# WaterCarver: Anonymous Confidential Blockchain System based on Account Model

Jiajun Xin*, Pei Huang†, Lei Chen*, Xin Lai*, Xiao Zhang*, Wulu Li*, Yongcan Wang*,
* Shenzhen Onething Technologies Co., Ltd., Shenzhen, China
† Department of Electrical and Computer Engineering, Clemson University
Clemson, SC 29634

***Abstract*—The Account-model-based blockchain system gains its popularity due to its ease of use and native support of smart contracts. However, the privacy of users now becomes a major concern and bottleneck of its further adoption because users are not willing to leaving permanent public records online. Conventionally, the privacy of users includes transaction confidentiality and anonymity. While confidentiality can be easily protected using confidential transaction technique, anonymity can be quite challenging in an account-model-based blockchain system, because every transaction in the system inevitably updates transaction sender's as well as receiver's account balance. Even when the privacy of a blockchain system is well-protected, however, regulation becomes a new challenge to counter for further adoption of this system.**

**In this paper, we introduce a novel transaction-mix protocol, which provides confidentiality and, moreover, anonymity to account-model-based blockchain systems. By leveraging the state of art verifiable shuffle scheme to construct a shuffle of confidential transactions, we build one practical anonymous confidential blockchain system, WaterCarver, upon account model with the help of confidential transactions, verifiable shuffle, and zero-knowledge proofs. We further provide an efficient and robust solution for dynamic regulation with multiple regulators. Our regulation method achieves flexibility and perfect forward secrecy. Experiments show that the overall Transactions Per Second (TPS) of our system can be as large as 600 on a simple desktop.**

***Index Terms*—blockchain, Ethereum, privacy, regulation**

## I. Introduction

The advancing cryptographic techniques discover, address and solve plenty of security and privacy problems in blockchain. These problems span over voting [1], [2], auction [3]–[5], billing [6]–[8], cloud computing [9]–[11], etc. Although a large percent of these problems are well-responded, they are handled individually instead of as a whole one. Therefore, we seek for a platform that can potentially solve multiple problems instantaneously or at least provide basic components to them for decreasing implementation costs, like a silver bullet. Ethereum [12] has the potential to be such a platform. It supports smart contracts that run on Ethereum and have enforced verifiable execution. However, privacy in Ethereum is not well-protected.

Ethereum is a public network like other blockchain systems, where miners validate and verify transactions for profit. Validated transactions and updated account balances are recorded within blocks in the global ledger. In this paradigm, each user's privacy is directly violated as their account balance and transaction history are accessible to everyone. Similarly, all results of smart contracts are also public for review.

While there are lots of works proposed to solve security and privacy issues for blockchain systems [13]–[15], only few of them [13] have met all the challenges including efficiency, anonymity, confidentiality, and decentralization for account-model-based blockchain systems, e.g. Ethereum. Regulation of a privacy-preserving blockchain system is also a crucial open issue. Some finical organizations have already shown great interest in cryptocurrency [16], [17] to embrace regulation for the market. The regulation process should be robust and adaptive to different scenarios. If not treated properly, Monero, a UTXO-based anonymous confidential blockchain system, which is responsible for holding money laundry records for cyber crimes [18], is a example of the consequences.

What this paper plans to accomplish is fourfold: (1) privacy, including anonymity and confidentiality; (2) flexible and adjustable regulation; (3) usability for large-scale adoption; (4) restrained modification of Ethereum for better adoption and universality. Our modifications are constrained to the first two layers of a blockchain system [19], data layer and the network layer, such that our proposed system is capable of accommodating any existing consensus protocols, incentive mechanisms, etc. in the higher layers.

**WaterCarver.** To tackle the above challenges, we propose an anonymous confidential blockchain system based on the account model, WaterCarver. In our proposed scheme, we use confidential transaction techniques to protect transaction confidentiality. Transaction senders send confidential transactions to convert their account balance to UTXOs stored in a transaction pool. Distributed untrusted third parties, named as shuffle service providers, are involved to shuffle the confidential UTXOs to achieve high-level anonymity. Transaction receivers retrieve the shuffled UTXOs using zero knowledge proof [20]. The knowledge of shuffling and retrieving UTXOs is transmitted through private communication among senders, shuffler service providers, and receivers. To alleviate users' off-chain operations, we also provide a variant of our proposed scheme, which does not rely on private communication but requires more storage as a trade-off. The regulation process is robust and flexible with modified Bulletproof and verifiable

shuffle.

