A Post Quantum FHE Blockchain with high throughput

Mike Mu
PQC Labs

January 18, 2024

Abstract

We propose a post-quantum blockchain fortified with lattice-based cryptography. The digital signature algorithm employed ensures security against the mounting threats posed by quantum computing. Its encryption algorithm is harnessed for the construction of decentralized fully homomorphic encryption (FHE) computation across the network. This empowers blockchain nodes to correctly process transactions in their encrypted form, without the knowledge of their plaintext content, the exclusive decryption privileges is reserved solely for individual asset holders, granting them access to their transaction details in plaintext. We also propose an optimized Byzantine Fault Tolerance consensus protocol, showcasing the system's potential to achieve a throughput of 30,000 transactions per second. We also propose a FHE native virtual machine (VM) that is designed to support major FHE operations such as addition, subtraction, comparison, and key switching. This intrinsic feature empowers users to develop arbitrary computation logic, facilitating the execution of computations on encrypted data. This VM not only promotes the composability of transactions but also upholds the fundamental tenets of confidentiality, decentralization, and anti-censorship measures within the blockchain ecosystem. Our approach to compliance is twofold: individual asset holders may undergo potential scrutiny of their transaction history in plaintext, while simultaneously, at the network level, a governing entity preserves an encrypted repository of the overall transaction history. The governing entity possesses the decryption key, enabling the unveiling of transaction details as required. This dual-tiered strategy ensures a nuanced and comprehensive adherence to compliance measures within our system.

Keywords: Post Quantum Blockchain; Lattice Based Cryptography; FHE; Scalable BFT; Privacy; Compliance

Contents

Chapter 1

Cryptography and Quantum Computing

1.1 Quantum computer breaks ECC

Quantum computers pose a serious threat to modern public key cryptography that almost all today's blockchains depend upon, as they can break the hardness of widely-used algorithms such as RSA, Elgamal, and ECC. These three cryptosystems are each based on a different hard problem - large integer factorization, the Discrete Logarithm Problem, and the Elliptical Curve Discrete Logarithm Problem, respectively - but are all susceptible to quantum attacks.

It's worth noting that the threat of quantum attacks on these algorithms is not just theoretical - Table 1.1 shows a large enough quantum computer would take low polynomial time to break RSA and ECC, rendering these widely-used systems insecure. As the development of quantum computers continues to advance, it's becoming increasingly urgent to find new cryptographic methods that can resist quantum attacks and ensure the continued security of cryptography transactions.

Table 1.1: PKI Classical vs Quantum Computing

Problem	Classical Steps	Key Size in bits	Quantum Steps
IFP	$\approx 10^{\sqrt[3]{log(PQ)}}$	3072 to 4096	$\approx O((log(PQ)^2)loglog(PQ)logloglog(PQ))$
DLP	$\approx 10^{\sqrt[3]{log(P)}}$	3072 to 4096	$\approx O((log(P)^2)loglog2(P)logloglog(P))$
ECDLP	$\approx sqrtP$	256 to 384	$\approx O((log(P)^2)loglog(P)logloglog(P))$
CVP	$\approx C^{D_L}$	2000 to 4000	$\approx C^{D_L}$

The exponential rate of progress in developing large quantum computers poses a significant threat to the security of ECC-based cryptocurrencies such as Bitcoin and Ethereum. While the development of large quantum computers is still an engineering challenge, it's uncertain whether these cryptocurrencies will be able to withstand the 20 years of quantum computer development ahead.

1.2 Cryptocurrency May Be Broken Now

Bitcoin, Ethereum, and many of their variants have reduced the length of their public keys (addresses) to 160 bits from the standard 256 bits in ECC, therefore compromise the security level to 80 bits from ECC's standard 128 bit. This makes them vulnerable to attacks by massive classical computers. In fact, this weakness was likely a factor in the successful recovery of bitcoin by the US Department of Justice in the high-profile ransomware attack on Colonial Pipeline in 2021.

1.3 Post Quantum Cryptography

As quantum computing continues to advance at an unprecedented rate, the need for post-quantum cryptography has become increasingly urgent. The security of many widely-used cryptographic systems, such as RSA and ECC, will be severely compromised once a large-scale quantum computer becomes a reality. Post-quantum cryptography offers a solution to this looming threat, using mathematical problems that are believed to be resistant to quantum attacks. As we cannot predict when a large-scale quantum computer will be developed, it is essential that we start preparing now and transition to post-quantum cryptographic systems to ensure the security of our digital infrastructure for years to come.

As history has shown us, even cryptography that was once thought to be unbreakable can be eventually compromised. ECC's security is already at risk due to the rapid development of quantum computing, and its vulnerability to future attacks is not a matter of if, but when. As such, it is important to take proactive measures to prepare for a potential future breach. The process of migrating away from ECC-based systems is not a trivial one, as it involves not only replacing the ECC for transaction authentication, but also migrating many of the other building blocks that rely on ECC or ECC-based bilinear mapping, such as ZKP, ZKVM, and ZKEVM. Therefore, the sooner we begin to prepare for this eventuality, the smoother and less painful the transition will be.

1.4 Lattice Cryptography

Lattice-based cryptography is a highly promising method for achieving quantum-resistant cryptography, with the security of such schemes stemming from the complexity of solving the shortest or closest vector problems in high-dimensional integer lattice systems. This security concept has both theoretical grounding and practical applications, with the National Institute of Standards and Technology (NIST) actively developing standards for post-quantum cryptography, of which lattice-based methods are one of the leading candidates. It is noteworthy that lattice-based cryptography has already demonstrated its potential for security, indicating its practical utility as a viable cryptographic method in the near future.

For this paper, we will use lattice-based cryptography for post-quantum digital signature schemes and to construct zero knowledge transactions. Specifically, we will employ CRYSTAL-DILITHIUM for PQDSA, as it offers a high level of security, speed, and compact signature size. We will use the same scheme to construct zero knowledge transactions, which will allow us to mask the clear text transactions while still being able to prove their correctness. This combination of security, efficiency, and privacy makes lattice-based cryptography a compelling choice for post-quantum cryptographic applications.

1.5 Intro to PQC

Given the NP hardness of SVP, CVP problems for quantum computers, we will use PQC Digital Signature Algorithm (for transaction authentication) to replace the ECDSA or EDDSA, The core idea is to construct a large random integer lattice A of dimension $n \times n$ as part of the public key, a n-dimension vector s as the secret key, and a n-dimension vector e as the noise term, we use them to sign a message that everybody can verify but nobody can forge the signature.

1.5.1 Lattice DSA

We present a high level overview how PQC DSA works.

1. Parameter Selection Fix a lattice dimension parameter N and normal bound k

2. Secret Lattice Creation Alice choose a small polynomial $f \in R[1]$ that determines her secret lattice

3. Random Polynomial Selection Alice chooses a random polynomial $y \in R[k]$

4. Hash Function A hash function is applied to certain quantities associated to f and y. The output from the hash function is a polynomial $c \in R[1]$ that depends randomly on the inputs.

5. Signature Creation Alice computes the polynomial

$$s = f * c + y \ \ in\ the\ ring\ R \tag{1.1}$$

6. Rejection Sampling: If

$$||s||_\infty \geq k - N \tag{1.2}$$

then Alice goes back to step 3 and selects a new value for y.

7. Publication Alice publishes the pair of polynomials (s, c) as her signatures. (s_1, c_1), (s_2, c_2), (s_3, c_3),... Now this transcript reveals no information about Alice's private key f.

1.5.2 Lattice KEM

We present a high level overview how PQC encryption works.

1. Key generation Alice will generate a random matrix A, a secret key s, and a noise vector (error term) e, both s and e have small coefficients, and compute:

$$As + e = t$$

Alice publish (A, t) as her public key, and keep s as her private key.

2. Encryption Bob will use Alice's public key to encrypt a message to Alice: First, Bob generates a random vector r. Next, Bob use the r to multiply A and t (Alice's public key):

$$(r_1, r_2) = r \times A \times t$$

Next, Bob adds two small error terms to r_1, r_2:

$$(e_1 + r_1, e_2 + r_2)$$

Next, Bob add his message m to the 2 terms:

$$(e_1 + r_1 + 0, e_2 + r_2 + m)$$

And send the resulting (u,v) to Alice:

$$(u, v) = (e_1 + r_1 + 0, e_2 + r_2 + m)$$

3. Decryption Alice use her private key s to decrypt (u, v) to obtain the message m:

$$m = v - u \times s - e_t$$

Other parties has no ability to cancel the small error e_t due to the hardness of SVP, CVP problems, even with a quantum computer. We will utilize the KYBER to encrypt the transactions on the blockchain to achieve privacy.

1.5.3 FHE

Fully Homomorphic Encryption (FHE) stands as a pinnacle achievement in the field of cryptography, revolutionizing the landscape by allowing computations on encrypted data without the need for decryption. At its core, FHE is characterized by its ability to support a rich set of algebraic operations while maintaining the confidentiality of the underlying data.

The fundamental principle behind FHE lies in the mathematical framework of lattice-based cryptography. In this context, a lattice serves as a grid-like structure within a mathematical space, and the hardness of certain lattice problems forms the foundation of the encryption's security. FHE schemes typically leverage mathematical structures like ideal lattices, which involve polynomial rings and their quotient rings.

