Eigen ZKZRU: A Privacy-specific ZK Rollup

Eigen Labs

July 16, 2022

Abstract

Decentralized Privacy and security are always the most important part of Web3.0. ZCash and Monero have taken great effort to achieve privacy-enabling technology by building their private payment infrastructure and ecosystem. However, it's not easy to achieve programmable privacy on Smart Contract regarding general assets, such as ERC1155.

In this paper, we present Eigen ZKZRU, a privacy-specific ZKRollup solution on ZKRollup on Ethereum ecosystem to achieve low gas-fee, better composability and high privacy enhancement. Eigen ZKZRU proposes an Abstract Balance Leaf Model, using Twisted ElGamal encryption [CMTA20] and Bulletproof [BBB+18] on DDH assumption to build the confidential transaction, and constructs the validity proof of batch transactions on account Merkle tree by algebraic composite NIZK, which is built on a Two-tier architecture, the base layer is a ZKRollup, which is built on R1CS constraints system and transpiled to PLONKish Arithmetization [GWC19], which is naturally friendly to utilize Twisted ElGamal encryption with custom gate, and hardware acceleration. The application layer is Pairing-free and easy-to-audit confidential transaction layer.

1 Motivation

Privacy invasion incidents become more and more frequent in Web3.0 world. Observe that the privacy of Web3.0 participants can be categorized into 3 aspects: PII[1]-Account Linkage, Transaction's Privacy, and Smart Contract Logic's Privacy.

Eigen Network proposes Eigen ZKZRU, a brand-new approach to implement confidential transaction on EVM-compatible blockchain. Eigen ZKZRU comes up with ABL, Abstract Balance Leaf of account tree, and applies different cryptography primitives to ABL, such as ZK Membership proof, Homomorphic arithmetics, private set interaction, and private comparator etc, thus expose the high-level private user interface to wallet.

1.1 Background: Privacy Issues in Web3.0

1.1.1 Disclosure of PII-Account linkage

The blockchain technology provides an excellent for people to perform anonymous transaction without exposing their ID on internet, but this capability has been broken as the development of on-chain data analysis, such as Nanshan and Dune, this makes it possible for users to exploit the relationship between PII and account. Monaco's paper [Mon15] proposed an new approach Random time-interval(RTI) model to identify and verify Bitcoin users based on the observation of Bitcoin transactions over time. Bonifazi et.al's paper [BCUV22] adopts the social network-based model to represent Ethereum and construct each user's spectrum then classify them into difference clusters on their spectra. All those works taking efforts to link individual with their blockchain addresses deprive a user of anonymity. Furthermore, cracking the public transaction graph is big business: companies like Chainalysis and Nansen run sophisticated forensic analysis to associate various wallets, monitor activity, and make probabilistic assumptions about who owns what.

Those sort of privacy invasion incidents happen frequently. In this March, the Juno Network's proposal 4 was submitted to voted away an whale wallet's funds to the Juno community pool and executed in proposal 16.

1.1.2 Vulnerable privacy and confidentiality of Smart Contract

From the Etherscan, not only can we see all account's balance on different tokens, but also all the counterparties and contracts they interact with. The open and verifiable nature empower the

[1]PII: Personally Identifiable Information, see Personal data.

trustless infrastructure of Blockchain, however, this also leads lots of heavy fairness issue in DeFi world.

First notorious issue is Miner Extractable Value(MEV). Miner extractable value (MEV) is a measure of the profit a miner (or validator, sequencer, etc.) can make through their ability to arbitrarily include, exclude, or re-order transactions within the blocks they produce. Through MEV, Miner can make profit from exploiting the transaction in the txpool, and inserting specific transaction before or after user's transaction.

Apart from MEV, Slippage on Uniswap and other popular DEXes is a pain. Slippage is the price difference between when users submit a transaction and when the transaction is confirmed on the blockchain. Slippage can be caused by high trading volume or low liquidity apparently, but the one of the root causes is that miner choosing the high gas-fee transactions to confirming, and the arbitrators can leverage this and make an impact on others by increasing their gas-fee.

The decentralized Smart Contract is the core feature of Ethereum, however, the lack of confidentiality is a major hindrance towards the broad adoption of it, since financial transactions (e.g., insurance contracts or stock trading) are considered by many individuals and organizations as being highly secret.

1.1.3 Dilemma of utility versus privacy protection

Tornado Cash is the leading anonymous payment DApp in Ethereum ecosystem. As a Layer 1 Dapp, it supports ERC20 only currently, and charges about 50Gwei + 100Gwei + 0.05% 0.2% * 0.05ETH for even a single 0.05ETH from source[2].

Some well-known public blockchain protocols like Zerocash and Monero provide confidential transaction infrastructure but forgo programmability and unclear a primori how to enable programmability without exposing transaction and data in cleartext to miners.

Another utility issue is Nullifier is not account-friendly. The Nullifier is a public input that is sent on-chain to be checked with the smart contract and the Merkle tree data. It avoids double-spending for instance.

Because the Nullifier is irrelevant to the addresses of sender or receiver. So the receiver can do withdraw with the valid Nullifier, and cutting off the relationship between sender and the receiver.

However, this is not easy to manage such Nullifier, which is secret key for each anonymous payment transaction.

1.2 Vision

Eigen Network aims to facilitate a drop-in privacy preservation protocol for every Web3 participant in EVM-compatible ecosystem.

Eigen Network focuses on building a new privacy-specific ZKRollup protocol to achieve address anonymity, smart contract data and logic confidentiality and offers users capabilities of low gas-fee confidential transaction, full Smart Contract privacy preservation and better composability of supporting most used asset like ERC20,ERC721 and ERC1166, in EVM-compatible ecosystem.

1.3 Related Works

There are some concurrent work to implement address anonymity, transaction and smart contract privacy preservation.

Hawk[KMS+16] splits a smart contract into a public contract, which runs on-chain, and a private contract executed off-chain. Hawk requires the figure of a manager to execute the private contract. The manager must be trusted by the participants, as it can see the transactions that take place in a private contract. The manager cannot affect the outcome of the contract, it just could prevent the contract execution, or cause the abortion of the contract, and in both cases, it would be castigated.

Zether[BAZB20] is a private transaction scheme built on top of Ethereum. It can be extended to support a limited class of smart contracts with I/O privacy—namely those that can be represented solely via homomorphic addition. This allows us to perform simple sealed-bid auctions (assuming bidders buy all units on offer) and private voting (assuming the votes are binary). Unfortunately, only transactions hiding the users' balances and the transfer amount can currently be implemented on Ethereum due to gas constraints. Unlike previous two constructions, Zether uses a "transparent" ZKP (i.e. a ZKP with no trusted setup) Σ-*Bullets*.

[2]Fee - how much does it cost to use? https://torn.community/t/fee-how-much-does-it-cost-to-use/68/3

Table 1: A comparison of Privacy-preserving Smart Contract Scheme

Scheme	Privacy type	Approach	Expressivity	Extension as
Hawk	I/O	Offchain MPC/TEE	Arbitrary functions	Ethereum L1
Zether	I/O	Elgamal and Σ-Bullet	Additive functions	Ethereum L1
AZTEC	I/O, function	Pairing based homomorphic Commitment	Additive functions	Ethereum L2
Eigen ZKZRU	I/O, function	Pair-free Homomorphic PKE and Bulletproof	Arbitrary functions	Ethereum L2

Table 2: A comparison between Eigen Network and AZTEC Network

Capabilities \ Projects	Eigen ZKZRU	AZTEC ZK-ZKRollup
EVM Compatibility	Partial/Language Level	Partial/Language Level
Prover Decentralization	Decentralized Eigen Relayer	Wasm in Browser
Circuit DSL	Circom	Noir
Prove System	Plonk	Plonk
Recursive Proof	Yes	Yes
GPU acceleration enable	Yes	No
Anonymous Address	Stealth Address	Nullifier

The AZTEC Protocol [Wil18] proposed an Join-split transactions model, and each splits is described as notes, which is UTXO-like and attached with a viewing key and spending key. and the notes can be spent in encrypted state based homomorphic arithmetic and range proof.