**Contribution.** Our contributions are described as follows:

- Confidentiality and anonymity are achieved simultaneously with verifiable shuffle and zero-knowledge proofs. Although there are some works [21] use verifiable shuffle technique as a component for plaintext UTXO coin mixer, we are the first to apply it to confidential transactions.
- We propose a flexible yet robust regulation methodology for the anonymous confidential blockchain system. In our methodology, regulators are allowed to join the system dynamically through the consensus process. The correctness of the blockchain will not be affected by regulators' malicious or careless behaviors. Our scheme balances the conflicting user privacy, decentralized structure of blockchain, and regulation.
- The usability of our system is considerably high based on the results of conducted experiments and benchmarks.

**Organization.** The rest of this paper is organized as follows: Background knowledge on Ethereum is introduced in Section II. Section III describes the models of our proposed scheme, followed by the design goals in Section IV. We provide preliminaries in Section V. The algorithmic components of WaterCraver are listed in Section VI and elaborated in details in Section VII. The experiments results are analyzed in Section VIII. Related work is provided in Section IX. Section X discusses the results and concludes the paper.

## II. Backgrounds on Ethereum

Ethereum [12] is the first and most influential account-based blockchain system. Unlike most UTXO-based blockchain systems where properties are carried by transactions, Ethereum records each user's property in the corresponding account. The transfers of a user's property are marked by the transactions this user involved, and the account balance of both the sender and the receiver is updated afterward. Consequently, a transaction must include the amount transferred, the sender's signature, both the sender and the receiver's account address. Note that in actual implementation, the sender's address is omitted and recovered using the confirmed signature due to storage size constraints.

According to [12], a block in Ethereum consists of a block header, where the hash value of the previous block, block number, consensus-related information, state root, transaction root, receipt root are stored, and a block body. The aforementioned three roots are described as follows:

- The state root that can be used to verify all state information, including the account balance, formally $H_r$;
- The transaction root that can be used to verify all the transactions in this block, formally $H_t$;
- The receipt root that can be used to verify all the receipts in this block, formally $H_e$.

With these three dedicated Merkle tree roots, light clients can easily verify the existence of one transaction, the balance of one account, etc. with a little extra effort on synchronizing the latest block header and corresponding Merkle path.

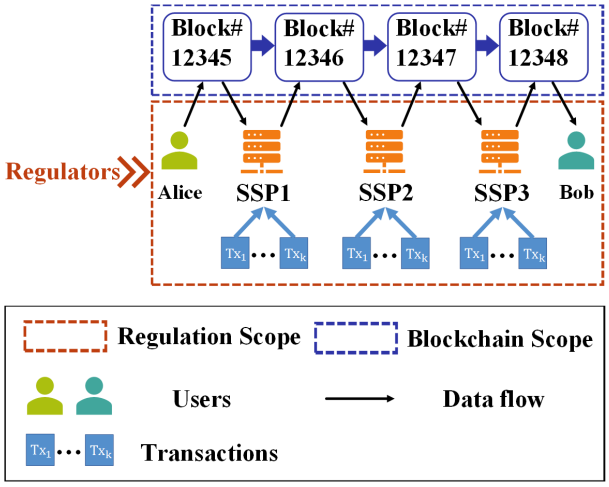


Fig. 1. System Model.

To measure the execution cost in Ethereum, a concept named gas is introduced. When one transaction is verified by a miner, the gas is the cost paid to the miner to cover the expense including signature verification. Gas system was originally designed to help Ethereum resist Denial-of-Service (DoS) attack. Later on, it implies an extra advantage that a complicated transaction becomes more welcomed by miners as long as a sufficient gas is paid.

Smart contract is the most important feature of account-based blockchain. In this paper, however, we focus on the privacy and regulation of transactions. It is straightforward to adapt our privacy and regulation schemes to smart contracts.

## III. Models

In this work, we provide an anonymous confidential blockchain system that can be integrated with a total plaintext system, like Ethereum. In this section, we describe how the system achieves anonymous confidential property.

### A. System Model

In an anonymous confidential blockchain system[1], there involves three prerequisite entities: users, blockchain peers and shuffle service providers (SSPs), as shown in Fig. 1. Regulators, which work as a complementary entity, participate in the system dynamically.

