832	5122	1088
*	NTRU-HPS-2048-677	1.9 **(0.5, 0.7)**	3.2 **(1.8, 0.7)**	3.0 **(0.5, 0.4)**	1.6, 2.1	0.7	0.5	0.7	5892	4873	930
*	Saber-KEM	2.3 **(0.5, 0.7)**	1.7 **(0.1, 1.2)**	3.1 **(0.5, 0.4)**	1.6, 2.5	0.7	0.5	0.6	5892	4927	1088
*	BIKE1-L3-FO	2.1 **(0.5, 0.7)**	6.1 **(4.3, 0.6)**	3.5 **(0.4, 0.5)**	1.7, 2.1	0.7	4.8	44.5	5892	10143	6206
*	FrodoKEM-976-SHAKE	2.2 **(0.5, 0.7)**	12.5 **(10.2, 0.8)**	3.3 **(0.5, 0.5)**	1.6, 2.5	0.7	10.3	11.5	5892	19576	15744
*	HQC-192	2.7 **(0.5, 0.7)**	3.7 **(0.5, 1.6)**	3.2 **(0.5, 0.5)**	1.5, 2.4	0.6	1.4	1.9	5892	8455	9026
*	ntrulpr761	2.2 **(0.5, 0.7)**	2.3 **(0.5, 0.6)**	3.3 **(0.5, 0.4)**	1.4, 2.0	0.7	1.4	1.7	5894	4977	1167
*	sntrup761	2.1 **(0.5, 0.7)**	7.6 **(6.3, 1.4)**	3.5 **(0.5, 0.4)**	1.5, 2.3	0.8	0.9	1.7	5892	5093	1039
*	SIKE-p610	2.3 **(0.5, 0.7)**	28.7 **(27.3, 0.8)**	3.3 **(0.6, 0.6)**	2.1, 1.3	0.7	51.1	51.1	5892	4397	486
(7)	Kyber768-90s	28.1 **(0.1, 26.3)**	29.2 **(0.2, 26.5)**	45.1 **(21.2, 21.3)**	2.0, 3.6	1.1	0.7	0.8	27216	28811	1088
*	NTRU-HPS-2048-677	28.6 **(0.1, 26.3)**	29.3 **(1.8, 26.6)**	50.3 **(23.4, 22.9)**	1.9, 2.7	1.3	0.5	0.7	28272	29018	930
*	Saber-KEM	27.4 **(0.1, 26.3)**	27.8 **(0.1, 26.4)**	45.6 **(21.4, 21.2)**	2.3, 2.7	0.8	0.6	0.7	28462	29718	1088
*	BIKE1-L3-FO	27.4 **(0.1, 26.3)**	32 **(4.2, 26.5)**	46.3 **(21.7, 21.4)**	2.4, 3.2	0.8	4.8	44.8	27862	34718	6206
*	FrodoKEM-976-SHAKE	28 **(0.1, 26.3)**	38 **(9.2, 26.7)**	46.7 **(23.4, 23.1)**	2.1, 5.5	1.0	10.6	11.2	27958	43647	15744
*	HQC-192	27.5 **(0.1, 26.3)**	28 **(0.4, 26.3)**	45.5 **(21.7, 21.2)**	2.0, 2.8	0.8	1.4	2.1	27838	32550	9026
*	ntrulpr761	29.3 **(0.1, 26.3)**	28.1 **(0.5, 26.8)**	45.4 **(21.3, 21.1)**	2.7, 2.9	0.8	1.2	1.7	27958	28998	1167
*	sntrup761	28 **(0.1, 26.3)**	34.2 **(6.4, 26.7)**	46.3 **(21.8, 21.7)**	2.5, 3.1	1.2	0.9	0.8	27934	28540	1039
*	SIKE-p610	27.7 **(0.1, 26.3)**	55.4 **(28.1, 26.4)**	46.2 **(21.2, 21.5)**	2.1, 2.6	0.9	51.1	51.0	27744	28590	486
(8)	Kyber768-90s	147.4 **(5.1, 140.2)**	143.1 **(0.2, 141.2)**	18.5 **(7.7, 8.0)**	2.2, 2.9	1.1	0.7	0.8	36385	37508	1088
*	NTRU-HPS-2048-677	147.3 **(5.1, 140.2)**	144.2 **(1.8, 141.2)**	18.6 **(7.9, 7.6)**	2.3, 3.1	1.1	0.6	0.7	36387	37261	930
*	Saber-KEM	148.2 **(5.1, 140.2)**	142.3 **(0.1, 140.7)**	18.1 **(7.7, 7.7)**	2.2, 3.4	0.9	0.6	0.7	36385	37313	1088
*	BIKE1-L3-FO	147.4 **(5.1, 140.2)**	146.6 **(4.2, 141.1)**	19.4 **(7.7, 7.8)**	2.4, 3.2	0.9	4.8	44.2	36385	42529	6206
*	FrodoKEM-976-SHAKE	147.2 **(5.1, 140.2)**	153.8 **(9.2, 140.8)**	19.4 **(7.8, 7.9)**	2.8, 3.9	0.9	10.4	10.9	36385	51963	15744
*	HQC-192	146.9 **(5.1, 140.2)**	144.9 **(0.5, 140.8)**	18.7 **(7.8, 8.1)**	2.4, 4.8	1.1	1.5	2.1	36385	40841	9026
*	ntrulpr761	147 **(5.1, 140.2)**	142.8 **(0.5, 141.2)**	18.4 **(7.6, 7.7)**	2.4, 3.8	1.0	1.3	1.7	36385	37361	1167