Eigen Labs proposes Abstract Balance Leaf(ABL) on Account Balance Merkle Tree, and applies standard ERC20 and ERC721 operations on it based on high-level cryptography primitives, such homomorphic arithmetic and accumulator etc.

To sum up, Table 1 presents the detail on privacy type, approach, expressivity and which blockchain protocol to extend.

For privacy type, I/O is corresponding short for Encrypted Input/Output. If end user encrypts the input, and send it to the miner to run the contract, it would be Encrypted Input. Otherwise, the end user need run the contract locally with clear input and encrypts the output, then sends the encrypted output to miners.

There are different approaches to obtain privacy enhancement, The classic methods include Homomorphic arithmetic or some commitment scheme, TEE, MPC and NIZK. Homomorphic arithmetic is deeply limited to to performance in computation, especially fully homomophgic encryption, so the practical approaches would be commitment scheme, like additive or multiplicative to perform special operations like **Split** and **Join** in AZTEC Protocol. The MPC needs large of communication complexity and this raises complex interoperability between on-chain and off-chain. The TEE uses special hardware to achieve computing efficiency and code integrity. NIZKs, Non-interactive zero-knowledge proofs, enable a party, known as the prover, to convince another party, known as the verifier, about knowledge of the witness for an NP statement without revealing any information about the witness (besides what is already implied by the statement being true).

Expressivity is the scope of the scheme being able to protect.

A deep comparison with AZTEC from implementation perspective on acceleration and architecture in Table 2. Eigen ZKZRU leverages decentralized proof-Of-Proving mechanism to accelerate the proving computation, and it's naturally for us to utilize hardware acceleration, too, such as GPU and ASIC for NTT computation in polynomial commitment and multi-scalar multiplications on elliptic curves. The second difference is we adopt Stealth Address to implements the address anonymity.

2 Eigen ZKZRU Design Rationale

Eigen ZKZRU aims to facilitate a drop-in privacy preservation application for every Web3 participant in EVM-compatible ecosystem, provides user the capacities as below.
Low Gas-fee. Free deposit, Batch transfer by ZK-Rollup, Low gas-fee withdrawal.
Composability. Not only ERC20, both ERC721 and Swap are supported.
Privacy Enhancement. Confidential transaction and address anonymity based on ZK-ZKRollup
Easy-to-use. Self-custodial Secret(Private Key/Secret words) Management and Multisig Wallet based Account Abstract.

2.1 Overview of Eigen ZKZRU

Eigen ZKZRU is a Two-Tier architecture, the base layer is ZKRollup layer, which empowers applications with zero-knowledge proofs. The upper layer is the confidential transaction layer, which

provides high-level interface of confidential asset, which is pegged to the token in Layer 1. The Two-Tier architecture is necessary due to two factors as below.

Privacy-oriented protocol. Eigen ZKZRU is to provide a privacy "Lego" for end-users to protect their asset's confidentiality and security before the asset flowing into other liquidity pool or protocols in DeFi world. Differ from the zkEVM solution, such as ZKSync, we can deploy our confidential transaction layer on other ZKRollup L2s. Hence we design the transaction layer on pair free cryptography scheme to try best to save the gas and lower the implementation complexity, and for privacy concern, we need to generate the Zero-knowledge transaction in user-side, like by browser.

Acceleration of ZK Proving. Proving a computation requires first translating it from a classical program to a ZK-friendly format, then use proof system, such as Groth16, Plonk, STARK, to generate the validity proof. No matter which proof system used, the bottleneck always ends up being either:

- Multiplications over large vectors of number on Group or Fields, especially variable-base and fixed-base multi-scalar multiplications (MSMs);
- Fast Fourier Transforms (FFTs) and Inverse FFTs (although there are techniques for FFT-less proof systems).

In systems where both FFTs and MSMs exist, about 70% of the time generating a proof is spent on MSMs, and the rest is dominated by FFTs. There are some methods to improve the performance, but hardware acceleration does really matter. For the ZKRollup layer, we design a the proof-of-proving model for miner to participant in the computation, and the miner would provide powerful and optimized computation.

2.1.1 Base Layer: ZK Rollup

The ZK Rollup is a very promising approach to scale Ethereum. The ZK Rollup consists of Rollup contract and A off-chain proving nodes. The Rollup contract maintains a transaction Merkle tree root list and a account Merkle tree. The proving nodes act as:

- a RPC server for user to interact with ZK Rollup
- a sequencer to sort the transaction and generates the transaction Merkle tree root, submit the root into L1 Rollup contract
- a prover to generate the Rollup block validity proof, and update the global account Merkle tree.

The prover generate the validity proof by the EigenZKit. EigenZkit is our ZKRollup *compiler*, which make it easy for the developer to write constraints by Circom, transpile the circuits to PLONKish Arithmetization [GWC19], optimize the proving with the Lookup table and aggregation proof, and finally generate the proof and solidity verifier on Plonk argument.

The rollup contract maintains a state root: the Merkle root of the state of the Rollup (meaning, the account balances, contract code, etc, that are "inside" the rollup). And check the validity of the new proof with public input transaction Merkle tree root and new account Merkle tree root.

The basic data representation of rollup blockchain's world state see Figure 1. The depth could be scaled to 28 or larger, which means our ledger merkle depth can reach up to 2^{28} accounts.

2.1.2 Confidential Transaction Layer: Abstract Balance Leaf Model

To achieve the goal of transaction confidentiality, we proposes an Abstract Balance Leaf Model. Similar to the "note" in AZTEC, The ABL is also an encrypted representation of abstract value, which maps into high-level digital asset deposited into Eigen ZKZRU. The ABL is an output of the Eigen ZKZRU encryption value from formula 6.

The specific protocol of Confidential Transaction Layer will be presented in section 2.2, 2.3 and 2.4.

2.2 Fundamental

Pedersen commitment [Ped91] uses random public generators $\mathbb{G}$ and H of a suitable large group where the DDH3 is hard, for making a blinded, non-interactive commitment to a value. Eigen ZKZRU use Elliptic Curve Pedersen Commitment to hide the secret. Worth to note that we use

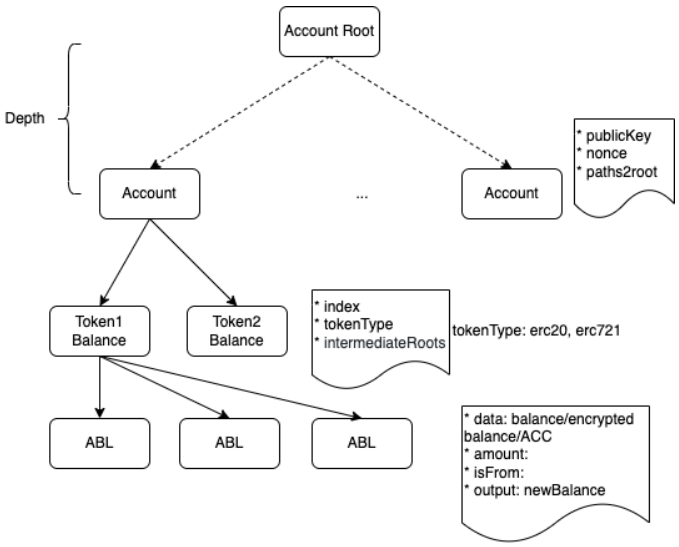

Figure 1: ZKRollup Account Tree

native Pedersen commitment scheme to describe our algorithm in this paper for easy understanding, and implements by Elliptic Curve Pedersen Commitment.