The users in one transaction include a sender and a receiver. We use Alice to denote one specific sender and Bob to denote a specific receiver. When Alice generates one transaction, she distributes the transaction to the blockchain peers. Blockchain peers are P2P network nodes with strong computation powers. On the one hand, they act as miners to propose blocks and receive mining rewards and gas; on the other hand, they verify

[1]We will use the term "this system" to refer the anonymous confidential blockchain system.

blocks to keep the blockchain correct and consensual. Then the transaction is on-chain and shuffled after processed by blockchain peers, so the receiver Bob can claim the transaction he owns and add the transaction output to his account balance.

The shuffle services are provided by user-selected SSPs, which are nodes with strong computation powers working for gas. One SSP receives $k$ transactions to shuffle as inputs and outputs another $k$ transactions. The input and output should be a valid permutation to provide anonymity to transactions. SSPs publish their static public keys and IP addresses for secure communication.

To regulate the system, regulators join the system through the consensus process in order to be globally accepted. By providing public keys and enforcing extra regulation information along with transactions, the regulators can gain knowledge of transaction amount or user information according to the regulation rules.

Without loss of generality, we assume that peers are honest in the majority. In other words, honest peers dominate the consensus process. In this paper, the peers decide the regulators through the consensus process and in turn the regulators affect block verification. Regulation information is recorded on-chain for announcement and block verification.

We assume that most selected SSPs are malicious and may collude with each other, while at least one of which is honest. The role of user, SSP and peer do not conflict with each other and one node can behave like all of them. We assume regulators are semi-honest, which means that they follow the protocol correctly, keep the gained information private but try to gain as much benefit as possible.

### B. Transactions Model

In correspondence with system model, there are three kinds of transactions: $\mathrm{tx}_{\mathrm{send}}$, $\mathrm{tx}_{\mathrm{recv}}$ and $\mathrm{tx}_{\mathrm{shuffle}}$.

A $\mathrm{tx}_{\mathrm{send}}$ is sent on-chain by the transaction sender. It contains the information of its sender and transaction amount, working as a UTXO without a receiver. The sender's account balance is updated by subtracting the same amount as the $\mathrm{tx}_{\mathrm{send}}$ contains. Upon receiving a $\mathrm{tx}_{\mathrm{send}}$, the $\mathrm{tx}_{\mathrm{recv}}$ adds the amount to the receiver's account balance and the received $\mathrm{tx}_{\mathrm{send}}$ is considered to be void. A $\mathrm{tx}_{\mathrm{send}}$ and a $\mathrm{tx}_{\mathrm{recv}}$ together make up a complete transaction process. Since the complete transaction process is different from the process in an original account based blockchain system, we define the overall transaction per second as follows:

**Definition 1** (Transactions per second)**.** *One transaction is completed when it is received by the receiver. The overall transactions per second (TPS) is calculated by the average number of* $\mathrm{tx}_{\mathrm{recv}}$ *per second.*

A $\mathrm{tx}_{\mathrm{shuffle}}$ is a special kind of transaction to provide anonymity for the system. It collects $k$ $\mathrm{tx}_{\mathrm{send}}$ for permutation, outputs $k$ new $\tilde{\mathrm{tx}}_{\mathrm{send}}$ with same amounts. After one round of shuffle, the sender information is erased. We specially refer this kind of $\tilde{\mathrm{tx}}_{\mathrm{send}}$ as UTXO. The input $\mathrm{tx}_{\mathrm{send}}$ are voided after permutation. We further introduce the concept of transaction pool.

**Definition 2** (Transaction pool)**.** *The transaction pool is the set of all valid UTXOs in the system.*

The transaction pool is filled by UTXOs associated with the $\mathrm{tx}_{\mathrm{send}}$ and $\mathrm{tx}_{\mathrm{shuffle}}$'s output while being deleted by $\mathrm{tx}_{\mathrm{recv}}$ and $\mathrm{tx}_{\mathrm{shuffle}}$'s input. All users, peers, and SSPs can calculate the transaction pool of the current block from the genesis block[2] with the transaction history. The current transaction pool is presented in the form of a trie root in the block header through consensus. In this way, all users can easily judge if one transaction is in the current transaction pool.

### C. Block Model

In this system, the block header contains two extra roots against the header in Ethereum: pool root and strategy root.