*	sntrup761	148.9 **(5.1, 140.2)**	148.5 **(6.4, 141.0)**	18.1 **(7.7, 7.8)**	2.2, 3.3	0.9	0.9	2.0	36387	37481	1039
*	SIKE-p610	147.4 **(5.1, 140.2)**	170.1 **(27.4, 140.9)**	18.8 **(8.0, 7.8)**	2.5, 3.5	1.0	50.9	51.1	36383	36781	486
(9)	Kyber768-90s	79.5 **(2.7, 75.9)**	77.1 **(0.2, 75.6)**	10.6 **(4.0, 3.8)**	2.1, 4.4	1.1	0.6	0.8	36385	37508	1088
*	NTRU-HPS-2048-677	79.7 **(2.7, 75.9)**	78.5 **(1.8, 75.7)**	10.9 **(3.9, 3.9)**	2.0, 3.5	0.9	0.6	0.6	36385	37259	930
*	Saber-KEM	79.9 **(2.7, 75.9)**	77.2 **(0.1, 75.7)**	10.2 **(3.9, 3.7)**	2.5, 3.2	0.9	0.5	0.7	36385	37313	1088
*	BIKE1-L3-FO	79.8 **(2.7, 75.9)**	81.4 **(4.1, 75.7)**	10.7 **(3.8, 3.9)**	2.7, 3.1	1.1	4.9	43.7	36387	42531	6206
*	FrodoKEM-976-SHAKE	80.6 **(2.7, 75.9)**	87 **(9.3, 75.8)**	11.9 **(3.9, 3.8)**	2.7, 3.4	0.9	10.3	10.9	36387	51946	15744
*	HQC-192	79.5 **(2.7, 75.9)**	78.4 **(0.5, 76.6)**	11.0 **(4.0, 3.9)**	2.8, 3.7	1.1	1.6	1.9	36385	40841	9026
*	ntrulpr761	79.7 **(2.7, 75.9)**	78 **(0.5, 75.7)**	10.9 **(3.9, 3.9)**	2.3, 3.5	1.0	1.3	1.8	36385	37361	1167
*	sntrup761	79.7 **(2.7, 75.9)**	83.9 **(6.4, 76.0)**	10.8 **(3.8, 3.9)**	2.3, 3.2	1.3	0.9	1.9	36383	37477	1039
*	SIKE-p610	79.7 **(2.7, 75.9)**	104.8 **(27.2, 75.7)**	11.0 **(3.9, 3.9)**	2.5, 3.5	1.1	50.9	51.1	36383	36781	486

C Evaluation Results Based on PQ Algorithms at NIST Security Level 4~5

In Table 7, we record all the execution time and on-chain storage spaces of our system based on studied PQ public-key algorithms at NIST security level 4~5. In the three pair of brackets at the same line of the table, we again highlight the consumed time of one PQ signature key pair generation operation together with one signing certificate (for PQ signature public key) operation, one PQ KEM key pair generation operation together with one signing certificate (for PQ KEM public key) operation, and one verifying certificate (for PQ signature public key) operation together with one verifying certificate (for PQ KEM public key) operation.

Table 7: The execution time (in MS) and on-chain certificate sizes (in byte) of the system based on PQ algorithms at NIST level 4~5.
NOTE: ⑩ denotes Dilithium5-AES, ⑪ denotes Falcon-1024, ⑫ denotes picnic3_L5, ⑬ denotes SPHINCS+-SHA256-256f-robust, ⑭ denotes SPHINCS+-SHA256-256f-simple, and * denotes the same PQ signing algorithm as the above line. In the three pair of brackets at the same line, the consumed time of one PQ signature key pair generation operation, one signing certificate (for PQ signature public key) operation, one PQ KEM key pair generation operation, one signing certificate (for PQ KEM public key) operation, one verifying certificate (for PQ signature public key) operation, and one verifying certificate (for PQ KEM public key) operation are highlighted in boldface.