Definition 1. ***Discrate logarithm assumption.*** *Let $\mathbb{G}$ be a group of prime order q, a generator $g \in \mathbb{G}$ and an arbitrary element $y \in \mathbb{G}$, it's infeasible to find $x \in ZZ_p$, such that $y = g^x$.*

Definition 2. ***Elliptic Curve Pedersen Commitment[Fra15].*** *An efficient implementation of the Pedersen Commitment will use secure Elliptic Curve Cryptography (ECC), which is based on the algebraic structure of elliptic curves over finite (prime) fields. Let F_p be the group of elliptic curve points, where p is a large prime. Also, let $\mathbb{G}$ be the base point of EC, and H be another EC point, and the discrete logarithm of H no one has to know ($H = qG$). Attackers who know q can produce valid proofs without knowledge of x. To commit to a value $x \in \mathbb{Z}_p$, we pick a random integer $r \in \mathbb{Z}_p$ and calculate the commitment as:*

$$Com(x, r) = xG + rH$$

To open the commitment we simply reveal the values x and r. EC Pedersen is also additive homomorphic, so it has the following properties:

$$\begin{aligned} Com(x + y, rx + ry) &= Com(x, rx) + Com(y, ry) \\ Com(k \cdot x, k \cdot r) &= k \cdot Com(x, r) \end{aligned} \tag{1}$$

EC Pedersen Commitment could be used for hide several values in following way:

$$C(r, \overline{a}) = rH + a_1G_1 + a_2G_2 + ... + a_nG_n \tag{2}$$

where each G_n has to be formed using NUMS[BBC+14] approach.

Definition 3. ***Decisional Diffie-Hellman assumption.*** *Given some group $\mathbb{G}$, and element element g, and the elements g^a, g^b and g^c, determine whether $g^{ab} = g^c$, The DDH assumption holds if for any PPT adversary $\mathcal{A}$, we have:*

$$Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1] \leq \mathsf{negl}(\lambda) \tag{3}$$

Definition 4. ***A Non-interactive Zero-knowledge(NIZK) proofs.*** *A zero-knowledge proof of a statement does not reveal any information beyond the validity of the statement. A NIZK proof system for input x in language L, with witness w, is a set of efficient PPT algorithms (K, P, V) such that:*

1. *Setup: generates the common reference string on security parameters λ, $pp \leftarrow K(1^\lambda)$.*
2. *Prover: $\pi \leftarrow P(pp, x, w)$ produces the proof.*
3. *Verifier: $V(pp, x, \pi)$ outputs 0, 1 to accept/reject the proof.*

Which satisfy the completeness, soundness, and zero-knowledge properties.
Completeness. *$\forall x \in L, w \in R_L$, such that:*

$$Pr[pp \leftarrow K(1^\lambda); \pi \leftarrow P(pp, x, w) : V(pp, x, \pi) = 1] = 1 \tag{4}$$

Soundness. *Assume x can be decided when seeing pp.*

$$Pr[pp \leftarrow K(1^\lambda); \exists(x, \pi), s.t. V(pp, x, \pi) = 1] = \mathsf{negl}(\lambda) \tag{5}$$

Zero-knowledge. *There exists a PPT simulator split into two stages S_1, S_2 such that for all PPT attackers A, the two distributions are computationally indistinguishable:*

$1:$ $pp \to K$
$2:$ $(x, w) \leftarrow A(pp), s.t.(x, w) \in R_L$
$3:$ $\pi \leftarrow P(pp, x, w)$
$4:$ $output(pp, x, \pi)$

And the simulator output:

$1:$ $(pp, \tau) \to S_1(1^\lambda)$
$2:$ $(x, w) \leftarrow A(pp), s.t.(x, w) \in R_L$
$3:$ $\pi \leftarrow S_2(pp, x, \tau)$
$4:$ $output(pp, x, \pi)$

Where τ should be thought of as local state stored by the simulator (passed between stages).

2.3 Abstract Balance Leaf Model

Paring based cryptography vs ECC based cryptography. Pairing-based cryptography(PBC) is a function that maps a pair of points on an elliptic curve into a finite field to construct or analyze cryptographic systems. Bilinear pairings can be used to transport the discrete logarithm problem on a certain class of elliptic curves over a finite field to the discrete logarithm problem on a smaller finite field. PBC is widely used in ZKP, such as the KZG commitment, Groth16 and Plonk etc. However, pairings are notoriously expensive in implementation complexity and processing time, and this is not very friendly to confidential transaction on ZKRollup. Therefore, we try to use the ECC based scheme to implement the confidential transaction layer.

Commitment Scheme VS Public Key Encryption. Pure commitment-based approach suffers from several issues due to lack of decryption capability, senders are required to honestly transmit the openings of outgoing commitments (includes randomness and amount) to receivers in an out-of-band manner. This issue makes the system much more complicated, as it must be assured that the out-of-band transfer is correct and secure. Second, users must be stateful since they have to keep track of the randomness and amount of each incoming commitment. To solve this, AZTEC utilizes set membership proof commitment scheme [CC+08, ALT+15], defining a new commitment scheme $Com(k; a) := (\mu_k^a, \mu_k^{ka} h^a)$ on $[1; k_{\text{MAX}}] \times \mathbb{Z}_p^* \longrightarrow \mathbb{G} \times \mathbb{G}$, where a is random in $\mathbb{Z}_p$ and h is generator of group $\mathbb{G}$, and hide the balance in commitment. This scheme need a universal trust setup. On the other hand, Zether built the confidential transaction on ElGamal Encryption[ElG85], which provides addictive homomorphic, computational hiding and unconditionally binding. Zether proposed Σ-*bullets* to bridge the interoperability between Σ-*protocol* and Bulletproof[BBB+18], with no trust setup. However the brute-force enumeration is necessary to find out the clear balance m on g^m in the extreme case that the receiver received an unknown balance from sender.

Another recent work [CMTA20] proposed PGC(Pretty Good Confidentiality) scheme on twisted ElGamal scheme, admits transparent setup, and its security is based solely on the widely-used discrete logarithm assumption.

Twisted ElGamal encryption [CMTA20]. Twisted ElGamal encryption is a public key encryption scheme improved based on the simplified version of the ElGamal. Twisted ElGamal encryption retains additive homomorphism and is as efficient and secure as the standard exponential ElGamal. More importantly, it is zero-knowledge friendly. Note that the second part of twisted ElGamal ciphertext can be viewed as Pedersen commitment [9] under the same commitment key, whose DL (discrete logarithm) relation is unknown to all users. Such structure of twisted ElGamal makes

it compatible with all zero-knowledge proofs whose statement is of the form Pedersen commitment. In particular, one can directly employ Bulletproofs[BBB+18] to generate range proofs for values encrypted by twisted ElGamal in a black-box manner, and these proofs can be easily aggregated. Therefore, Twisted ElGamal enables us to easily devise zero-knowledge proofs for essential correctness of transactions and various application-dependent policies in a modular fashion.

Moreover, twisted ElGamal is very efficient. Compared with the most efficient reported implementation of Paillier PKE, twisted ElGamal is an order of magnitude better in key and ciphertext size and decryption speed (for small message space), two orders of magnitude better in encryption speed. Consequently, we achieve an efficient and effective protection approach for account privacy by designing a twist EIGamal-based homomorphic encryption scheme.

Eigen ZKZRU extends a twisted ElGamal encryption on elliptic curve, and combines with EdDSA signrature scheme to , and optimizes the Camenisch and Stadler [CS97] notation for proofs of knowledge 6. This scheme can be easily proved that it holds the unconditionally hiding and computational binding same as Pedersen Commitment by the work from [BAZB20, CMTA20] on the DDH Assumption. Hence, the final proof of Zero-knowledge scheme would be as follow in formula 6.

$$PK\{(m,k,r) : PK = g^k \wedge (X,Y) = (PK^r, g^r h^m)\} \wedge (0 \leq m < \text{MAX}) \tag{6}$$

There are four primary algorithms in twisted ElGamal, as presented below.

$\mathsf{Pgen}(1^\lambda)$ λ is security parameter. Run the Gen(1^λ), choose another generator element $h \leftarrow\!\!\$\, \mathbb{G}$. Output pp = (G, g, h, p) as global public parameters.