Pool root, formally $H_{pool}$, is the hash of a trie[3] root node. This trie contains information on the current transaction pool, which can be calculated using the aforementioned method and turned into a trie as described in [12].

Strategy root, formally $H_s$, is the hash of a trie root node that contains the regulation strategy for the next block.

We allow users to convert between their own plaintext account balance and confidential account balance. The convert records are preserved on-chain for verification.

The block body contains the three transactions ($\mathrm{tx}_{\mathrm{send}}$, $\mathrm{tx}_{\mathrm{recv}}$ and $\mathrm{tx}_{\mathrm{shuffle}}$) and convert records, together with the complete strategy trie. The strategy trie is provided by peers and regulators and voted through the consensus process for users' acceptance.

## IV. Design Goals

There are lots of threats against the blockchain system [22], e.g., DoS attack and double spending, most of which are already well-responded by Ethereum. Therefore, we focus on two unaddressed subjects in this paper: privacy and regulation.

### A. Privacy.

Anonymity and confidentiality are the primary concerns we considered in a blockchain system. Anonymity refers to the non-identifiability of the sender and the receiver in one transaction while confidentiality means that the transaction amount is only known by the sender and the receiver. In this part, we focus more on anonymity.

Our system should achieve robust $k$-anonymity [23], which is a classical and widely used model to protect user privacy and guarantees the ambiguity among $k$ participants. The idea of $k$-anonymity is to ensure any one of the $k$ inputs maps to any one of the $k$ outputs with similar possibility and its implementation varies in different scenarios. However, all the

[2]In blockchain systems, genesis block refers to the first block of one system.

[3]It can also be implemented using Merkle tree or similar structures, we use trie for consistency with Ethereum

existing $k$-anonymity schemes are somewhat vulnerable to interference attacks especially when $k$ is small [23].

### B. Regulation.

We focus on three aspects of blockchain regulation: correctness, flexibility, and security.

**Correctness.** Correctness is the foundation of a valid regulation. We consider the regulation correctness requires regulators correctly regulating the system within its regulation scope. More specifically, if peers in the system check and guarantee the correctness of proof and signature in $\mathrm{tx}_{\mathrm{send}}$ and $\mathrm{tx}_{\mathrm{shuffle}}$, then regulators can correctly recover the hidden amount or permutation used in the transaction.

**Flexibility.** Regulation flexibility means the regulation strategy of one blockchain system, including each regulator's regulation scope and public keys can be easily updated. Since blockchain is a decentralized system, a flexible regulation strategy should be decided by peers through the consensus process, and announced in block $i$, and take effect only in the block $i+1$. The strategy from the last round of the consensus should be exactly followed by transactions and breed a new strategy for the next round.

The flexibility is not only valuable for a blockchain with dynamic regulation strategy, but also very helpful for long-term static regulation strategy. With this flexibility, a long-term regulator can easily update its public key periodically for security. It only needs to negotiate with peers, update the regulation key through the consensus and announce in the block.

**Secrecy.** The extra information or trapdoor accompanying the regulation process should not harm the secrecy of the original blockchain system. We carry out the following definition:

**Definition 3** (Secrecy)**.** *A regulation method achieves secrecy if a PPT adversary has a negligible probability to impact the functionality or gain knowledge regarding transactions based on extra regulation information.*

Because the blockchain system shares its data all over the network for peer verification and all historical data cannot be deleted or modified. The compromise of secret keys inevitably causes data leakage. More importantly, there is no way to detect the data leakage or stop the damage since all the data can be already stored locally by the adversary. Therefore, we further define perfect forward secrecy as follow:

**Definition 4** (Perfect Forward Secrecy)**.** *A regulation method achieves perfect forward secrecy if the regulator's long-term private key is compromised and the secrecy still holds for previous regulatory activities.*

### C. TPS.

A high TPS clearly benefits the usability and more importantly, user privacy, especially for $k$-anonymity-based shuffle algorithms. This is because $\mathrm{tx}_{\mathrm{shuffle}}$ can only shuffle $\mathrm{tx}_{\mathrm{send}}$ in the transaction pool and its size is the theoretical upper bound of $k$. Suppose the average shuffle rounds is $\tilde{n}$ decided by users' habit. Then, the size of transaction pool can be estimated by TPS $\times\ \tilde{n}$ where $\tilde{n}$ is not system-controllable. Therefore, we push the upper bound of this system's TPS higher to make the transaction pool larger.