In the context of blockchain, FHE enables the delegation of computations on encrypted data to a blockchain node, which performs the necessary operations without ever decrypting the data. The encryption process involves encapsulating data in a form that allows mathematical operations to be conducted directly on the ciphertext. This includes operations such as addition, multiplication, and other complex computations.

One of the critical components in FHE is the incorporation of noise into the encrypted data. Noise acts as a safeguard against information leakage during computation. Managing and mitigating noise buildup is an ongoing challenge in FHE design, impacting both the efficiency and security of the scheme.

Bootstrapping is a fundamental technique in Fully Homomorphic Encryption (FHE) systems, designed to address a critical challenge known as "noise" accumulation during computation. Noise, in the context of FHE, is the unintended distortion or corruption introduced to the encrypted data as a result of homomorphic operations. Over successive computations, this noise can accumulate and eventually compromise the correctness of the results.

The bootstrapping technique is employed to rejuvenate the ciphertext, effec-

tively reducing the noise to an acceptable level and allowing for continued, error-free computation. The term "bootstrapping" draws an analogy from the concept of pulling oneself up by the bootstraps, as the technique essentially enables the encryption scheme to refresh its own state.

The process involves taking an encrypted ciphertext, which may have accumulated excessive noise, and applying a special homomorphic operation known as the "bootstrap circuit" or "bootstrapping gate." This operation effectively performs a decryption followed by re-encryption homomorphically, reducing the noise while preserving the correctness of the underlying plaintext.

Here is a simplified overview of the bootstrapping process:

Encrypted Data with Noise: Start with an encrypted ciphertext that has undergone multiple homomorphic operations, accumulating noise.

Bootstrap Circuit: Apply the bootstrapping gate or circuit, which performs a homomorphic decryption followed by a homomorphic re-encryption on the ciphertext. This operation reduces the noise while maintaining the encrypted state.

Result: The output is a refreshed ciphertext with reduced noise, ready for further homomorphic operations without compromising the correctness of the computation.

Note that bootstrapping comes with a computational cost. The process is inherently more resource-intensive than standard homomorphic operations, and its frequency depends on the specific FHE scheme and the desired level of security and correctness. Efficient implementation of bootstrapping is a key focus in FHE research, as it directly influences the practicality and performance of FHE in real-world applications. Advances in bootstrapping techniques contribute significantly to the ongoing progress of FHE as a viable solution for secure and privacy-preserving computations.

Moreover, FHE operates in a multi-layered fashion. Initially, a user encrypts their data with a public key, and the ciphertext is then sent to the node. The node performs the computations on the encrypted data, yielding a result that remains encrypted. Finally, the user, holding the corresponding private key, decrypts the result to obtain the final outcome.

While FHE's theoretical foundations are firmly rooted in advanced mathematical concepts, its practical implications are profound. FHE opens avenues for secure and privacy-preserving computations in cloud computing, secure data outsourcing, and collaborative machine learning, among other applications. As cryptographic research continues to advance, FHE remains a focal point, continually evolving to strike an intricate balance between mathematical complexity, practical efficiency, and robust security.

1.6 A Math Primer of Lattice, SVP, CVP

In this section we give a brief introduction to the problem of finding the shortest and closest vector from a lattice, and there is no quantum algorithm to solve it in polynomial time.

1. SVP, CVP - NP Hard Problem for Quantum Computer

   SVP Problem Let $L \in \mathbb{R}^n$ be a lattice, the shortest vector problem is to find a shortest non zero vector in L, that is, to find a vector v_0 in L, satisfying:

   $$||v_0|| = \min_{v \in \mathbf{L}} ||(v)||, v \neq 0$$

   CVP Problem let $L \in \mathbb{R}^n$ be a lattice, the closest vector problem is the problem of finding, for a given target vector $t \in \mathbb{R}^n$, a vector in L that is closest to t, in other words The CVP for L with target vector t is to find a vector v_0 in L, such that:

   $$||v_0 - t|| = \min_{v \in \mathbf{L}} ||v - t||$$

2. Babai's algorithm to solve CVP Input:

   A lattice $L \in \mathbb{R}^n$ A basis

   $$\beta = \left[v_1, ...v_n\right]$$

   A target vector $t \in \mathbb{R}^n$

   Compute: $\alpha_1, ...\alpha_n \in \mathbb{R}$ so that $t = \alpha_1 v_1 + ... + \alpha_n v_n$ $v_0 \leftarrow \lfloor \alpha_1 \rceil v_1 + ... \lfloor \alpha_n \rfloor v_n$

Output: v_0

Babai's algorithm only works reasonably well with a "good" basis - orthogonal basis of vectors $\left[v_1, ...v_n\right]$. For a "bad" basis - "highly non orthogonal basis" it will end up finding a vector outside the lattice. As the dimensions increase, the failure of a bad basis to solve CVP increases exponentially.

3. Gran-Schmidt Algorithm

$$v_1^* \leftarrow v_1, v_k^* \leftarrow v_k - \sum_{i=1}^{k-1} \frac{v_k \cdot v_i^*}{||v_i^*||^2}.v_i^* \tag{1.3}$$

$v_k^* \leftarrow$ projection of v_k on to $Span(v_1, v_2, ..., v_{k-1})^\perp$

Gran-Schmidt algorithm only works well with non-integer lattice, but in our realm the elements are large integers, we must round result to the nearest integers, which will lose "accuracy" in the process, therefore not being able to find the shortest and closest integers.

4. Lattice reduction

Now we introduce Gran-Schmidt Algorithm with Rounding

Size Condition

An ordered basis $v_1, ...v_n$ for L satisfies Size Condition if the output from Gram-Schmidt Algorithm satisfies:

$$\frac{|v_i v_j^*|}{||v_j^*||^2} \leq \frac{1}{2} \; for \; all \; 1 \leq j \leq i \leq n \tag{1.4}$$

Lovasz Condition

An ordered basis $v_1, ..., v_n$ for L satisfies Lovasz Condition, if the output from Gram-Schmidt Algorithm satisfies:

$$(||v_{i+1}^*||)^2 \geq \left(\frac{3}{4} - \frac{(|v_{i+1} \cdot v_i^*|)^2}{||v_i^*||^4}\right) ||v_i^*||^2 \tag{1.5}$$

5. LLL Algorithm

Algorithm 1 LLL Algorithm

1: Input a basis $[v_1, ...v_n]$ for a : lattice L
2: $k \leftarrow 2$
3: **while** $k \leq N$ **do**
4: **for** $j = k-1, k-2, ...1$ **do**
5: $v_1^*, ...v_k^* \leftarrow v_1, ...v_k$ ▷ Gram-Schmidt
6: $\mu_{k,j} \leftarrow (v_k \cdot v_j^*)/||v_j^*||^2$
7: $v_k \leftarrow v_k - \lfloor \mu_{k,j} \rceil v_j$ ▷ Size Reduction
8: **end for**
9: $v_1^*, ...v_k^* \leftarrow v_1, ...v_k$ ▷ Gram-Schmidt
10: $\mu_{k,k-1} \leftarrow (v_k \cdot v_{k-1}^*)/||v_{k-1}^*||^2$
11: **if** $||v_k^*||^2 \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) ||v_{k-1}^*||^2$ **then** ▷ Lovasz Condition
12: $k \leftarrow k-1$
13: **else**
14: Swap v_{k-1} and v_k
15: Set $k \leftarrow \max(k-1, 2)$
16: **end if**
17: **end while**
18: Output: The LLL reduced basis $[v_1, ...v_n]$

6. LLL-BKZ algorithm with block size β

Rather than swapping v_k and v_{k-1} in step 14 of LLL Algorithm 1, we instead take the sub lattice spanned by a block of vectors $v_i, v_{i+1}, v_{i+\beta-1}$ and replace these vectors with a KZ reduced basis for the sub lattice. The LLL-BKZ algorithm terminates in no more than $O(\beta^{\alpha\beta} n^b)$ steps for some small constants a, b and find a non-zero vector $v \in L$, satisfying:

$$||v|| \leq \left(\frac{\beta}{\pi e}\right)^{\frac{n-1}{\beta-1}} \lambda_1(L) \tag{1.6}$$

7. Learn with errors

Most lattice-based cryptographic algorithms rely on hiding one or more small vectors in a lattice. What this means is that the lattice L has a non-zero vector v whose length $||v||$ is significantly smaller than the shortest non-zero length predicted by the Gaussian heuristic.

$$\lambda_1(L) \coloneqq min_{0 \neq v \in L}(||v||) \tag{1.7}$$

$$\gamma(L) \coloneqq \sqrt{\frac{n}{2\pi e}} \cdot \left(Det(L)^{\frac{1}{n}}\right) \; wheren \; \lambda_1(L) \leq n^{\frac{-1}{2}}\gamma(L) \tag{1.8}$$

If $\lambda_1(L)$ is polynomially smaller than $\gamma(L)$, then it takes exponential time to solve SVP or CVP.