$\mathsf{KGen}(\mathbf{pp})$ On input pp, choose a secret key $sk \leftarrow\!\!\$\, \mathbb{Z}_p$, and set the public key $pk = g^{sk}$.

$\mathsf{Enc}(\mathbf{pk, m})$ on input the public key pk and a message $m \in M$: pick random $r \leftarrow\!\!\$\, \mathbb{Z}_p$, compute $X = pk^r, Y = g^r \cdot h^m$

$\mathsf{Dec}(\mathbf{sk, C{=}(X,Y)})$, calculate $h^m = Y/X^{\frac{1}{sk}}$, then figure out m from h^m by brute-force enumeration with time complexity $\mathcal{O}(\text{MAX})$, where $\text{MAX} = 2^{31}$. The Baby-step/Giant-step algorithm [Sha71] with an efficient table lookup scheme would decrease it to $\mathcal{O}\left(\sqrt{\text{MAX}}\right)$.

The above scheme can be proved IND-CPA secure(1-plaintext/2-recipient) base on the divisible DDH assumption from [CMTA20].

Furthermore, we construct the set membership proof on this scheme from the work [CC+08] and Boneh-Boyen signatures [JY09], and extend it to a pairing free scheme in Appendex B.

Edwards-curve Digital Signature Algorithm. Edwards-curve Digital Signature Algorithm (EdDSA[3]) is a digital signature scheme using a variant of Schnorr signature based on twisted Edwards curves. It is designed to be faster than existing digital signature schemes without sacrificing security. The sketch of EdDSA is presented in Algorithm 1.

Algorithm 1: EdDSA signature generation (sketch)

Input: Group parameters $(\mathbb{G}, g, p)$, signer's key pair $(sk, pk = g^{sk})$, signer's long-term secret n for session-key generation, and message M, pick cryptographic hash function H: $M \times \mathbb{G} \to \mathbb{Z}_p$

Output: Signature (R, s) of M

$r \leftarrow H(n, M) \mod p$;

$R \leftarrow rG$;

$h \leftarrow H(R, M) \mod p$;

$s \leftarrow r + h \cdot sk \mod p$;

return (R, s)

Zero-knowledge proofs on ABL. In ABL model, we ensure that the encrypted transaction are correct by using ZKP that claiming correctness without revealing any additional information. Since twisted ElGamal compatible with all zero-knowledge proofs whose statement is of the form Pedersen commitment, we can directly employ Bulletproof to generate the balance's range proof.

We define the a language $L_{transfer}$ to describe our ABL model in definition 7.

[3]EdDSA: https://datatracker.ietf.org/doc/html/rfc8032

Table 3: Simple Confidential Transaction Example

public key	balance(ABL)	constraints
...	...	...
y_{Alice}	$\text{Comm}(b_{Alice} - b_t,\ y_{Alice})$	$b_{Alice} - b_t \geq 0$
y_{Bob}	$\text{Comm}(b_{Bob} + b_t,\ y_{Bob})$	$b_{Bob} + b_t \geq b_{Bob}$
y_{Cathe}	$\text{Comm}(b_{Cathe}, y_{Cathe})$	...

$$
\begin{aligned}
L_{transfer} = \{((pk_s, pk_r, C_s, C_r, sk_s, v, r_1, r_2))| \\
pk_s = sk_s G \wedge C_s = \text{HPKE}.Enc(pk_s, v, r_1) \wedge \\
C_r = \text{HPKE}.Enc(pk_r, v, r_2) \wedge v \in [0, \text{MAX}) \wedge \\
\text{HPKE}.Dec(sk_s, \hat{C}_s - C_s) \in [0, \text{MAX})\}
\end{aligned} \tag{7}
$$

The conjunction of $L_{transfer}$ is $\pi_{transfer}$, the sketch of the prove process is described as below in Figure 2.

Figure 2: Caption

Proof sketch: $\pi_{transfer}$

	Prover		Verifier
1 :	**Prover**		**Verifier**
2 :	$w \leftarrow (r, v)$		
3 :	$a, b \leftarrow\$ \mathbb{Z}_p$		
4 :	$A_1 \leftarrow pk_1^a, A_2 = pk_2^b,$		
5 :	$B \leftarrow g^a h^b$		
6 :		$A_1, A_2, B \longrightarrow$	
7 :			$e \leftarrow\$ \mathbb{Z}_p$
8 :		$\longleftarrow \mathbf{e}$	
9 :	$z \leftarrow a + er$		
10 :	$t \leftarrow b + ev$		
11 :		$z, t \longrightarrow$	
12 :			check if
13 :			$pk_1^z = A_1 X_1^e$
14 :			$pk_2^z = A_2 X_2^e$
15 :			$g^z h^t = BY^e$

2.4 Payment Mechanism over Abstract Balance Leaf

We use Twisted ElGamal encryption [CMTA20] and Bulletproof [BBB+18] on DDH assumption to build the confidential transaction, and constructs the validity proof of batch transaction on account Merkle tree by algebraic composite NIZK. Without lose the generality, we also implement set membership proof on our scheme naturally, to support general asset private payment, such as ERC721, and ERC1155, Eigen ZKZRU extends the ABL, and proposal Zero-knowledge computation on ABL to support non-fungible transaction, order-based token exchange etc. For instance, the state in ERC20 token would be describe as below after after transaction t in Table 3.

To stimulate above computation regarding of the constraints, we use homomorphic PKE to encode the balance and transfer amount, use NIZK to enforce senders to build confidential transactions honestly and make validity publicly verifiable, and use digital signature to authenticate transactions. Let $HPKE = (Setup, KeyGen, Sign, Verify, Enc, Dec)$ be homomorphic PKE integrated with EdDSA signature scheme on message space $\mathbb{Z}_p$, the $HPKE$ uses same key for encryption and signing, which proposed a general construction from [PSST11]. And let $NIZK = (Setup, CRSGen, Prove, Verify)$ be a NIZK proof system. The privacy-preserving payment mechanism consists of a trusted setup, user algorithm and a smart contract.

Setup. The setup algorithm calls $Setup_{nizk}$ and $Setup_{nizk}$ as subroutines, which are the setup algorithms for the proof system and the signature scheme, respectively. The former setup could depend on the relations for which proofs are constructed. If these subroutines are trustless, then the whole setup is trustless, meaning that its correctness can be verified publicly. we use Plonk argument and EDDSA, the Plonk argument needs an trust setup and EDDSA is trustless.

Setup algorithm is formally described in Algorithm 2. Apart from setting up the proof system and signature scheme, it initializes account tree accountTree, minc contract, token registry contract, a last roll over epoch updateNumber to keep track of the last epochs accounts were updated, The setup also specifies an epoch length E and a MAXimum amount value MAX.

Algorithm 2: Payment Mechanism on ABL: Setup

Input: Security parameter λ

1 $pp_{hpke} \leftarrow \text{HPKE.Setup}(1^\lambda)$;

2 $pp_{nizk} \leftarrow \text{NIZK.Setup}(1^\lambda)$;

3 $crs \leftarrow \text{NIZK.CRSGen}(pp_{nizk})$;

4 $pp = (pp_{hpke}, pp_{nizk}, crs)$;

5 Initialize

- account Merkle tree contract: accountTree
- MiMC contract: mimc
- token registry address: tr
- updates and updateNumber for keeping track of the rollup blockchain
- transaction Merkle tree root array: txRoots

6 Deploy Rollup contract with parameters pp, accountTree, txRoots, minc, tr, MAX, BAL_DEPTH, TX_DEPTH

User algorithm. A user can run one of the following algorithms to interact with the smart contract. The output of these algorithms are raw transactions. We leave it implicit that they will be signed (using the public key of the Ethereum account from which they are sent) and destined to the Eigen ZKZRU smart contract. All the algorithms get the security parameter as input but we show it explicitly only for the first one. The algorithm is presented in Algorithm 3.