## V. Preliminaries

### A. Assumptions.

$\mathbb{G}$ is a group of order $p$ with a generator $g$. We assume decisional Diffie-Hellman (DDH) assumption holds in $\mathbb{G}$, which states that it is computationally intractable to differentiate the value $g^{ab}$ from a random element in $\mathbb{G}$ given $g^a$ and $g^b$ for uniformly and independently chosen $a, b \in \mathbb{Z}_p$.

### B. General Notations.

| Notations | Description |
|---|---|
| $\mathbb{G}^n$ | $n$ dimensional vectors space over $\mathbb{G}$ |
| $\mathbb{Z}_p^n$ | $n$ dimensional vectors space over $\mathbb{Z}$ |
| $\mathbb{Z}_p^*$ | $\mathbb{Z}_p \setminus \{0\}$ |
| $\boldsymbol{g}$ | $(g_1, \ldots, g_n) \in \mathbb{G}^n$ |
| $\boldsymbol{g}^a$ | $(g_1^a, \ldots, g_n^a) \in \mathbb{G}^n$ |
| $\boldsymbol{a}$ | $(a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ |
| $\boldsymbol{g}^{\boldsymbol{a}}$ | $\prod_n^{i=1} g_i^{a_i} \in \mathbb{G}^n$ |
| $\boldsymbol{a} + \boldsymbol{b}$ | $(a_1 + b_1, \ldots, a_n + b_n) \in \mathbb{Z}_p^n$ |
| $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ | $\sum_{i=1}^n a_i \cdot b_i \in \mathbb{Z}_p$ |
| $\boldsymbol{a} \circ \boldsymbol{b}$ | $(a_1 \cdot b_1, \ldots, a_n \cdot b_n) \in \mathbb{Z}_p^n$ |
| $a_i$ | $i$ th element in $\boldsymbol{a}$ |
| $\boldsymbol{a}_{[i:j]}$ | $(a_i, \ldots, a_j)$ |
| $a \vert\vert b$ | Concatenation of elements |

Suppose Hash() is a collision resistant hash function which maps arbitrary length input to an element in $\mathbb{Z}_p^*$ in a deterministic manner.

### C. Preliminaries and their notations

*1) Pedersen Commitment:* Pedersen commitment [24] can hide and commit any value within $\mathbb{Z}_p$. It is perfectly hiding, statistically binding and additively homomorphic. We use the following notations in this paper: (1) given input $x$ and opening $r$, $C_x = \mathrm{commit}(x) = g^x h^r$ denotes the process to commit a value $x$; (2) given a commitment $C_x$, $\mathrm{Cverify}(C_x) \stackrel{?}{=} g^x h^r$ denotes the process to verify a commitment; (3) given two commitments $(C_{x_1}, C_{x_2})$, $C_{x_1} C_{x_2} = g^{x_1+x_2} h^{r_1+r_2}$ denotes the process to additively compute the committed value.

*2) Non-Interactive Zero-Knowledge Proofs:* A zero-knowledge proof of knowledge is introduced by Goldwasser *et al.* [20] which is a two-party protocol between a prover and a verifier. The prover proves its knowledge of some secrets without revealing the secrets to the verifier. Fiat *et al.* [25] showed any interactive zero-knowledge proof can be turned to non-interactive zero-knowledge proof of knowledge

(NIZK). In this paper, we follow the notation introduced by Camenisch and Stadler [26] for various proofs of knowledge of discrete logarithms and proofs of the validity of statement about discrete logarithms. For instance, NIZK$\{(y, \hat{y}; \alpha, \beta) : y = g_0^{\alpha} \wedge \hat{y} = g_1^{\beta}\}$ denotes a *"non-interactive zero-knowledge proof of knowledge of integer $\alpha, \beta$ such that $y = g_0^{\alpha}$ and $\hat{y} = g_1^{\beta}$ holds where $y, \hat{y}$ are public and $g_0, g_1$ are public parameters"*. We refer to [27] for some commonly used building blocks for NIZK, e.g., Chaum-Pedersen proof.