The CVP is known to be NP-hard, and the SVP is NP-hard under a randomized reduction hypothesis, combined with "learn with error", they form NP hard problem for quantum computers. The best lattice reduction methods LLL-BKZ requires exponential time to the dimension N for quantum computers to find very short or close vectors with the noise of small errors. This makes SVP and CVP good candidates for post quantum cryptography.

Chapter 2

High Throughput Blockchain

2.1 Blockchain is Slow, Low Throughput

The current state of the blockchain exhibits slow speed and low throughput.

The penalty of decentralization: Decentralization refers to the distribution of decision-making power among nodes in a network, which is a key feature of blockchain and other decentralized systems. However, this can come at a cost in terms of efficiency and speed. In a centralized system, decisions can be made quickly and efficiently by a single entity, but in a decentralized system, decision-making must be distributed among nodes, which can lead to slower decision-making and increased complexity.

Fault tolerance with malicious participants: One of the key benefits of a decentralized system is its ability to resist attacks by malicious participants, but this comes at the cost of increased complexity and slower decision-making. In a decentralized system, multiple nodes must agree on a decision, which can take longer and require more communication than in a centralized system.

POS is constrained to be below 5,000 transaction per second (TPS), often between 1,000-2,000 TPS in practice. Nodes need to communicate for reaching consensus: In a decentralized system, nodes must communicate with each other in order to reach consensus on a decision. This can lead to delays and increased network traffic, especially when there are many nodes involved in the network.

One leader at a time to propose a block, slow and bandwidth consumption: Most blockchain systems use a leader-based consensus algorithm, where on each round, a single node is responsible for proposing a new block. This can be slow and inefficient, especially when there are many nodes in the network, as the leader node must communicate with all other nodes in order to reach consensus.

None leader nodes need 2 rounds of communications: prepare and commit, to achieve consensus: In a leader-based consensus algorithm, non-leader nodes must communicate with each other in order to reach consensus on a decision. This can require at least two rounds of communication, which can be slow and inefficient, especially when there are many nodes in the network.

2.2 High Speed, High Throughput Blockchain

We strive to develop a high speed, high throughput blockchain by 2 major techniques.

One technique is to introduce parallel processing, where multiple leaders can propose multiple new blocks in parallel.

For example, in a sharded blockchain system, different shards can have their own leaders who propose new blocks simultaneously. This increases the overall TPS of the system as a whole. Additionally, using consensus algorithms that allow for parallel processing, such as MIR-BFT that we will introduce below, will also significantly increase the throughput of the system.

Another way to improve the efficiency of consensus in decentralized systems is by using mathematical proofs of correctness for transactions. This involves using mathematical algorithms and protocols, such as zero-knowledge proofs or zk-SNARKs, to prove that a transaction is valid without having to re-execute it.

With this approach, non-leader nodes don't need to re-execute each transaction, but instead they just verify the proof and update the state of the chain. This reduces the overall computational load on the system and speeds up the consensus process, and increases the transaction privacy and security.

2.2.1 Network Assumptions

MIR BFT is the work of IBM researchers in Europe. In a nutshell, MIR BFT is a parallel Byzantine Fault Tolerance protocol. It works under such consumptions:

1. Byzantine Fault Assumption: - N nodes, where F nodes may be Byzantine Faulty (malicious) - Any number of clients can be Byzantine Faulty

2. Asynchronous networks: - Messages may be dropped, delayed, re-ordered; - The network will reach synchronous state for a while, after some time;

2.2.2 MIR BFT Consensus Protocol

1. In a classical BFT protocol, a block is made in 3 steps: Step 1, a leader node propose a block by sending it to all other nodes; Step 2, each node who receive the proposal will communicate with other nodes, then pre-commit the block; Step 3, each node will check one more round with other nodes on the pre-committed block, and commit it if less than 1/3 of the nodes have divergence. The most time consuming and bandwidth consuming operation is on step 1, a leader can only propose a block at a time, wait for its finality, then the next leader can propose another block, wait for its finality. As of step 2 and 3, because each node is at the load balance state, their communication cost is low to moderate, and linear to the size of the transactions in the block.

2. MIR BFT eliminate the bottleneck by enabling multiple leaders to propose their respective blocks in parallel, imagine at time t, we have: node 1 propose block 1 containing unique set of transactions; node 2 propose block 2 containing unique set of transactions; node 3 propose block 3 ...; node 4 propose block 4 ...; At time t_1: node 1 propose block 5; node 2 propose block 6; node 3 propose block 7; node 4 propose block 8; ...

   In mathematical term, for block number n, node i should produce it if:

   $$(n - i)\ mod\ (k \cdot i) = 0$$

   where k is an integer constant that determines the interval between blocks produced by each node. In the above example, k = 4, so node 1 produces blocks 1, 5, 9, 13, node 2 produces blocks 2, 6, 10, 14, and node 3 produces blocks 3, 7, 11, 15.

3. Prevent duplicate transactions (or double spending). To prevent the hack that the client can send multiple copies of the same transactions to different nodes for them to include in their respective proposed blocks, MIR BFT constructs a transaction sharding scheme that hash each transaction into a bucket, a particular node will be only assigned to pick a particular bucket number into a block. This way, in a certain time epoch, all duplicate transactions will always be hashed into one bucket, say bucket

number 1, that can be only picked by node 1, when node 1 sees multiple identical transactions it will only include one unique transaction into the proposed block.

4. To prevent message censorship, for example, node 1 doesn't like a particular transaction or client so it will always drop it, MIR BFT rotate the bucket assignment periodically, at time epoch t, bucket 1, 5, 9, 13 is assigned to node 1, at epoch t+1, bucket 1, 5, 9, 13 will be assigned to a different node, such as node 3, this way each nodes receives transactions from nearly randomized clients, and it has no way to censor any particular client.

5. MIR BFT further suggests two more optimizations to increase the transaction throughput. First is the Light Total Order Broadcast that only broadcast the transaction hash - the transaction digest to ALL correct nodes, and only broadcast the full transaction payload to ONE correct node, to cut the communication cost. Second is called Signature Verification Sharding, to let only one correct node verify the transaction signature as opposed all nodes verifying.

6. MIR BFT is a generalization of PBFT/Avardvark, it only changes the leader election part of PBFT, so it is easy to verify its correctness. It is also very fault tolerant in the sense that the leader set can grow in stable epoch time, and shrink in recovery epoch time.

MIR BFT researchers claim a transaction throughput of 60,000 Bitcoin type of transactions, where each transaction size is about 500 Bytes, and 30,000 Hyperledger type of transactions where each size is about 3,500 Bytes when running a 100 nodes network on AWS WAN, each with 1 GBPS bandwidth and 32GB RAM, 2.0Ghz CPU. As we employ the PQC primitives our signature size will be comparable with Hyperleger's signature size of 3.5K Bytes, since FHE and its bootstrapping builds up computational complexity, we expect 10,000 TPS as a starting point, and further optimize the transaction data, block data structure, hardware, bandwidth, and potentially adopt ZKP technique to increase the transaction throughput. Eventually we expect to reach a transaction throughput of 30,000 TPS.

Chapter 3

FHE Virtual Machine

3.1 FHEVM

We plan to introduce a virtual machine for the blockchain. A virtual machine is a crucial component for blockchain programmability, as it allows for the execution of smart contracts in a sandboxed environment. Smart contracts are self-executing programs that can be programmed to execute automatically when certain conditions are met. These programs can be used for a variety of purposes, from creating decentralized applications (DApps) to transacting digital assets. Since it is a FHE blockchain, we need a VM that can perform native FHE operations.

In our ambitious endeavour to develop an FHE Virtual Machine (FHE VM), we embark on a unique convergence of fully homomorphic encryption (FHE) and microprocessor architecture. Unlike conventional virtual machines, the FHE VM demands a distinctive re-implementation of fundamental operators such as addition (+), subtraction (-), greater than (¿) and smaller than (¡), and the manipulation of word sizes. This specialized FHE VM is designed to facilitate arbitrary computations on encrypted data, reflecting a novel synergy between cryptographic intricacies and microprocessor-level functionality.

The challenge lies in adapting traditional arithmetic and logic operations to operate homomorphically, considering the inherent constraints and complexities of FHE schemes. Each operation necessitates meticulous transformation and integration to ensure compatibility with the homomorphic nature of the encrypted data. Moreover, the FHE VM's design necessitates an acute understanding of

the underlying lattice-based cryptography and the associated trade-offs in terms of security, computation efficiency, and noise management.

As a pioneering effort at the intersection of FHE research and microprocessor architecture, the FHE VM aims not only to re-implement basic operators but to optimize their performance within the context of homomorphic computation. This venture requires careful consideration of circuit design, gate complexity, and computational overhead. Ultimately, the FHE VM aspires to be a cornerstone in advancing secure and privacy-preserving computations, bridging the realms of cryptography and microprocessor architecture in a symbiotic fusion of innovation.