Smart Contract. The Smart Contract describe how users interact with Eigen ZKZRU. *Deposit* deposits the token on Rollup contract, and mint on L2, *Withdraw* does the exact opposite by burning the token on L2 and returns the token left back to L1. *Transfer* does the updates the cipher balance of both sender and receiver, and creates the proof $\pi_{transfer}$ for the state transition on L2. *BalanceOf* returns the clear balance of the account, and *UpdateAccountTree* generates the proof for each transaction batch, and updates the Rollup chain, which is maintained in Rollup contract.

In our scheme, users need generate a new key pair for each transaction, the private key acts as the blinding factor for the new commitment of the new transaction, and update previous blinding factors locally.

where the π_{block} would be generated by language 8.

Algorithm 3: Payment Mechanism on ABL: User algorithm

Input: $pp_{nizk}, pp_{hpke}, (\mathbb{G}, p, g, h)$

Function `CreateAccount`$(\hat{v}, sn)$**:**

- $sn \in \{0,1\}^n$
- $(sk, pk) \leftarrow \text{HPKE.KeyGen}(pp_{HPKE})$
- $\hat{C} \leftarrow \text{HPKE.Enc}(pk, \hat{v}, r)$
- **return** $sn, (sk, pk), \hat{C}$

Function `CreateTX`(sk_s, pk_s, v, pk_r)**:**

- $r_1, r_2 \leftarrow\!\!\$\ \mathbb{Z}_p$
- $C_s \leftarrow \text{HPKE.Enc}(pk_s, v; r_1)$
- $C_r \leftarrow \text{HPKE.Enc}(pk_r, v; r_2)$
- $memo \leftarrow (pk_s, pk_r, C_s, C_r) \in L_{transfer}$
- $w \leftarrow (sk_s, v, r_1, r_2)$
- $\pi_{transfer} \leftarrow \text{NIZK.}Prove(memo, w)$
- $\sigma \leftarrow \text{HPKE.}Sign(sk_s, sn, memo, \pi_{transfer})$
- **return** sn, $memo$, $\pi_{transfer}, \sigma$

Function `BalanceOf`$(sk, \hat{C})$**:**

- $\hat{v} \leftarrow \text{HPKE.Dec}(sk, \hat{C})$
- **return** $\hat{v}$

Function `VerifyTX`$(sn,\ memo,\ \pi_{transfer}, \sigma)$**:**

- $output_1 \leftarrow sn \geq sn^*$
- $output_2 \leftarrow \text{HPKE.Verify}(pk_s, (sn, memo, \pi_{transfer}, \sigma))$
- $output_3 \leftarrow \text{NIZK.Verify}(crs.memo, \pi_{transfer})$
- **return** $output_1 \wedge output_2 \wedge output_3$

Algorithm 4: Payment Mechanism on ABL: Smart Contract

Function `Deposit`$(b, \hat{C}, R, type, sn)$**:**

// `amount` b
// `encrypted amount` $\hat{C}$
// `blinding factor` $R = h^r$
// `token type` $type$
// `serial number` sn

1 **if** $type = 0$ **then**
2 require msg.value $= 0 \wedge$ b $= 0$
3 **else**
4 **if** $type = 1$ **then**
5 require b $\geq 0 \wedge$ msg.value $\geq$ b
6 transfer(ETH, msg.sender, address(this), b)
7 **else**
/* `transfer ERC20 token` */
8 transfer(tokenAddress, msg.sender, address(this), b)
9 **end**
10 **end**
11 $(X, Y) = \hat{C}$
12 $Y \stackrel{?}{=} g^b \cdot R$
13 amountComm $= \hat{C}$
14 depositHash = mimc.hash([y, amountComm, $type$])
15 accountTree.insertLeaf(depositHash)
16 st = accountTree.getRootFromProof()
return st

Function `Withdraw`$(pi_{withdraw}, txinfo, pk_r, b)$**:**

// `withdrawProof` $pi_{withdraw}$
// `transaction info` $txInfo$
// `recipient` pk_r
// `amount` b

17 uint txLeaf = mimc.hash(txInfo)
18 require txInfo.txRoot == accountTree.getRootFromProof()
19 require pp_{hpke}.verify($pi_{withdraw}$)
20 **if** *txInfo.token_type_from == 1* **then**
21 transfer(ETH, address(this), pk_r)
22 **else**
23 transfer(tokenAddress, address(this), pk_r)
24 **end**

Function `Transfer`$(sn, memo, \sigma, \pi_{transfer})$**:**

// `sender private key` sn
// `transaction amount` $memo$
// `sender private key` σ
// `transaction amount` $\pi_{transfer}$

25 $(pk_s, pk_r, C_s, C_r) = memo$
26 check msg.sender is pk_s
27 $X, Y = acc[y]$
28 NIZK.VerifyTX(sn, memo, $\pi_{transfer}, \sigma$)
29 $acc[pk_s] = acc[pk_s]/C_s$
30 $acc[pk_r] = acc[pk_r] \cdot C_r$

Function `BalanceOf`(sk)**:**

// `private key` sk

31 $\hat{C}$ = acc[y]
32 b = HPKE.BalanceOf(sk, $\hat{C}$)
33 **return** b

Function `UpdateTxTree`$(txRoot)$**:**

require msg.sender is sequencer
require msg.value > MinDeposit
txRoots.push(msg, txRoot)

Function `UpdateAccTree`$(\pi_{block}, txRoot, stateRoot)$**:**

// `block proof` π_{block}
// `public input` $x = stateRoot$

34 require txRoot is in txRoots
35 require prev(txRoot) is confirmed
36 require Verify(π_{block}, x) = true
37 currentRoot = stateRoot
38 updateNumber++
39 updates[currentRoot] = updateNumber

2.5 Address Anonymity

Eigen ZKZRU adopts fast dual key exchange based stealth address scheme for address anonymity.

In general, the Stealth address and Nullifier are two widely used scheme for anonymous transaction. Compared to Nullifier used by Tornado Cash and AZTEC, Stealth address is account-friendly. Stealth address enables the sender to send the money to an "invisible" recipient, and the actual recipient can derive the private key of the "invisible" recipient's address.

The widely-used Stealth Address protocol is Dual-Key StealthAddress Protocol (DKSAP), which is designed for a wallet solution ShadowSend uses two pairs of keys, a scan keypair and a spend keypair to provide decentralized anonymous currency. However, The drawback of DKSAP is that it requires the receiver to continuously calculate and determine whether it is the real receiver of the transaction until it detected a transaction that matches. In this process, the receiver needs to perform many time-consuming elliptic curve scalar multiplication operations, which limits the application of DKSAP. Some new Enhanced DKSAP protocols [CM17] proposed more efficient approach to eliminate the intensive curve scalar multiplication, we optimizes the Enhanced DKSAP protocol by binding the *txdata* to random r, and combining with the confidential transaction, we can directly move the assets of receivers in the transaction from the one-time address to real address in next section **??**.

In The Enhanced DKSAP protocol shown in Figure 3, the recipient have m keypairs (b_i, B_i) and one view keypair (v, V) on group $(\mathbb{G}, p, g)$, m+1 hash function $H_i, i \in [0, m]$ for previous key, and none of those keys will be open. A proxy server/audit is necessary to monitor the transactions on chain, which includes some R value and computes the pk', and check whether pk' matches pk from corresponding transaction. For each public key pk, only the correct receiver can calculate the private key $H_0(S) + H_i(S)b_i$ and spend the money.