*3) Integrated encryption scheme:* Integrated encryption scheme (IES) is a hybrid encryption scheme that allows one party to send messages to others with known public keys within one round of interaction. The idea of IES is to setup a shared key using the receiver's public key and its own private key, then encrypt the message using symmetric encryption with the shared key and attach its public key with the cipher for the receiver to calculate the shared key. W.L.O.G., we use ECIES [28] due to its efficiency. We use the following notations in this paper: (1) given message $msg$ and shared key $k$, $c = Enc(k; msg)$ denotes the process to encrypt the message and get the cipher $c$; (2) given cipher $c$ and shared key $k$, $msg = Dec(k; c)$ denotes the process to decrypt the cipher and get the message.

*4) Bulletproof:* Bulletproof [29] is the state of the art range proof protocol that proofs a committed value lies in a given range. We only provide a simple and general steps of the scheme and focus on the modifications we made. Please refer to [29] for details of the scheme. We use Fiat-Shamir Heuristic [25] to make the original proof non-interactive. The prover carries out the following proof:

NIZK : $\{(C_v, n; v, r) : C_v = g^v h^r \wedge v \in [0, 2^n - 1]\}$. The prover computes:

$$\begin{aligned}
&\boldsymbol{a}_L \in \{0,1\}^n \quad \text{s.t.} \langle \boldsymbol{a}_L, \mathbf{2}^n \rangle = v \\
&\boldsymbol{a}_R = \boldsymbol{a}_L - \mathbf{1}^n \in \mathbb{Z}_p^n \\
&\alpha \xleftarrow{\$} \mathbb{Z}_p \qquad A = h^{\alpha} \boldsymbol{g}^{\boldsymbol{a}_L} \boldsymbol{h}^{\boldsymbol{a}_R} \in \mathbb{G}.
\end{aligned}$$

The prover randomly and uniformly selects two blinding vectors $\boldsymbol{s}_L, \boldsymbol{s}_R \xleftarrow{\$} \mathbb{Z}_p^n$ as well as random number $\rho \xleftarrow{\$} \mathbb{Z}_p$ and computes:

$$\begin{aligned}
&S = h^{\rho} \boldsymbol{g}^{\boldsymbol{s}_L} \boldsymbol{h}^{\boldsymbol{s}_R} \in \mathbb{G} \\
&y = \text{Hash}(A||S) \qquad z = \text{Hash}(A||S||y) \\
&x = \text{Hash}(A||S||y||z) \\
&\boldsymbol{l} = l(x) = \boldsymbol{a}_L - z \cdot \mathbf{1}^n + \boldsymbol{s}_L \cdot x \in \mathbb{Z}_p^n \\
&\boldsymbol{r} = r(x) = \boldsymbol{y}^n \circ (\boldsymbol{a}_R + z \cdot \mathbf{1}^n + \boldsymbol{s}_R \cdot x) + z^2 \cdot \mathbf{2}^n \in \mathbb{Z}_p^n
\end{aligned}$$

The prover further sends $A, S, \boldsymbol{l}, \boldsymbol{r}$ with other parameters to the verifier as the proof [4] and we refer to [29] for verification process and other details.

*5) Verifiable Shuffle of Commitments:* Bayer and Groth [30] provided an efficient verifiable shuffle of ElGamal ciphertexts which are multiplicative homomorphic. The idea of Bayer and Groth's algorithm is to commit to the permutation with known random numbers, prove the validity of the permutation and prove that the same permutation has been applied to the input ciphertexts. We instantiate a modified argument of verifiable shuffle of Pedersen commitments which are addictive homomorphic using [30] and further provide a way for regulation in the scenario of blockchain. Please refer to [30] for details of the scheme.

Suppose $\pi()$ is a permutation between input commitments and output commitments. The argument proves knowledge of a permutation $\pi()$ such that given randomness $\{\rho_i\}_{i=1}^k$, commitments $\{C_1, \ldots, C_k\}, \{C'_1, \ldots, C'_k\}$ we have $\{C'_i = C_{\pi(i)} h^{\rho_{\pi(i)}}\}_{i=1}^k$.

We use Fiat-Shamir Heuristic to make the original proof non-interactive. The prover carries out the following proof:

NIZK : $\{(\boldsymbol{C}, \boldsymbol{C'}; \pi(), \boldsymbol{\rho}) : \{C'_i = C_{\pi(i)} h^{\rho_{\pi(i)}}\}_{i=1}^k\}$.