3.2 FHEVM vs ZKVM

In recent years, Zero-Knowledge Proofs (ZKP) have emerged as a prominent and groundbreaking advancement in cryptography, revolutionizing privacy-preserving computations across various domains. The robustness of ZKP, particularly in the realm of verifiable computation and confidentiality, has garnered considerable attention and utilization. However, amidst this cryptographic landscape, Fully Homomorphic Encryption (FHE) stands as an even more cutting-edge paradigm, albeit in its nascent stages of development. FHE's unique capability to enable computations on encrypted data without decryption presents a transformative potential for secure and privacy-preserving operations. While ZKP has gained popularity for its efficiency and applicability in scenarios requiring knowledge proofs, the distinctive and powerful properties of FHE may well position it to gain increased recognition and utilization in the future. In this discourse, we delve into a comparative analysis of these two cutting-edge cryptographic paradigms, exploring their respective strengths, challenges, and potential implications for the evolving landscape of secure computation.

Comparing the FHE Virtual Machine (FHE VM) and Zero-Knowledge Virtual Machine (ZK VM) entails a nuanced evaluation of their strengths and weaknesses across various dimensions. Notably, the FHE VM confronts challenges in implementation complexity, primarily stemming from the intricate nature of fully homomorphic encryption (FHE) operations. The need for specialized adaptations of basic operators adds to the complexity, and it's worth highlighting that FHE VM, being a relatively novel concept, lacks prior work for reference, necessitating groundbreaking efforts. In contrast, the ZK VM landscape has witnessed considerable progress, with several established frameworks available. While ZK VM introduces complexity through sophisticated zero-knowledge proof protocols, the existing body of work provides a foundation that can be leveraged, streamlining development efforts and referencing established methodologies. The distinction lies not only in the nature of complexity but also in the availability of established frameworks within the respective domains.

Computation complexity diverges between the FHE Virtual Machine (FHE VM) and the Zero-Knowledge Virtual Machine (ZK VM), presenting a nuanced perspective. FHE VM grapples with intricacies in design and implementation, engaging in matrix vector multiplication as its primary computational task. Despite historically expensive bootstrapping operations, recent advancements, particularly with Torus FHE, have significantly enhanced the efficiency of this process. In contrast, ZK VM, initially perceived as having a lower computational burden when focusing on proving knowledge, has witnessed a shift. Recent ZK proof constructions, like Groth16 and Plonk, extend beyond mere knowledge proofs to verify the correctness of arbitrary computations. This involves the evaluation of high-degree polynomials with large scalar coefficients and circuit satisfiability, resulting in prolonged proof generation times and substantial computational resource requirements. While FHE VM encounters design complexities, its computational demands appear to be more favorable, potentially one or two orders of magnitude less than the evolving demands of ZK VM, particularly with recent advancements in homomorphic encryption techniques.

The speed of operations between the FHE Virtual Machine (FHE VM) and the Zero-Knowledge Virtual Machine (ZK VM) reflects a nuanced interplay of computational demands. Contrary to the conventional perspective, FHE VM showcases noteworthy speed advantages attributed to its foundational operations, particularly matrix vector multiplication and efficient low modulus computation. In contrast, ZK VM, especially with recent zero-knowledge proof protocols like Groth16, Plonk, or Stark, encounters substantial computational overhead. These protocols involve the computation of high-degree polynomials with large coefficients, often resulting in prolonged proof generation times, measured in minutes if not hours. The conventional notion that FHE VM lags in computational speed, especially with larger and more complex computations, undergoes reconsideration when accounting for the nature of operations involved in both cryptographic paradigms.

In terms of ease of programmability, the FHE Virtual Machine (FHE VM) holds a notable advantage over its counterpart, the Zero-Knowledge Virtual Machine (ZK VM). FHE VM, despite its demand for a specialized understanding of FHE-specific operators, provides an intuitive programming experience akin to standard computations, even with the encryption of data. This inherent simplicity arises from the fact that, unlike ZK VM, FHE operations do not require developers to navigate intricate processes such as witness computation, prover circuit design, or the management of a trusted setup. While both VMs present challenges, FHE VM's user-friendly approach enhances its appeal, offering a more straightforward and accessible programming paradigm.

The size of computation payload may favor ZK VM, as zero-knowledge proofs can succinctly represent the validity of a computation without revealing its

details. FHE VM, dealing with encrypted data, may incur a larger payload due to the necessity of carrying additional encryption metadata.

Integratability with current VMs is a nuanced aspect. Adapting existing VMs, like LLVM to ZK VM or WASM to ZKWASM, aligns with the modular nature of zero-knowledge proof systems. FHE VM integration, however, poses a more significant challenge due to the unique requirements of homomorphic encryption and potential impacts on existing execution models.

In conclusion, the choice between FHE VM and ZK VM depends on the specific use case, balancing considerations of computational complexity, speed, ease of programmability, payload size, and integratability. Both VMs represent cutting-edge advancements in privacy-preserving computation, each with its own set of trade-offs and applications within the evolving landscape of secure computation.

3.3 FHEVM Design

A fully homomorphic encryption (FHE) virtual machine is a virtual machine that is optimized for performing FHE operations such as addition, subtraction, multiplication, division, modulus, equality, inequality etc.

As the time of this paper, there isn't any recognized and established VM designed specifically to execute FHE operations. It is on us to perform this groundbreaking research and development.

A Fully Homomorphic Encryption (FHE) virtual machine represents a groundbreaking endeavor tailored for the seamless execution of FHE operations, encompassing fundamental arithmetic computations like addition, subtraction, multiplication, division, modulus operations, as well as more complex logical operations such as equality and inequality checks.

The design and development of an FHE virtual machine entail confronting several intricate challenges that underscore the pioneering nature of this research initiative. Firstly, FHE operations involve intricate algebraic computations on encrypted data, introducing challenges related to computational intensity and efficiency. Given the nature of homomorphic encryption, achieving optimal performance in terms of execution speed and resource utilization becomes a delicate balancing act, necessitating innovative algorithms and optimizations.

Furthermore, the lack of a recognized and established VM specifically tailored for FHE operations at the time of this paper introduces a considerable void in

the existing cryptographic landscape. As such, the onus falls upon us to pioneer this area, addressing the dearth of dedicated virtual machines optimized for FHE computations. This requires the development of novel algorithms and methodologies that can efficiently perform homomorphic operations while mitigating computational overhead.

One of the significant challenges lies in the selection of appropriate cryptographic primitives and protocols that can seamlessly integrate with the virtual machine architecture. The need for a well-designed interface between the FHE virtual machine and the underlying cryptographic mechanisms poses a challenge, demanding a nuanced understanding of cryptographic protocols and their interactions with virtual machine architectures.

In terms of existing frameworks, the landscape is relatively uncharted as of now. Leveraging existing virtual machine frameworks for general-purpose computations might not be directly applicable to the specialized requirements of FHE operations. This underscores the necessity for a dedicated framework that aligns with the unique characteristics and computational demands of fully homomorphic encryption.

In summary, the development of an FHE virtual machine necessitates overcoming challenges related to computational efficiency, algorithmic innovations, cryptographic protocol integration, and the absence of established frameworks. This pioneering initiative requires groundbreaking research to advance the capabilities of FHE computations within a virtualized environment, contributing to the evolution of privacy-preserving technologies and expanding the possibilities of secure computation on encrypted data.

Chapter 4

Transaction Privacy

4.1 The Problem of Pseudo Anonymity

In the predominant paradigm of blockchain systems, exemplified by notable instances such as Bitcoin and Ethereum, transactional details are meticulously documented in an open ledger accessible to all participants within the network. This transparency extends to the revelation of sender and receiver addresses, transaction amounts, and timestamps, forming a comprehensive and publicly available record inscribed indelibly in the blockchain.

However, this ostensible transparency belies a critical caveat: the purported pseudonymity of transactions does not confer genuine anonymity. Despite the absence of direct links to real-world identities, the meticulous observer armed with the capacity to correlate a Bitcoin or Ethereum address to an actual individual—attainable through exchanges or publicly available data—can potentially unravel the entirety of transactions affiliated with said address. This inherent lack of privacy in blockchain systems becomes a salient concern, given the potential ramifications for users, encompassing the compromise of financial privacy, augmented susceptibility to targeted malfeasance or scams, and other latent risks.

The burgeoning discourse on privacy within the blockchain space has propelled the exploration of sophisticated solutions, among which Fully Homomorphic Encryption (FHE) emerges as a beacon of promise. Unlike traditional blockchain setups, where transactional details are unveiled for public scrutiny, FHE introduces an innovative paradigm by enabling computations directly on encrypted

data. This transformative approach holds the potential to obfuscate transactional intricacies, shielding users from the prying eyes that seek to unveil patterns of behavior or derive additional insights into financial activities. The realm of privacy-preserving technologies, marked by the ascent of FHE, zero-knowledge proofs and cryptographic innovations, is undergoing a renaissance—a concerted effort to fortify blockchain ecosystems with the robust cloak of privacy, safeguarding users against the perils of unwarranted exposure and mitigating the risks associated with the transparency paradox.

4.2 Compliance

Architecting a privacy-centric blockchain that aligns with regulatory mandates poses a formidable challenge. Striking a delicate equilibrium between user privacy and regulatory access to transactional data becomes imperative, as users seek anonymity and security while regulators necessitate transactional insights for law enforcement and fiscal oversight.