The Enhanced DKSAP protocol

	Sender	Auditor	Receiver
2 :			$(b_i, B_i)_{1 \le i \le m}, (v, V)$
3 :			View key v $\longleftarrow$
4 :		$V \leftarrow vG$	
5 :		Scan pubkey V $\longleftarrow$	
6 :	$rr \leftarrow\$ \mathbb{Z}_p$		
7 :	$r = H(txdata \| rr)$		
8 :	$R \leftarrow rG, S \leftarrow rV$		
9 :	$pk \leftarrow H_0(S)G + \sum H_i(S)B_i$		
10 :		TX(pk, R, ...) $\longrightarrow$	
11 :		$pk' \leftarrow H_0(vR)G + \sum H_i(vR)B_i$	
12 :		$pk' \stackrel{?}{=} pk$	
13 :			TX(pk',...) $\longrightarrow$
14 :			$sk = H_0(S) + H_i(S)b_i$

Figure 3: The Enhanced DKSAP protocol

We extend the language $L_{transfer}$ previously in definition 7 to support stealth address. For simplicity, we always withdraw the asset in the stealth address to the B_0, whose balance is in encryption.

2.6 Proof-of-Proving model

One of the main advantages of the ZK Rollups is the fast finality of transactions that validity proofs guaranteed. To enhance the effectiveness, and decentralize proof generation, the PoP model aims to cover the key properties that such a consensus model for a L2 ZK Rollup requires:

- Permissionless: allow every computing power to participate in.
- Efficiency: sorting batch and generates the batch's validity proof deterministically.
- Decentralization: avoid control from any single party.
- Security: protection from malicious attacks.

Generally speaking, to speed up the transaction finality under L1's equivalent security guarantee as L1, a PoP model provides two basic capabilities:

- Transaction sorting: we first sort the transaction by sequencer, and submit the batch Mekle Proof into Rollup contract.
- Proof generation: the prover generates the block validity proof, update the account tree in Rollup contract.

sequencer. With regard to sequencer, there exists some general solution from Vitalc's Rollup guide[4]. And existing implementations include Chainklink FSS, or Polygon Hermez's Proof-of-Donation. To achieve more decentralization and efficiency, we use HotStuff [YMR+18] consensus algorithm to reach a high sorting performance. Once a batch of transaction selected by the a sequencer, it submits the batch as a proposal, if the proposal get the agreement from more than 2/3 sequencers, and then it submits the batch's txRoot into L1 Rollup contract, and puts down a large deposit. The deposit plus a reward will return to the user after the batch confirmed.

prover. Proof generation is a huge computation intensive work for prover and it's very necessary to get production of ZK validity proofs with high performance so the network can keep its service level. This is why we introduce a decentralized prover cluster, instead of generating the proof via compiling the prover into WebAssembly and running it user's browser as AZTEC.

To propose a new batch to the network, because the transactions have been ordered, The provers generate the proof, then submits the proof and public input to the Rollup contract. and it's easy to observe that, if give a fixed transaction or block gas limit per block, the finality can be quickly reached if and only if the proof is verified successfully.

Once mined, these data availability L1 transactions define the L2 transactions that will be executed and the specific order. This creates a deterministic new state, which can be computed as a virtual future state by network nodes.

Since the roles of prover are permissionless, the malicious provers maybe exist in proof-of-proving model. A malicious prover may tamper the transactions, or generate validity proof of less transactions than expected to save computation power. But this is not possible since the block validity proof includes the transactions' signature proof, and the transaction batch's Merkle proof. The former proof guarantees the individual transaction is valid, and the latter guarantees the transaction batch aggregation proof is valid.

If multiple provers run a race to produce Rollup blocks, the Rollup contract will receive the first one.

2.7 Rollup mechanism

Eigen ZKZRU is an implementation of Rollup in which the Node does not publish transaction data to the main chain, but only publishes the new Merkle root at every update after confirmation of a batched-up transaction, the basic protocol see Figure 4.

Node. The Nodes provide the standard RPC service for users to interact ZKZRU with native Wallet, like EigenSecret, or Metamask. The node would maintain a transaction pool, and execute the Proof-of-Proving model to reach an consensus of the order of the transactions, and submit the transaction Merkle root in L1 Rollup contract.

[4] An Incomplete Guide to Rollups: https://vitalik.ca/general/2021/01/05/rollup.html

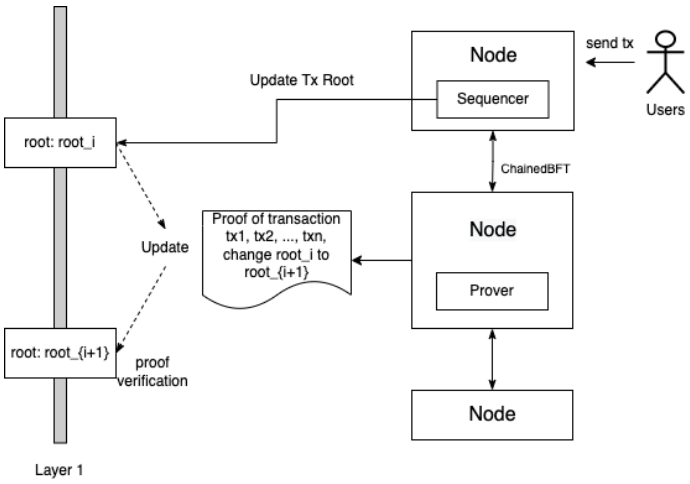

Figure 4: The Rollup mechanism

- Sequencer: The sequencers are designed to sort the transaction by PoS consensus, and submit the transaction root into L1 after it puts down a large deposit. if that user ever submits a fraudulent batch, that deposit would be part burned and part given as a reward to the fraud prover. To realize an decentralized sequencing mechanism, we adopt ChainedBFT to reach a consensus of the transaction order in L2.
- Prover: The provers are the actors that perform basic ZK Rollup functionalities. They are charged with creating blocks, making the transactions into a bundle, and performing the calculations and submitting the data to the main EVM-compatible chain for verification. The Provers retrieve a batch of transactions from L2 txpool, and generate the validity Zero-knowledge proof via EigenZkit, and submits the proof and public inputs to Rollup contract.

Rollup contract. The Rollup contract maintains the Rollup blockchain in L1. Once the block is produced by the Node, and submitted to the L1 Rollup contract, the *updateVerifier* is a ZKRollup to verify the validity of the proof π_{block} under the current account Merkle Tree. The main fields in the Rollup contract include:

- updateNumber: uint256, block number
- currentRoot: uint256, merkle root of account tree
- updateVerifier: NIZK Verifier for validity proof verification

We define the language L_{block} on algebraic composition of $L_{transfer}$ in Statement 8.

$$
\begin{aligned}
L_{block} = \wedge_{0<i<\text{TX\_DEPTH}}\{ & \\
& \{((pk_s^i, pk_r^i, C_s^i, C_r^i, sk_s^i, v^i, r_1^i, r_2^i))| \\
& pk_s^i = sk_s^i G \wedge C_s^i = \text{HPKE}.Enc(pk_s^i, v^i, r_1^i) \wedge \\
& C_r^i = \text{HPKE}.Enc(pk_r^i, v^i, r_2^i) \wedge v^i \in [0, \text{MAX}) \wedge \\
& \text{HPKE}.Dec(sk_s^i, \hat{C}_s^i - C_s^i) \in [0, \text{MAX})\}\}
\end{aligned} \tag{8}
$$

We translate above constraints into R1CS, transpile R1CS to Plonk Constraints, and generates the final proof π_{block} by PlonK [GWC19], which represents the vector of wire values as well as the different gates selectors as polynomials using interpolation. A polynomial division check ensures that the prover knows satisfying inputs and outputs for each gate, but does not ensure a correct wiring, e.g. that the output of one gate is the input to another.A permutation argument establishes the consistency of the assignment of wires to gates. The main reasons for this transpilation includes the URS setup, Pedersen Commitment/HPKE on custom gate etc.

Furthermore, the optimization for recursive proof, such as GPU-based acceleration for EC scalar multiplication and batch opening polynomial commitments, is applied to realize a very fast block validity proof verification on previous confirmed block in Rollup chain. The implementation is presented at EigenZKit.