Sig. Alg.	KEM Alg.	CertCA/ ID Gen.	CertKE Gen.	Cert. Chain Verification	Cert. Qry.	Cert. Rvk.	Enc.	Dec.	CertCA/ ID size	CertKE size	EncSec size
⑩	Kyber1024-90s	3.2 **(0.8, 0.9)**	3.2 **(0.2, 1.5)**	3.4 **(0.7, 0.7)**	1.5, 2.1	0.7	0.6	0.8	7835	6809	1568
*	NTRU-HPS-4096-821	3.1 **(0.8, 0.9)**	5.3 **(2.6, 2.1)**	4.1 **(0.7, 0.7)**	1.7, 2.3	0.7	0.5	0.6	7835	6475	1230
*	FireSaber-KEM	3.4 **(0.8, 0.9)**	2.2 **(0.2, 1.3)**	3.5 **(0.8, 0.7)**	1.5, 2.0	1.2	0.7	0.7	7835	6553	1472
*	FrodoKEM-1344-SHAKE	3.4 **(0.8, 0.9)**	21 **(17.3, 1.0)**	3.8 **(0.8, 0.8)**	1.7, 5.8	0.7	19.0	20.1	7835	26767	21632
*	HQC-256	2.7 **(0.8, 0.9)**	3.1 **(0.9, 1.9)**	6.4 **(0.7, 0.7)**	1.7, 2.4	1.2	2.2	3.2	7837	12482	14469
*	ntrulpr857	2.9 **(0.8, 0.9)**	2.7 **(0.6, 1.1)**	3.8 **(0.7, 0.7)**	1.6, 1.9	0.8	1.5	1.9	7835	6422	1312
*	sntrup857	2.7 **(0.8, 0.9)**	10.4 **(9.3, 1.5)**	3.5 **(0.7, 0.7)**	1.6, 2.2	0.7	1.0	2.0	7835	6559	1999
*	SIKE-p751	2.4 **(0.8, 0.9)**	12.1 **(10.4, 1.1)**	3.6 **(0.8, 0.7)**	1.8, 2.3	0.9	17.3	18.2	7835	5801	596
⑪	Kyber1024-90s	57.2 **(40.7, 12.0)**	12.6 **(0.3, 11.8)**	2.7 **(0.2, 0.2)**	1.6, 2.0	0.6	0.6	0.8	3711	3485	1568
*	NTRU-HPS-4096-821	51.9 **(40.7, 12.0)**	15.6 **(2.5, 11.8)**	2.2 **(0.2, 0.1)**	1.6, 1.9	0.7	0.5	0.6	3708	3151	1230
*	FireSaber-KEM	56.9 **(40.7, 12.0)**	12.7 **(0.2, 11.9)**	2.3 **(0.2, 0.1)**	1.6, 1.9	0.7	0.6	0.7	3711	3224	1472
*	FrodoKEM-1344-SHAKE	53 **(40.7, 12.0)**	30.8 **(17.4, 12.0)**	2.6 **(0.2, 0.2)**	1.5, 2.2	0.8	18.9	20.1	3711	23446	21632
*	HQC-256	50.2 **(40.7, 12.0)**	13.5 **(0.8, 11.8)**	2.2 **(0.2, 0.2)**	1.3, 2.7	0.7	2.2	4.6	3705	9152	14469
*	ntrulpr857	54 **(40.7, 12.0)**	13.1 **(0.6, 11.8)**	2.5 **(0.2, 0.1)**	1.4, 1.8	0.8	1.4	1.9	3710	3095	1312
*	sntrup857	62.8 **(40.7, 12.0)**	20.9 **(8.3, 12.2)**	2.1 **(0.2, 0.1)**	1.5, 2.3	0.9	1.0	1.9	3707	3233	1999
*	SIKE-p751	56.1 **(40.7, 12.0)**	24.7 **(10.2, 11.7)**	2.1 **(0.2, 0.1)**	1.5, 3.0	0.7	16.8	18.4	3707	2478	596
⑫	Kyber1024-90s	45.1 **(0.1, 43.3)**	45.4 **(0.3, 43.2)**	77.4 **(37.3, 36.9)**	2.7, 4.3	1.2	0.7	0.8	49985	49924	1568
*	NTRU-HPS-4096-821	46 **(0.1, 43.3)**	48.2 **(2.5, 43.7)**	72.3 **(34.3, 34.5)**	2.9, 3.7	1.2	0.6	0.7	49759	51284	1230
*	FireSaber-KEM	44.5 **(0.1, 43.3)**	46.9 **(0.2, 47.7)**	73.1 **(34.9, 34.6)**	3.3, 3.9	1.1	0.5	0.7	49685	49442	1472
*	FrodoKEM-1344-SHAKE	44.7 **(0.1, 43.3)**	62.8 **(17.7, 44.5)**	72.4 **(34.8, 34.2)**	2.9, 6.0	1.1	19.4	20.2	50463	71417	21632