Our compliance solution introduces an innovative approach through the infusion of selective transparency into the blockchain's fabric. This method orchestrates a nuanced balance, affording specific entities access to designated transactional information while safeguarding the confidentiality of other data. In practice, individual account holders willingly provide transactional details. Subsequently, leveraging the advanced features of Fully Homomorphic Encryption (FHE), we employ the key switching capability to re-encrypt the entire transaction history. This re-encryption process creates a distinct ledger, accessible exclusively to a dedicated governance entity holding the decryption key.

Crucially, both the encryption and decryption keys employed in this process are threshold keys derived from a threshold cryptography setup, specifically employing T of N keys from the members constituting the governance entity. This multi-key arrangement enhances security and decentralization, ensuring that no single entity holds absolute control over the decryption process.

In essence, this compliance-centric design champions a meticulous harmony between user privacy and regulatory imperatives. By incorporating a suite of features, notably leveraging FHE's advanced cryptographic capabilities, the system not only assures user privacy but also enables regulators to monitor specific transactions, thus fostering a blockchain ecosystem that is both compliant and privacy-preserving.

4.3 Our Approach to Privacy

Our approach to achieving robust privacy within the blockchain ecosystem is anchored in an advanced system design harnessing cutting-edge cryptographic techniques, primarily leveraging Fully Homomorphic Encryption (FHE). To establish a secure and privacy-preserving foundation, each network participant will first engage in a Distributed Key Generation (DKG) process. This collaborative endeavor yields a network encryption and decryption key, collectively utilized for encrypting transactions before their placement on the blockchain.

Notably, the decentralization of key management is a pivotal aspect of our design. Each participant exclusively holds a partial decryption key, ensuring that no individual or coalition of participants less than a predefined threshold (T) possesses the ability to decrypt transactions. Setting T out of N as a ratio, representing a significant majority of network participants, for instance, 75%, signifies that only a collaboration exceeding this threshold can collectively decrypt the transaction history.

Acknowledging the operational challenges associated with garnering an overwhelming majority for routine decryption tasks, particularly in response to individual regulator or law enforcement requests, we introduce a separate governance entity. This entity, comprising fewer members, holds a distinct group key that encrypts the entire transaction history. When faced with a regulatory request, the governance entity members employ a voting mechanism to authorize or deny the revelation anonymously. Upon achieving a quorum, defined by a predefined agreement threshold, the group members collaboratively decrypt the specified transactions and provide the necessary details to the relevant authorities.

Crucially, on-chain storage of encrypted transactions, inclusive of account balances and transaction amounts, becomes an integral part of our privacy-centric design. The encrypted data is then re-encrypted by each network node using individual user keys. This additional layer ensures that users can decrypt the transaction data on their end, revealing their account details, including transaction history, in plaintext.

In essence, this comprehensive system design, entwining distributed key generation, threshold cryptography, and strategic governance entities, establishes a formidable privacy infrastructure within the blockchain. By integrating FHE's capabilities, our approach not only fortifies user privacy but also orchestrates a dynamic balance, empowering the network to respond effectively to regulatory and law enforcement requirements without compromising the fundamental tenets of individual confidentiality and blockchain transparency.

Chapter 5

Blockchain Design Considerations

5.1 Block Reward

Block reward is the incentive given to miners for successfully adding a new block to the blockchain. It will be a combination of newly created cryptocurrency and transaction fees.

The block reward is a crucial aspect of the blockchain's monetary policy, as it directly affects the inflation rate and the total supply of the cryptocurrency. A higher block reward incentivizes more miners to participate in the network, increasing the security and decentralization of the blockchain. However, it also leads to higher inflation and a larger supply of the cryptocurrency, potentially reducing its value. We will set the block reward to decrease over time in a predetermined manner.

The specific rate of decay for the block reward depends on various factors, including the desired inflation rate, the total supply of the cryptocurrency, and the network's security needs. Details will be released in our engineering specifications in a separate document.

5.2 Transaction fee

Transaction fees serve several purposes. First and foremost, they incentivize miners or validators to include transactions in a block. Without transaction fees, there would be little motivation for miners to prioritize adding transactions

to the blockchain.

Secondly, transaction fees are used to mitigate spam attacks. If there were no fees, malicious actors could flood the network with an overwhelming number of transactions, potentially causing congestion and network issues.

Finally, transaction fees can also be used to provide a source of revenue for the network's developers or to fund other initiatives related to the network's development and maintenance.

The transaction fee will be set to balance the need for incentivizing miners to include transactions in a block and the desire to keep transaction fees low for users. We plan to set the initial transaction fee in the range of 0.1% to 0.3% of per transaction amount, and review and possibly adjust it depending on factors such as the size of the network, the demand for block space, and the cost of maintaining the network.

5.3 Storage model

At our blockchain project, we understand the importance of having a storage model that balances security, speed, throughput, and zero-knowledge friendliness. We are currently exploring several storage model candidates, including Merkle Patricia Trie (MPT), Merkle Mountain Range (MMR), Radix Tree, Binary Merkle Tree, Recursive Length Prefix, and others. Each model has its unique strengths and weaknesses, and we are carefully evaluating them to determine which one can best meet our project's requirements. Our goal is to choose a storage model that can provide high speed and throughput while maintaining a high level of security and allowing for efficient zero-knowledge proofs. We believe that by considering all these factors, we can create a blockchain platform that is truly innovative and efficient.

Based on the criteria of security, speed, throughput, and zero-knowledge friendliness, the Merkle Patricia Trie (MPT) and the Merkle Mountain Range (MMR) stand out as promising storage models. MPT is a tried-and-true storage model used by Ethereum and offers excellent security and zero-knowledge friendliness, while MMR offers a novel approach to storing data that allows for efficient proofs of inclusion and exclusion, making it a strong candidate for high-speed, high-throughput applications. Both storage models have been extensively researched and implemented in existing blockchains, making them reliable choices for our consideration.

Chapter 6

ecosystem

1. Wallet We plan to develop a browser extension wallet

2. Smart Contract Development Tool We plan to develop a framework to write, test, compile and deploy smart contracts,

3. Asset creation and trading We plan to program native smart contracts for users to invoke to create assets, tokens and to swap them on a on-chain exchange.

4. Group signature We plan to develop a native group signature, threshold signing ability to enable asset management with multi-signature transactions.

5. dAPP We plan to invite/work with external teams to develop various apps to enrich our ecosystem.

In summary, PQChain is a quantum-resistant blockchain system developed with a focus on privacy, compliance, programmability, and speed. Employing lattice-based cryptography, PQChain not only ensures resistance against quantum threats but also integrates Fully Homomorphic Encryption (FHE) for enhanced privacy and regulatory alignment. Emphasizing programmability, the system aims to cultivate a diverse community and support various applications. Additionally, PQChain addresses transactional bottlenecks by prioritizing speed,

offering users a scalable and efficient blockchain experience. This research-driven approach positions PQChain as a significant advancement in blockchain technology, with the potential to attract a substantial user base based on its innovative features and future-proof design.

Chapter 7

Mathematics and Algorithms

In this chapter, we delve into the mathematical underpinnings of lattice-based post-quantum cryptography (PQC) and lattice-based zero-knowledge proofs. The content is intended for readers seeking a deeper understanding of the mathematics behind these concepts and to verify their validity.

7.1 Lattice based DSA, Dilithium

Below is a general implementation of lattice based digital signature algorithm

1. Parameter Selection
   Fix a lattice dimension parameter N and normal bound k

2. Secret Lattice Creation
   Alice choose a small polynomial $f \in R[1]$ that determines her secret lattice

3. Random Polynomial Selection
   Alice chooses a random polynomial $y \in R[k]$

4. Hash Function
   A hash function is applied to certain quantities associated to f and y. The output from the hash function is a polynomial $c \in R[1]$ that depends randomly on the inputs.

5. Signature Creation
   Alice computes the polynomial

$$s = f * c + y \ \ in\ the\ ring\ R \tag{7.1}$$

6. Rejection Sampling: If

$$||s||_\infty \geq k - N \tag{7.2}$$

   then Alice goes back to step 3 and selects a new value for y.

7. Publication
   Alice publishes the pair of polynomials (s, c) as her signatures.
   (s_1, c_1), (s_2, c_2), (s_3, c_3),...

   Now this transcript reveals no information about Alice's private key f.

Below is a simplified and less efficient design of the scheme that is based on the "Fiat-Shamir with Aborts" approach.