A Bulletproof protocol

BulletProof doesn't depends on trust setup. BulletProof use NUMS D strategy, where a hash function is utilized to compute the generator for Pedersen Commitment. The main building block of BulletProof is Inner Product Argument(IPA). We use curve Secp256k1 in our implementation.

Algorithm 5: Compute Generator: ComputesGenerators

Input: The elliptic curve public generator $g \in \mathbb{G}$ and an integer n.
Output: The set of generators for Pedersen Commitment (g, h, **g**, **h**)

. Compute h = MapToGroup('something', p)
i = 0
while $i < n$ **do**
  $c \leftarrow\$ ZZ_p$
  $d \leftarrow\$ ZZ_p$
  g[i] $\leftarrow c \cdot G$
  h[i] $\leftarrow d \cdot G$
  i $\leftarrow$ i + 1
end
return (g, h, **g**, **h**);

Algorithm 6: $Setup_{IP}$

Input: the set of generators(g, h, **g**, **h**)
Output: $params_{IP}$

u $\leftarrow$ MapToGroup('something', p)
return $params_{IP}$ = (g, h, **g**, **h**, u)

Algorithm 7: $Setup_{RP}$

Input: the input interval [a, b), and the field modulus p

. **Output:** $params_{RP}$
if *b is not a power of 2* **then**
  return "b must be a power of 2"
$n \leftarrow log_2 b$
(g, h, **g**, **h**) $\leftarrow$ ComputeGenerators(g, n)
$params_{IP} \leftarrow Setup_{IP}(g, h, \mathbf{g}, \mathbf{h})$
return $params_{RP}$ = ($params_{IP}$, n)

B Boneh-Boyen Signatures without pairings

Boneh-Boyen Signatures scheme [JY09] used by the designated authority (which could be the verifier) to sign each element of the set Φ is the one proposed by Boneh and Boyen. Based on this scheme, which is secure under the q-SDH assumption, it is possible to prove knowledge of a signature on a message, without revealing the signature nor the message.

For a given secret x, and two random generatos g_1, g_2 of G_1, the signature of a message $m \in \mathbb{Z}_p$ is obtained by computing $\sigma = g_1^{1/(x+m)}$. Given a bilinear pairing e, a signature σ of m is valid if $e(\sigma, yg_2^m) = e(g_1, g_2)$ holds, where $y = g_2^x$.

[ALT+15] proposed BB signature without pairing under the DDH assumption. Let G be a cyclic group with prime order p where the Decisional Diffie-Hellman (DDH) problem is assumed to be hard and g_1, g_2 two random generators of $\mathbb{G}$. The signer's private key is $x \in \mathbb{Z}_p$ and it's public key is $y = g^x$, so the signature would be $\sigma = g_1^{1/(x+m)}$, which implies that $A^x = g_1 A^{-m}$. So the signer can prove the signature is valid by generating a ZKPK π:

$$SOK(x : X = g_2^x \wedge A^x = g_1 A^{-m}) \tag{9}$$

which means the discrete logarithm of $(g_1 A^{-m})$ in the base A is equal to the discrete logarithm of x in the base g_2.

Algorithm 8: Vector Commitment: $Commit_{IP}$

Input: $params_{IP}$, **a, b**

. **Output:** The commitment C

Compute C $\leftarrow g^a h^b \in \mathbb{G}$

return C.

Algorithm 9: Prove of Inner Product: $Prove_{IP}$

Input: $params_{IP}, commit_{IP}$,c, **a, b**

. **Output:** $proof_{IP}$

x $\leftarrow$ Hash(**g, h**, P, c) $\in \mathbb{G}$

Compute P' $\leftarrow u^{x \cdot c} P$

Allocate arrays **l, r** $\in \mathbb{G}^n$

ComputeProof(**g**, **h**, P'„ u^x, **a, b, l, r**)

$proof_{IP} \leftarrow$ (**g, h**, P', u^x, **a, b, l, r**)

return $proof_{IP}$.

Algorithm 10: Prove of Inner Product: ComputeProof

Input: g, h, P, u, **a, b, l, r**

Output: g, h, P, u, **a, b, l, r**

x = Hash(g, h, P, c $\in \mathbb{Z}_p^*$;

Compute P' = $u^{x \cdot c} P$;

if $n = 1$ **then**

  return (**g, h**, P, u, a, b, **l, r**) ;

else

  n' $\leftarrow$ n/2 ;

  $C_L \leftarrow < a_{[:n']}, b_{[n':]} > \in \mathbb{Z}_p$;

  $C_R \leftarrow < a_{[n':]}, b_{[:n']} > \in \mathbb{Z}_p$;

  $L \leftarrow g_{[n':]}^{a_{[:n']}} h_{[:n']}^{b_{[n':]}} u^{C_L} \in \mathbb{G}$;

  $R \leftarrow g_{[n':]}^{a_{[n':]}} h_{[:n']}^{b_{[:n']}} u^{C_R} \in \mathbb{G}$;

  Append L, R to **l, r**, respectively;

  x $\leftarrow$ Hash(L, R);

  $g' \leftarrow g_{[:n']}^{x^{-1}} g_{[n':]}^{x} \in \mathbb{G}^{n'}$;

  $h' \leftarrow g_{[:n']}^{x} g_{[n':]}^{x^{-1}} \in \mathbb{G}^{n'}$;

  $P' \leftarrow L^{x^2} P R^{x^{-2}} \in \mathbb{G}$;

  $a' \leftarrow a_{[:n']} x + a_{[n':]} x^{-1} \in \mathbb{Z}_p^{n'}$;

  $b' \leftarrow b_{[:n']} x + b_{[n':]} x^{-1} \in \mathbb{Z}_p^{n'}$;

  return ComputeProof(**g', h'**, P', u, **a', b', l, r**)

end

Algorithm 11: Prove of Inner Product: $Verify_{IP}$

Input: $params_{IP}, commit_{IP}, proof_{IP}$

Output: True or false

i $\leftarrow$ 0

while $i < logn$ **do**

  n' $\leftarrow$ n/2

  x $\leftarrow$ Hash(**l**[i], **r**[i])

  $g' \leftarrow g_{[:n']}^{x^{-1}} g_{[n':]}^{x} \in \mathbb{G}^{n'}$

  $h' \leftarrow g_{[:n']}^{x} g_{[n':]}^{x^{-1}} \in \mathbb{G}^{n'}$

  $P' \leftarrow L^{x^2} P R^{x^{-2}} \in \mathbb{G}$

  i $\leftarrow$ i + 1

end

The verifier computes c = a· and accepts if $P = g^a h^b u^c$.

Algorithm 12: Bulletproofs: $Prove_{RP}$

Input: $params_{RP}$, v
Output: $proof_{RP}$

$\lambda \leftarrow\$ \mathbb{Z}_p$
$V \leftarrow g^v h^\lambda \in \mathbb{G}$
$a_L \in \{\{0,1\}\}^n$ s.t. $< a_L, 2^n >= v$
$a_R - a_L - 1^n \in \mathbb{Z} + p^n$
$\alpha \leftarrow\$ \mathbb{Z}_p$
$A \leftarrow h^\alpha g^{a_L} h^{a_R} \in \mathbb{G}$
$s_L, s_R \leftarrow\$ \mathbb{Z}_p^n$
$\rho \leftarrow\$ \mathbb{Z}_p$
$S \leftarrow h^\rho g^{s_L} h^{s_R} \in \mathbb{G}$
$y \leftarrow Hash(A, S) \in \mathbb{Z}_p^n$
$z \leftarrow Hash(A, S, y) \in \mathbb{Z}_p^n$
$\tau_1, \tau_2 \leftarrow\$ \mathbb{Z}_p$
$T_1 \leftarrow g^{t_1} h^{\tau_1}$
$T_2 \leftarrow g^{t_2} h^{\tau_2}$
$x \leftarrow Hash(T_1, T_2) \in \mathbb{Z}_p^n$
$\mathbf{l} \leftarrow l(X) = a_L - z1^n + s_L X \in \mathbb{Z}_p^n$
$\mathbf{r} \leftarrow r(x) = y^n \cdot (a_R + z1^n + s_R X) + z^2 2^n \in \mathbb{Z}_p^n$
$\hat{t} \leftarrow <\mathbf{l}, \mathbf{r}> \in \mathbb{Z}_p$
$\tau_x \leftarrow r_2 x^2 + \tau_1 x + z^2 \lambda \in \mathbb{Z}_p$
$\mu \leftarrow \alpha + \rho x \in \mathbb{Z}_p$
$commit_{IP} \leftarrow Commit_{IP}(params_{IP}, l, r)$
$proof_{IP} \leftarrow Prove_{IP}(params_{IP}, commit_{IP}, \hat{t}, l, r)$
return $proof_{RP} = Prove_{RP}(\tau_x, \mu, \hat{t}, V, A, S, T_1, T_2, commit_{IP}, proof_{IP})$