Suppose $k = u \cdot v$. The prover randomly and uniformly selects $\pi()$ and $u$ random numbers $\boldsymbol{r} \xleftarrow{\$} \mathbb{Z}_p^u$ and computes:

$$\begin{aligned}
&\{C'_i = C_{\pi(i)} h^{\rho_{\pi(i)}}\}_{i=1}^k \in \mathbb{G} \\
&\boldsymbol{a} = \{\pi(i)\}_{i=1}^k \in \mathbb{Z}_p^k \\
&\boldsymbol{C}_A = (\boldsymbol{g}^{\boldsymbol{a}[\mathbf{1:v}]} h^{\boldsymbol{r_1}}, \ldots, \boldsymbol{g}^{\boldsymbol{a}[(\boldsymbol{u-1})\boldsymbol{v}+\mathbf{1}:\boldsymbol{uv}]} h^{\boldsymbol{r_u}}) \in \mathbb{G}^u \\
&x = \text{Hash}(C_{A_1} || \ldots || C_{A_u}).
\end{aligned}$$

The prover randomly and uniformly selects $u$ random numbers $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_p^u$ and computes:

$$\begin{aligned}
&\boldsymbol{b} = \{x^{\pi(i)}\}_{i=1}^k \\
&\boldsymbol{C}_B = (\boldsymbol{g}^{\boldsymbol{b}[\mathbf{1:v}]} h^{\boldsymbol{s_1}}, \ldots, \boldsymbol{g}^{\boldsymbol{b}[(\boldsymbol{u-1})\boldsymbol{v}+\mathbf{1}:\boldsymbol{uv}]} h^{\boldsymbol{s_u}}) \in \mathbb{G}^u \\
&\rho = -\langle \boldsymbol{\rho}, \boldsymbol{b} \rangle \in \mathbb{Z}_p.
\end{aligned}$$

The prover engages in the product argument and the multi-exponentiation argument to finish the proof.

The prover sends $\boldsymbol{C}, \boldsymbol{C'}, \boldsymbol{C}_A, \boldsymbol{C}_B$ together with the product argument and multi-exponentiation proof.

The verifier checks $\boldsymbol{C}, \boldsymbol{C'} \in \mathbb{G}^k$, $\boldsymbol{C}_A, \boldsymbol{C}_B \in \mathbb{G}^u$, calculates the intermediate parameters and verifies the above two arguments.

## VI. Algorithmic Components of WaterCarver

The following algorithms are described to implement the anonymous confidential transactions and regulations.

**System setup.** The algorithm $\mathrm{Setup}$ outputs a list of public parameters $pp$:

$\mathrm{Setup}$
Input: security parameter $1^{\lambda}$
Output: public parameters $pp$

This one-time-only algorithm is executed in a trusted way and the $pp$ is published to all the parties. No global secrets or trapdoors are kept. No other algorithms are executed in a trusted way.

**Key generation.** This algorithm $\mathrm{KeyGen}$ outputs a public key pair $(pk, sk)$ for users or SSPs. All public keys are

[4]The proof can be further improved to make the proof length shorter.

assumed to be kept available and pre-distributed to all users and SSPs.

KeyGen
Input: public parameter $pp$
Output: public key pair $(pk, sk)$

**Convert plaintext balance to confidential balance.** This algorithm $\mathrm{Conv_{p2c}}$ updates one user's account by converting plaintext balance to confidential balance.

$\mathrm{Conv_{p2c}}$
Input:
- public parameters $pp$
- public key pair $(pk, sk)$
- plaintext account balance $balance$
- confidential account balance $C_s$
- convert amount $x$

Output:
- updated plaintext account balance $balance'$
- updated confidential account balance $C_s'$

This process allows any users to hide their account balance by converting plaintext balance to confidential balance within the same amount. Every user's initial confidential account balance is 0. However, this algorithm cannot protect the privacy solely because of the equality between confidential balance and converted plaintext balance. Users need to send or receive any confidential transaction to effectively conceal the confidential account balance to others. A record of this process should also be preserved on-chain for verification.

**Convert confidential balance to plaintext balance.** This algorithm $\mathrm{Conv_{c2p}}$ updates one user's account by converting confidential balance to plaintext balance. This algorithm is the reverse process of $\mathrm{Conv_{p2c}}$. To guarantee that the updated confidential account balance is valid, a range proof $\pi$ is provided. Note that the range proof can be omitted if the user chooses to directly reveal the confidential account balance.