KeyGen

Algorithm 2 Simplified Lattice-based DSA

KeyGen

1: $A \leftarrow R_q^{k \times l}$
2: $(s_1, s_2) \leftarrow S_n^l \times S_n^k$
3: $t :- As_1 + s_2$
4: return $(pk = (A, t))$, $(sk = (A, t, s_1, s_2))$

Sign(sk, M)

5: $z \coloneqq \perp$
6: **while** $z \coloneqq \perp$ **do**
7: $\quad y \leftarrow S_{\gamma 1 - 1}^l$
8: $\quad w_1 \coloneqq \text{HighBits}(Ay, 2\gamma_2)$
9: $\quad c \in B_t \coloneqq H(M||w_1)$
10: $\quad z \coloneqq y + cs_1$
11: $\quad$ **if** $||z|| \geq \gamma_1 - \beta$ or $||LowBits(Ay - cs_2, 2\gamma 2)|| - \infty \geq \gamma_2 - \beta$ **then**
12: $\quad\quad z \coloneqq \perp$
13: $\quad$ **end if**
14: **end while**
15: return $\sigma = (z, c)$

Verify(pk, M, $\sigma = (z, c)$)

16: $w_1^{'} \coloneqq HighBits(Az - ct, 2\gamma_2)$
17: return $[||z||_\infty \leq \gamma_1 - \beta]$ and $[c = H(M||w_1^{'})]$

We set $q = 2^{23} - 2^{13} + 1$ and $n = 256$, All algebraic operations are over the polynomial ring R_q. First it generates a $k \times l$ matrix A each of whose entries is a polynomial in the ring $R_q = \mathbb{Z}[X]/(X^n + 1)$. Then the algorithm samples random secret key vectors s_1 and s_2, each coefficient is an element of R_q with small coefficients of size at most η. The second part of the public key is derived as $t = As_1 + s_2$

Dilithium made several optimizations to the above DSA scheme.

First, it reduced the public key size of $k \times l$ matrix of polynomials by having A generated from some seed ρ using SHAKE-128, the public key is therefore (ρ, t) and its size is dominated by t. Dilithium further shrinks the bit-representation size of t by a factor if two at the expense of increasing the signature by 100 bytes.

Also, Dilithium adds a seed to the secret key and use this seed together with the message to produce the randomness y in Line 03 of signing algorithm.

The rest is parameter selections to make digital signing fast and secure: how to choose matrix A and its polynomial rings $\mathbb{Z}_q[X]/X^{256}+1$ so that matrix multiplication can be efficient via Number Theoretic Transform (NTT), which is a variant of FFT over finite field $\mathbb{Z}_q$ rather than over complex numbers; how to choose a prime q so that the group $\mathbb{Z}_q^*$ has an element of order $2n = 512$, or equivalently $q \cong 1 \pmod{512}$; how to choose an efficient SHAKE-128 implementation so that time-consuming operation of expanding a seed ρ into the polynomial matrix A, which is needed for both signing and verification.

We will omit the detailed discussions of choosing parameters in this paper, rather, we just list 2 set of candidates now and implement one set later.

Table 7.1: Candidates of Dilithium Parameters

NIST Security Level	3	5
q [modulus]	$2^{23}-2^{13}+1$	$2^{23}-2^{13}+1$
d [dropped bits from t]	13	13
$\tau[\#\, of \pm 1's\, in\, c]$	49	60
Challenge entropy $\left[log\left(\binom{256}{\tau}\right)\right]$	225	257
γ_1[y coefficient range]	2^{19}	2^{19}
γ_2[low order rounding range]	$(q-1)/32$	$(q-1)/32$
(k,l)[dimension of A]	$(6,5)$	$(8,7)$
η[secret key range]	4	2
β $[\tau \cdot \eta]$	196	120
ω [max. # of 1's in hint h]	55	75
Repetitions	5.1	3.85

The Key Generation, Signing, and Verification algorithms for Dilithium are presented below. We present the deterministic version of the scheme in which the randomness used in the signing procedure is generated (using SHAKE-256) as a deterministic function of the message and a small secret key. Since our signing procedure may need to be repeated several times until a signature is produced, we also append a counter in order to make the SHAKE-256 output differ with each signing attempt of the same message. Also due to the fact that each (possibly long) message may require several iterations to sign, we compute an initial digest of the message using a collision-resistant hash function,and use this digest in place of the message throughout the signing procedure.

Algorithm 3 Full Dilithium DSA Algorithm

KeyGen

$\zeta \leftarrow \{0,1\}^{256}$
$(\rho, \varsigma, K) \in \{0,1\}^{256\times 3} \coloneqq H(\zeta)$ ▷ H is SHAKE-256
$(s_1, s_2) \in S_\eta^l \times S_\eta^k \coloneqq H(\varsigma)$
$\mathbf{A} \in R_q^{k\times l} \coloneqq \mathbf{ExpandA}(\rho)$ ▷ A is generated in NTT representation as $\hat{\mathbf{A}}$
$t \coloneqq \mathbf{A}s_1 + s_2$ ▷ Compute $\mathbf{A}s_1$ as NTT^{-1} $(\hat{\mathbf{A}} \cdot NTT(s_1))$
$(t_0, t_1) \coloneqq Power2Round_q(t, d)$
$tr \in \{0,1\}^{384} \coloneqq CRH(\rho||t_1)$
return $(pk = (\rho, t_1),\ sk = (\rho, K, tr, s_1, s_2, t_0))$

Sign(sk, M)

$\mathbf{A} \in R_q^{k\times l} \coloneqq \mathbf{ExpandA}(\rho)$ ▷ A is generated in NTT representation as $\hat{A}$
$\mu \in \{0,1\}^{384} \coloneqq CRH(tr||M)$
$\kappa \coloneqq 0,\ (z, h) \coloneqq \perp$
$\rho^{'} \in \{0,1\}^{384} \coloneqq CRH(K||\mu)$ or $\rho^{'} \leftarrow \{0,1\}^{384}$ for randomized signing
while (z, h) $= \perp$ **do** ▷ Pre-compute $\hat{s_1} \coloneqq NTT(s_1)$, $\hat{s_2} \coloneqq NTT(s_2)$, $\hat{s_0} \coloneqq NTT(s_0)$
  $y \in \tilde{S}_{\gamma_1}^l \coloneqq \text{ExpandMark}(\rho^{'}, \kappa)$
  w $\coloneqq$ Ay ▷ w $\coloneqq NTT^{-1}(\hat{A} \cdot NTT(y))$
  $w_1 \coloneqq HighBits_q(w, 2\gamma_2)$
  $\tilde{c} \in \{0,1\}^{256} \coloneqq \text{H}(\mu||w_1)$
  $c \in B_\tau \coloneqq \text{SampleInBall}(\tilde{c})$ ▷ Store c in NTT representation as $\hat{c} = NTT(c)$
  $z \coloneqq y + cs_1$ ▷ Compute cs_1 as $NTT^{-1}(\hat{c} \cdot \hat{s_1})$
  $r_0 \coloneqq \text{LowBits}_q(w - cs_2, 2\gamma_2)$ ▷ Compute cs_2 as $\text{NTT}^{-1}(\hat{c} \cdot \hat{s_2})$
  if $||z||_\infty \geq \gamma_1 - \beta$ or $||r_0||_\infty \geq \gamma_2 - \beta$ **then**
    (z, h) $\coloneqq \perp$
  else
    h $\coloneqq \text{MakeHint}_q(-ct_0, w - cs_2 + ct_0, 2\gamma_2)$ ▷ Compute ct_0 as $\text{NTT}^{-1}(\hat{c} \cdot \hat{t_0})$
    if $||ct_0||_\infty \geq \gamma_2$ or (# of 1's in h $\geq \omega$) **then**
      $(z, h) \coloneqq \perp$
    end if
  end if
  $\kappa \coloneqq \kappa + l$
end while
return $\sigma = (z, h, \tilde{c})$

Verify(pk, M, σ) **= (z, h,** $\hat{c}$**)**

A $\in R_q^{k\times l} \coloneqq \text{ExpandA}(\rho)$ ▷ A is generated in NTT representation as $\hat{A}$
$\mu \in \{0,1\}^{384} \coloneqq \text{CRH}(\text{CRH}(\rho||t_1||M))$
$c \coloneqq \text{SampleInBall}(\hat{c})$
$w_1^{'} \coloneqq \text{UseHint}_q(\text{h, Az - ct}_1 \cdot 2^d, 2\gamma_2)$ ▷ Compute as $\text{NTT}^{-1}(\hat{A} \cdot NTT(z) - NTT(c) \cdot NTT(t_1 \cdot 2^d))$
return $[||z||_\infty < \gamma_1 - \beta]$ and $[\tilde{c} = H(\mu||w_1^{'})]$ and [# of 1's in h $\leq \omega$]

Above is the pseudo-code for deterministic and randomized versions of Dilithium. The only difference between the two versions is in Line 12, where $\rho^{'}$ is either a function of the key and message, or is chosen completely at random.

7.1.1 Security Analysis

The best attacks involve finding short vectors in some lattice, which takes exponential steps for quantum and classical computer. The main difference between the MLWE and MSIS problems is that the MLWE problem involves finding a short vector in a lattice in which an "unusually short" vector exists. The MSIS problem, on the other hand, involves just finding a short vector in a random lattice. In knapsack terminology, the MLWE problem is a low-density knapsack, while MSIS is a high-density knapsack instance. While the MLWE and MSIS problems are defined over polynomial rings, we do not currently have any way of exploiting this ring structure, and therefore the best attacks are mounted by simply viewing these problems as LWE and SIS problems with the ring R_q being replaced by $\mathbb{Z}_q$.

7.2 Lattice based KEM, Kyber

7.2.1 Module LWE

The construct of module learn with error is to reduce the size of matrix A and matrix multiplication of LWE (Learn With Error) with the help of NTT (Number Theory Transform), and add k recursion to RLWE (Ring Learn With Error) to get the optimized performance and security, whose level can be tuned with the selection of parameter k.