*	HQC-256	45.8 **(0.1, 43.3)**	48.9 **(0.9, 43.5)**	73.9 **(35.6, 34.4)**	2.5, 4.6	1.2	2.3	3.2	50017	56971	14469
*	ntrulpr857	44.4 **(0.1, 43.3)**	45.2 **(0.6, 43.5)**	79.4 **(37.5, 36.7)**	2.9, 3.8	1.2	1.5	2.1	49571	50083	1312
*	sntrup857	45.3 **(0.1, 43.3)**	52.9 **(8.3, 43.7)**	73.9 **(35.7, 34.7)**	3.2, 3.7	1.1	1.0	2.2	49091	51276	1999
*	SIKE-p751	45 **(0.1, 43.3)**	72.7 **(10.2, 43.1)**	72.7 **(35.1, 34.4)**	3.1, 3.1	1.0	17.1	18.2	48163	50326	596
⑬	Kyber1024-90s	385.8 **(17.5, 368.6)**	368.1 **(0.2, 366.5)**	23.9 **(10.4, 10.3)**	2.7, 3.6	1.3	0.7	0.9	50592	52083	1568
*	NTRU-HPS-4096-821	386.2 **(17.5, 368.6)**	370.4 **(2.5, 366.8)**	25.3 **(10.3, 10.4)**	2.4, 3.2	1.2	0.7	0.8	50594	51751	1230
*	FireSaber-KEM	386 **(17.5, 368.6)**	372.7 **(0.2, 370.6)**	24.7 **(10.6, 10.8)**	3.1, 3.6	1.3	0.6	0.8	50594	51829	1472
*	FrodoKEM-1344-SHAKE	386.2 **(17.5, 368.6)**	388.2 **(17.2, 368.7)**	24.9 **(10.7, 11.1)**	2.8, 3.7	1.2	19.1	19.9	50594	72044	21632
*	HQC-256	387 **(17.5, 368.6)**	370.4 **(0.8, 366.2)**	24.2 **(10.4, 10.3)**	2.8, 3.7	1.1	2.2	3.3	50596	57758	14469
*	ntrulpr857	386.3 **(17.5, 368.6)**	371.9 **(0.6, 365.8)**	24.0 **(10.3, 10.6)**	2.4, 4.2	1.1	1.5	2.0	50594	51698	1312
*	sntrup857	386.9 **(17.5, 368.6)**	376.2 **(8.3, 368.1)**	23.4 **(10.3, 10.5)**	3.5, 4.5	1.6	1.0	2.0	50594	51835	1999
*	SIKE-p751	384 **(17.5, 368.6)**	377.2 **(10.2, 367.9)**	23.9 **(10.6, 10.3)**	2.9, 3.2	1.0	17.3	18.2	50594	51077	596
⑭	Kyber1024-90s	160.3 **(7.1, 152.1)**	154.5 **(0.3, 152.9)**	11.3 **(4.0, 3.9)**	2.9, 3.3	1.3	0.6	0.9	50596	52087	1568
*	NTRU-HPS-4096-821	158.1 **(7.1, 152.1)**	156.6 **(2.5, 151.8)**	11.0 **(4.0, 4.0)**	2.8, 3.4	1.2	0.6	0.8	50594	51751	1230
*	FireSaber-KEM	161.2 **(7.1, 152.1)**	154.1 **(0.2, 151.8)**	11.6 **(4.2, 4.1)**	3.5, 4.2	1.1	0.7	0.8	50596	51507	1472
*	FrodoKEM-1344-SHAKE	160.2 **(7.1, 152.1)**	172.2 **(17.4, 151.6)**	13.2 **(4.0, 4.2)**	2.9, 3.9	0.9	19.1	20.1	50594	72044	21632
*	HQC-256	160.9 **(7.1, 152.1)**	156.7 **(0.9, 151.8)**	12.1 **(3.9, 4.1)**	3.0, 3.9	1.1	2.4	3.3	50594	57756	14469
*	ntrulpr857	159.3 **(7.1, 152.1)**	154.1 **(0.6, 150.9)**	11.1 **(4.0, 4.1)**	2.9, 3.7	1.1	1.5	2.1	50594	51698	1312
*	sntrup857	159.7 **(7.1, 152.1)**	162.2 **(8.4, 151.0)**	11.2 **(3.9, 4.0)**	2.8, 3.7	1.1	1.0	2.1	50596	51837	1999
	SIKE-p751	159.8 **(7.1, 152.1)**	163.1 **(10.2, 151.2)**	11.5 **(4.0, 4.0)**	3.4, 3.8	1.4	17.2	18.4	50594	51077	596