Algorithm 13: Bulletproofs: $Verify_{RP}$

Input: $params_{RP}, proof_{rp}$
Output: True or false

$y \leftarrow Hash(A, S) \in \mathbb{Z}_p^n$
$z \leftarrow Hash(A, S, y) \in \mathbb{Z}_p^n$
$x \leftarrow Hash(T_1, T_2) \in \mathbb{Z}_p^n$
$h_i \leftarrow h_i^{y^{-i+1}} \in \mathbb{G}, \forall i \in [1, n]$
$P_l \leftarrow p \cdot h^\mu$
$P_r \leftarrow A \cdot S^x \cdot g^{-z} \cdot (h')^{z \cdot y^2 + z^2 \cdot 2^n} \in \mathbb{G}$
$output_1 \leftarrow (P_l \stackrel{?}{=} P_r)$
$output_2 \leftarrow (g^{\hat{t}} h^{r_x} \stackrel{?}{=} V^{z^2} \cdot g^{\sigma(y,z) \cdot T_1^x \cdot T_2^{x^2}})$
$output_3 \leftarrow Verify_{IP}(proof_{IP})$
return $output_1 \wedge output_2 \wedge output_3$

C Set Membership Proof on BB Signature without pairing

Definition 5. ***Borreomean ring signature[MP15]*** *let $\mathcal{V}$ be some set of verification keys, and f be a function which maps finite subsets of $\mathcal{V}$ to $0, 1$, we call f an admissibility function, then an admissible set V of verification keys is one for which $f(\mathcal{V}) = 1$.*

A Borromean ring signature σ is a signature on a message m with a set V of verification keys and admissibility function f which satisfy the following:

1. *σ can be produced only by parties who together know all the secret keys to an admissible set $\mathcal{V}$ of verification keys.*
2. *Given only σ , $\mathcal{V}$, and m, it is statistically indistinguishable which admissible set V was used.*

Proving knowledge of a valid BB Signature $A_k = g^{1/y+k}$ on a value k commited in $Com = g_1^k h^v$, without using parings on the prover's side can be done in Algorithm 14 and Algorithm 15.

Algorithm 14: Set Membership Proof: $Prove_{SM}$

Input: challenge ch, public input=(set Φ), private input = $(k, A_k = g^{1/y+k})$
Output: $proof_{SM}$

$v \leftarrow\$ \mathbb{Z}_p^*$;
$Com \leftarrow g_1^k h^v$;
$A_k \leftarrow g^{1/(y+k)}$;
$l \leftarrow\$ \mathbb{Z}_p^*$;
$B \leftarrow A_k^l, B_1 \leftarrow B^{-1}, D \leftarrow B_1^k g^l$;
$k_1, l_1, r_1 \leftarrow\$ \mathbb{Z}_p^*$;
$Com_1 \leftarrow g_1^{k_1} h^{r_1}, D_1 = B_1^{k_1} g^{l_1}$;
$C \leftarrow H(Com, B, D, Com_1, D_1, ch)$;
$s_1 \leftarrow k_1 + c \times k \mod p$;
$s_2 \leftarrow r_1 + c \times r \mod p$;
$s_3 \leftarrow l_1 + c \times l \mod p$;
$proof_{SM} = (Com, B, D, s_1, s_2, s_3)$

Algorithm 15: Set Membership Proof: $Verify_{SM}$

Input: challenge ch, $proof_{SM}$
Output: True or false

$\hat{C} \leftarrow g_1^{s_1} h^{s_2}, \hat{D} = B_1^{s_1} g^{s_3} D^{-c}$;
$output_1 \leftarrow B \neq 1_{G_1}$;
$output_2 \leftarrow H(Com, B, D, \hat{C}, \hat{D}, ch)$;
return $output_1 \wedge output_2$

D Nothing Up My Sleeve

This method enables the generation of verifiable random values. NUMS allows a prover to pick values in a way that demonstrates the values were not selected for a "nefarious purpose" - for example, to create a "backdoor" to the algorithm [BBC+14]. The algorithm on curve Secp256k1 is described in Algorithm 16.

Algorithm 16: Nothing Up My Sleeve: MapToGroup

Input: The input string m, and the filed prime modules p
Output: AN elliptic curve point if successful or some error

```
1  i ← 0 ;
2  while i < 256 do
3  |  x ← Hash(m, i) ;
4  |  rhs ← x^3 + 7 ;
5  |  if rhs is a sqaure (mod p) then
6  |  |  y ← √rhs mod p ;
7  |  |  if (x, y) is not the point at infinity then
8  |  |  |  return (x, y) ;
9  |  i ← i + 1;
10 end
11 return "Can not map to group";
```

References

[ALT+15] Ghada Arfaoui, Jean-François Lalande, Jacques Traoré, Nicolas Desmoulins, Pascal Berthomé, and Saïd Gharout. A practical set-membership proof for privacy-preserving nfc mobile ticketing. *arXiv preprint arXiv:1505.03048*, 2015.

[BAZB20] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

[BBC+14] Benjamin Black, Joppe W Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Elliptic curve cryptography (ecc) nothing up my sleeve (nums) curves and curve generation. *Tech. Rep.*, 2014.

[BCUV22] Gianluca Bonifazi, Enrico Corradini, Domenico Ursino, and Luca Virgili. Defining user spectra to classify ethereum users based on their behavior. *Journal of Big Data*, 9(1):1–39, 2022.

[CC+08] Jan Camenisch, Rafik Chaabouni, et al. Efficient protocols for set membership and range proofs. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 234–252. Springer, 2008.

[CM17] Nicolas T Courtois and Rebekah Mercer. Stealth address and key management techniques in blockchain systems. In *ICISSP 2017-Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, pages 559–566, 2017.

[CMTA20] Yu Chen, Xuecheng Ma, Cong Tang, and Man Ho Au. Pgc: Decentralized confidential payment system with auditability. In *European Symposium on Research in Computer Security*, pages 591–610. Springer, 2020.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Annual International Cryptology Conference*, pages 410–424. Springer, 1997.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[Fra15] BF França. Homomorphic mini-blockchain scheme, 2015.

[GWC19] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[JY09] David Jao and Kayo Yoshida. Boneh-boyen signatures and the strong diffie-hellman problem. In *International Conference on Pairing-Based Cryptography*, pages 1–16. Springer, 2009.

[KMS+16] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[Mon15] John V Monaco. Identifying bitcoin users by transaction behavior. In *Biometric and surveillance technology for human and activity identification XII*, volume 9457, pages 25–39. SPIE, 2015.

[MP15] Gregory Maxwell and Andrew Poelstra. Borromean ring signatures. *Accessed: Jun*, 8:2019, 2015.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[PSST11] Kenneth G Paterson, Jacob CN Schuldt, Martijn Stam, and Susan Thomson. On the joint security of encryption and signature, revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 161–178. Springer, 2011.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. 1971.

[Wil18] Zachary J Williamson. The aztec protocol. *URL: https://github.com/AztecProtocol/AZTEC*, 2018.

[YMR+18] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069*, 2018.