M-LWE Parameters: $q \geq 0$ a modulus, $m, k > 0$ be integers, χ a distribution over $\mathbb{Z}[X]/X^n + 1$.

Input: m samples of the form $a_i, < a_i, s > +e_i$, where: $a_i \leftarrow (\mathbb{Z}[X]/X^n + 1)^k$, $e_i \leftarrow \chi$

Challenge: find the secret key $s \in (\mathbb{Z}[X]/X^n + 1)^k$

The storage size of M-LWE is $\mathcal{O}(k^2 n)$

The computation complexity is $\mathcal{O}(k^2 n \log n)$

Below is the visualization of LWE, R-LWE, M-LWE.

Learn With Error as below, with storage size of $\mathcal{O}(n^2)$ and computation complexity of $\mathcal{O}(n^2 n)$

$$\begin{bmatrix} a_{00} & a_{01} & \dots & a_{0n} \\ a_{10} & a_{11} & \dots & a_{1n} \\ \vdots & & & \vdots \\ a_{n0} & a_{n1} & \dots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_n \end{bmatrix} + \begin{bmatrix} e_0 \\ e_1 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_n \end{bmatrix}$$

Ring Learn With Error as below, with storage size of $\mathcal{O}(n)$ and computation complexity of $\mathcal{O}(n \log n)$

$$\begin{bmatrix} a_{0,0} & -a_{n,0} & -a_{n-1,0} & \dots & -a_{1,0} \\ a_{1,0} & a_{0,0} & -a_{n,0} & \dots & -a_{2,0} \\ a_{2,0} & a_{1,0} & a_{0,0} & \dots & -a_{3,0} \\ a_{3,0} & a_{2,0} & a_{1,0} & \dots & -a_{4,0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n-1,0} & a_{n-2,0} & a_{n-3,0} & \dots & -a_{n,0} \\ a_{n,0} & a_{n-1,0} & a_{n-2,0} & \dots & a_{0,0} \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} + \begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_{n-1} \\ e_n \end{bmatrix} = \begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_{n-1} \\ t_n \end{bmatrix}$$

Module Learn With Error as below, with storage size of $\mathcal{O}(k^2 n)$ and computation complexity of $\mathcal{O}(k^2 n \log n)$

$$\begin{bmatrix} a_{0,0}(X) & \dots & a_{0,k}(X) & \vdots \\ \vdots & & \vdots & \\ a_{k,0}(X) & \dots & a_{k,k}(X) & \end{bmatrix} \cdot \begin{bmatrix} s_0(X) \\ \vdots \\ s_k(X) \end{bmatrix} + \begin{bmatrix} e_0(X) \\ \vdots \\ e_k(X) \end{bmatrix} = \begin{bmatrix} t_0(X) \\ \vdots \\ t_k(X) \end{bmatrix}$$

7.2.2 Kyber explained

With M-LWE explained, we can have a high level view of how KYBER works.

1. Key generation Alice will generate a random matrix A, a secret key s, and a noise vector (error term) e, both s and e have small coefficients, and compute:

   $$As + e = t$$

   Alice publish (A, t) as her public key, and keep s as her private key.

2. Encryption Bob will use Alice's public key to encrypt a message to Alice:
   First, Bob generates a random vector r.
   Next, Bob use the r to multiply A and t (Alice's public key):

   $$(r_1, r_2) = r \times A \times t$$

   Next, Bob adds two small error terms to r_1, r_2:

   $$(e_1 + r_1, e_2 + r_2)$$

   Next, Bob add his message m to the 2 terms:

   $$(e_1 + r_1 + 0, e_2 + r_2 + m)$$

   And send the resulting (u,v) to Alice:

   $$(u, v) = (e_1 + r_1 + 0, e_2 + r_2 + m)$$

3. Decryption Alice use her private key s to decrypt (u, v) to obtain the message m:

   $$m = v - u \times s - e_t$$

   Other parties has no ability to cancel the small error e_t due to the hardness of SVP, CVP problems, even with a quantum computer.

   We will utilize the KYBER to encrypt the transactions on the blockchain to achieve privacy.

7.2.3 KYBER Full Algorithm

Below is the full KYBER algorithm we will implement:

Algorithm 4 Full KYBER KEM Algorithm

KeyGen

$d \leftarrow B^{32}$
$(\rho, \sigma) \coloneqq G(d)$
$N \coloneqq 0$
for i from 0 to k **do** ▷ Generate Matrix $\hat{A} \in R_q^{k \times k}$ in NTT domain
  for in from 0 to k-1 **do** $\hat{A}_{ij} \coloneqq \text{Parse}(\text{XOF}(p, j, i))$
  end for
end for
for i from 0 to k-1 **do** ▷ Sample $s \in R_q^k$ from B_{η_1}
  $s_i \coloneqq \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$
  N $\coloneqq$ N + 1
end for
for j from 0 to k-1 **do** ▷ Sample $e \in R_q^k$ from B_{η_1}
  $e_i \coloneqq \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$
  N $\coloneqq$ N + 1
end for
$\hat{s} \coloneqq \text{NTT(s)}$
$\hat{e} \coloneqq \text{NTT(e)}$
$\hat{t} = \hat{A} \cdot \hat{s} + \hat{e}$
$pk \coloneqq (\text{Encode}_{12}(\hat{t} \bmod^+ q) || \rho))$
$sk \coloneqq \text{Encode}_{12}(\hat{s} \bmod^+ q)$

Encrypt(pk, m, r)

N $\coloneqq$ 0
$\hat{t} \coloneqq \text{Decode}_{12}(pk)$
$\rho \coloneqq pk + 12 \cdot k \cdot n/8$
for i from 0 to k-1 **do** ▷ Generate matrix $\hat{A} \in R_q^{k \times k}$ in NTT domain
  for j from 0 to k-1 **do** $\hat{A}_{ij} \coloneqq \text{Parse}(\text{XOF}(\rho, j, i))$
  end for
end for
for i from 0 to k-1 **do** ▷ Sample r $\in R_q^k$ from B_{η_1}
  $r_i \coloneqq \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$
  $N \coloneqq N + 1$
end for
for i from 0 to k-1 **do** ▷ Sample $e_1 \in R_q^k$ from B_{η_2}
  $e_{1_i} \coloneqq \text{CBD}_{\eta_2}(\text{PRF}(\sigma, N))$
  $N \coloneqq N + 1$
end for
$e_2 \coloneqq \text{CBD}_{\eta_2}(\text{PRF}(r, N))$ ▷ Sample $e_2 \in R_q^k$ from B_{η_2}
$\hat{r} \coloneqq \text{NTT(r)}$
u $\coloneqq \text{NTT}^{-1}(\hat{A}^T \cdot \hat{r}) + \text{e}_1$ ▷ $u \coloneqq A^T \cdot r + e_1$
$v \coloneqq \text{NTT}^{-1}(\hat{t}^T \cdot \hat{r}) + e_2 + \text{Decompress}_q(\text{Decode}_a(\text{m}), 1)$ ▷ $v \coloneqq t^T \cdot r + e_2 + \text{Decompress}_q(m, 1)$

$c_1 \coloneqq \text{Encode}_{d_u}(\text{Compress(u, d}_u))$
$c_2 \coloneqq \text{Encode}_{d_v}(\text{Compress(v, d}_v))$
return c $= (c_1 || c_2)$ ▷ $c \coloneqq (\text{Compress}(u, d_u), \text{Compress}(v, d_v))$

Decrypt(sk, c)
$u \coloneqq \text{Decompress}_q(\text{Decode}_{d_u}(c), d_u)$
$v \coloneqq \text{Decompress}_q(\text{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$
$\hat{s} \coloneqq \text{Decode}_{12}(sk)$
$m \coloneqq \text{Encode}_1(\text{Compress}_q(v - NTT^{-1}(\hat{s}^T \cdot NTT(u)), 1))$
return m

7.3 FHE

7.3.1 LWE

We introduced LWE in the previous section. The LWE problem revolves around the difficulty of distinguishing random linear equations from noisy versions thereof, forming the basis of various cryptographic primitives. This problem has gained prominence due to its resilience against quantum attacks, offering a robust foundation for post-quantum cryptographic schemes. Researchers are actively exploring lattice-based constructions, leveraging the hardness of LWE, to design secure and efficient cryptographic protocols.

$$Key\ \mathbb{A} \in \mathbb{Z}_q[n \times m]$$
$$LWE_A(s, e) = \mathbb{A}s + e(mod\ q)$$
$$e_{max} \leq \beta = \mathbb{O}(\sqrt{(}n))$$
$$q, m = poly(n)$$

7.3.2 Encryption Decryption with LWE

The concept of encryption with Learning With Errors (LWE) at its core revolves around leveraging the assumed hardness of the LWE problem to establish secure communication channels. In an LWE-based encryption scheme, the public key consists of random linear equations, while the private key involves the underlying secrets that satisfy these equations. The encryption process involves introducing random noise to the linear equations, creating a set of noisy equations that conceal the original secrets. The recipient, possessing the knowledge of the secrets, can then decrypt the message by effectively solving the system of equations and eliminating the noise.

The security of LWE-based encryption relies on the presumed computational difficulty of distinguishing between the original linear equations and the noisy

versions. This hardness assumption forms the foundation for the confidentiality of the encrypted messages. Importantly, LWE-based encryption schemes are known for their resilience against quantum attacks, presenting a viable alternative in the post-quantum cryptographic landscape.

One time pad:

$$b = LWE_A(s, e)$$

Secret key:

$$s \in \mathbb{Z}_q^n$$

Message

$$m \in \mathbb{Z}^m$$

For encryption, generate a random pair:

$$[A, e]$$

$$E_s(m; [A, e]) = [A, b + M]$$

where

$$b = As + e$$

Decryption is noisy:

$$D_s(A, b + m) = (b + m) - As = m + e \; mod \; q$$

When decrypting a ciphertext encrypted under an LWE-based scheme, the low-order bit of the intended message "m" becomes entangled with a noise term "e." This inherent noise complicates the accurate recovery of the original message, necessitating a careful mitigation strategy.

To address this issue, a common approach involves scaling up the entire message "m" and subsequently rounding off the low-order bits, which encapsulate the error term. This scaling-up operation effectively diminishes the influence of the noise term, allowing for a more accurate recovery of the intended message. The rounding process serves to isolate and discard the perturbed low-order bits, rectifying the impact of the error term on the decrypted result.

7.3.3 Circular Security

Circular security in Fully Homomorphic Encryption (FHE) represents a desirable property, allowing for the decryption of an encrypted private key to derive an encrypted form of the plaintext while maintaining security. This concept addresses the circularity problem inherent in FHE schemes, ensuring that operations involving encrypted keys do not compromise the overall security of the

encryption system. In essence, circular security facilitates secure and privacy-preserving computations, permitting users to perform operations on their encrypted data, including their own encrypted private keys, without compromising the confidentiality of the information.

$$E_s(m; [A, e]) = [A, b + M]$$

where

$$b = As + e$$

$$D_s(A, b + m) = (b + m) - As = m + e \; mod \; q$$

$$D_s([-A, 0]) = 0 + As$$

We can use the property to randomly encrypt the secret key.

$$[-A, 0] + E_s(0, \beta) = E_s(As, \beta)$$

$E(As)$ does not leak s

Decryption is also linear, this linearity gives us the basic homomorphic operations:

Add: $E(m_1, \beta_1) + E(m_2, \beta_2) \subset E(m_1 + m_2, \beta_1 + \beta_2)$

Neg: $-E(m, \beta) = E(-m, \beta)$

Mul: $c \times E(m, \beta) = E(c \times m, c \times \beta)$

Const: $[O, m] \in E(m, O)$

Key: $[-A, 0] \in E(As, 0)$

We can perform a limited number of additions and multiplications by small constants. Decryption is linear in the secret key $s^{'} = (-s, 1)$

7.3.4 Public Key Encryption

In Fully Homomorphic Encryption (FHE), the public key encryption aspect allows for the encryption of data using both the secret key and the public key. Typically, public key encryption in FHE involves two main operations: encryption and decryption.

Encryption: Users can encrypt plaintext data using the public key. This ciphertext, generated with the public key, can then be processed using FHE operations without the need for decryption.

Decryption: The encrypted data can be decrypted using the corresponding secret key, revealing the original plaintext. This step ensures that only those with the secret key can retrieve meaningful information from the encrypted data.

The ability to use the public key for encryption is particularly advantageous in scenarios where data needs to be securely transmitted to a party that possesses the secret key. This dual-key functionality enhances the flexibility and utility of FHE in various applications, offering a secure means of performing computations on encrypted data without compromising the confidentiality of the information.

Public key:

$$[a_1, b_1] = E_s(0), ..., [a_n, b_n] = E_s(0)]$$

Encrypt(m):

$$(\sum_i r_i \times [a_i, b_i + (0, m) + E_s(m, 0) = E_s(m)])$$

We can use a party's public key to encrypt, and the party can use its decryption key to decrypt.

Also, we can use homomorphic decryption to do multiplication, now that encryption is linearly homomorphic:

$$E(m) = (a, as + e + m)$$

$$D(a, b) = b - as = m + e$$

Decryption is linear in $s\prime = (-s, 1)$, we can decrypt homomorphically using an encryption of $s\prime$. Given:

$$E(m) = (a, b), E^{'}(s^{'} = (E^{'}(-s), E^{'}(1)))$$

We can compute:

$$E(m) \star E^{'}(s^{'}) = a \star E^{'}(-s) + b \star E^{'}(1) = E(m)$$

Furthermore, given $E(m)$ and $E^{'}(cs^{'})$

We can compute $E(m) \star E^{'}(cs^{'}) = E(cm)$

7.3.5 Bootstrapping FHE

Bootstrapping in Fully Homomorphic Encryption (FHE) is a critical process that addresses a fundamental challenge known as "noise accumulation." The core idea revolves around refreshing or "bootstrapping" an encrypted ciphertext to reduce the noise level, allowing for continued secure computations without compromising the integrity of the encrypted data.

In FHE schemes, performing homomorphic operations on encrypted data introduces noise, which accumulates with each computation. As the noise level increases, it can eventually reach a point where accurate decryption becomes challenging, potentially leading to errors in the computed results. Bootstrapping is essential to counteract this noise growth and maintain the security and correctness of FHE computations.

The process involves decrypting an encrypted ciphertext using the secret key and then re-encrypting the result using the same FHE scheme but with fresh noise. This effectively "refreshes" the ciphertext, reducing the accumulated noise and allowing for continued secure computations. Bootstrapping is particularly crucial in scenarios where long sequences of homomorphic operations are required, ensuring that the noise remains manageable and does not compromise the security guarantees of the FHE system.

The bootstrapping process works as the followings:

Let $c = Enc(m \star (q/2) + e)$

$f_S = msb(Dec_S(c)) \star (q/2) = m \star (q/2)$

Evaluate f_c homomorphically on $S = Enc_s(S)$

$f_c(S) = f_c(S) = msb(Dec_s(c)) = m \star (q/2) = Enc_s(m \star (q/2))$

After the process we derive an output with a noise level of $msb(Dec_s(c))$, but not on e, therefore we "refresh" the noise to a low base level.

7.3.6 FHE Summary

As explained above, encryption process encompasses several key principles, each contributing to the robustness and versatility of secure computations on encrypted data.

Circular security, as a foundational concept in Fully Homomorphic Encryption (FHE) and lattice cryptography, is designed to prevent the revelation of information about the original secret key when encrypting it using the secret key itself. While circular security lacks theoretical proof, empirical evidence suggests its effectiveness in bolstering the system against potential vulnerabilities arising from self-referential cryptographic operations. Despite the absence of formal proof, the robustness of circular security remains evident in practice, contributing to the overall resilience of FHE systems in real-world cryptographic scenarios.

Next, linear decryption characterizes the process of decrypting linearly encrypted data. This linear nature facilitates the secure and efficient execution of operations on encrypted data, forming a foundational aspect of FHE computations.

A transformative step involves converting an encryption E to E', where E' possesses the capability to evaluate arbitrary low-depth functions. This transformation enhances the expressive power of FHE, enabling the computation of diverse functions on encrypted data while maintaining security.

Introducing bootstrapping further extends the FHE capabilities. While bootstrapping allows for non-linear, low-depth functions, its primary role lies in mitigating noise accumulation. By refreshing the ciphertext using a transformed encryption E', bootstrapping effectively reduces noise levels, ensuring the sus-

tained security and accuracy of FHE computations over extended sequences.

In summary, the FHE encryption process encompasses circular security, linear decryption, transformation to E' for evaluating low-depth functions, and the crucial role of bootstrapping in both handling non-linear functions and maintaining low noise levels. These principles collectively underscore the sophistication and resilience of FHE in enabling secure computations on encrypted data within the lattice cryptography framework.

Bibliography

[1] Peter Shor *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer* 1995.08.30.
Shor 1995

[2] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, Marko Vukolić *Mir-BFT: High-Throughput Robust BFT for Decentralized Networks* 2019.06.13
Stathakopoulou 2019

[3] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, Malika Izabachene *TFHE: Fast Fully Homomorphic Encryption Over the Torus* 2019
Chillotti 2019

[4] Leo Ducas,Daniele Micciancio *FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second* 2015
Ducas,Micciancio 2015

[5] Jung Hee Cheon, Andrey Kim, Miran Kim, Yongsoo Song *Homomorphic Encryption for Arithmetic of Approximate Numbers* 2017
Cheon, Kim, Kim, Song 2017

[6] Vadim Lyubashevsky, Ngoc Khanh Nguyen, Gregor Seilor *Practical Lattice-Based Zero-Knowledge Proofs for Integer Relations* 2020
Lyubashevsky, Nguyen, Seilor 2020

[7] Vadim Lyubashevsky, Ngoc Khanh Nguyen *BLOOM: Bimodal Lattice One-Out-of-Many Proofs and Applications* 2022
Lyubashevsky, Nguyen 2022