$\mathrm{Conv_{c2p}}$
Input:
- public parameters $pp$
- public key pair $(pk, sk)$
- plaintext account balance $balance$
- confidential account balance $C_s$
- convert amount $x$
- proof $\pi$

Output:
- updated plaintext account balance $balance'$
- updated confidential account balance $C_s'$

$\mathrm{tx_{send}}$ **generation.** This algorithm $\mathrm{tx_{send}Gen}$ outputs a valid $\mathrm{tx_{send}}$ for block $n$ from a sender Alice to a receiver Bob, together with its proof $\pi$ and additional information for Bob.

$\mathrm{tx_{send}Gen}$
Input:
- public parameters $pp$
- Alice's secret key $sk_A$
- Bob's public key $pk_B$
- Alice's account balance $C_s^{A_0}$
- transaction amount $x$
- strategy trie in previous block $H_s^{n-1}$
- regulation parameter $rp$
- Regulator's signature $sig_R$

Output:
- Alice's updated account balance $C_s^{A_1}$
- $\mathrm{tx_{send}}$ which includes
  - transaction amount's commitment $C_x$
  - proof $\pi$
  - random IES key
  - IES encrypted message $msg$
  - Gas
  - Regulator's signature $sig_R$
  - Alice's signature $sig_A$

The proof $\pi$ in this algorithm is a library of the range proof for $\mathrm{tx_{send}}$, the range proof for Alice's updated account balance[5] $C_s^{A_1}$, and some other NIZKs that will be introduced later. The IES encrypted message, which is encrypted using the random one-time shared key, informs Bob of the spending of $\mathrm{tx_{send}}$ to receive the transaction. Note that in a regulation-free scenario, the strategy trie in the previous block and the regulator's pre-signed parameter can be omitted from the input list. Accordingly, the regulator's signature can be omitted from the output list.

$\mathrm{tx_{send}}$ **verification.** This algorithm $\mathrm{tx_{send}Verify}$ outputs 1 iff the input $\mathrm{tx_{send}}$ is valid.

$\mathrm{tx_{send}Verify}$
Input:
- public parameters $pp$
- $\mathrm{tx_{send}}$
- Alice's public key $pk_A$
- previous block $blk_{n-1}$

Output:
- 0 or 1

$\mathrm{tx_{recv}}$ **generation.** This algorithm $\mathrm{tx_{recv}Gen}$ outputs a valid $\mathrm{tx_{recv}}$. It deletes one $\mathrm{tx_{send}}$ from the UTXO pool, receives the $\mathrm{tx_{send}}$'s amount and adds the amount to Bob's account balance. Once this transaction is on-chain, the UTXO pool should be updated.

There is no regulation for this algorithm, but the information contained can be derived by regulating the related $\mathrm{tx_{send}}$ and $\mathrm{tx_{shuffle}}$.

[5]In the rest of the paper, we use account balance to denote confidential account balance if not specified.

$\mathrm{tx}_{\mathrm{recv}}\mathrm{Gen}$

Input:
- public parameters $pp$
- Bob's secret key $sk_B$
- $\mathrm{tx}_{\mathrm{send}}$
- Bob's knowledge regarding $\mathrm{tx}_{\mathrm{send}}$
- Bob's account balance $C_s^{B_0}$

Output:
- Bob's updated account balance $C_s^{B_1}$
- $\mathrm{tx}_{\mathrm{recv}}$ which includes
  - proof $\pi$
  - Bob's signature

Besides, Bob's signature is essential for $\mathrm{tx}_{\mathrm{recv}}$. Although Bob has proved his knowledge to receive the $\mathrm{tx}_{\mathrm{send}}$, a malicious sender can also update Bob's account to an unknown state if Bob's signature is not required.

$\mathrm{tx}_{\mathrm{recv}}$ **verification.** This algorithm $\mathrm{tx}_{\mathrm{recv}}\mathrm{Verify}$ outputs 1 iff the input $\mathrm{tx}_{\mathrm{recv}}$ is valid.

$\mathrm{tx}_{\mathrm{recv}}\mathrm{Verify}$

Input:
- public parameters $pp$
- $\mathrm{tx}_{\mathrm{recv}}$
- previous block $blk_{n-1}$

Output:
- 0 or 1

$\mathrm{tx}_{\mathrm{shuffle}}$ **generation.** This algorithm $\mathrm{tx}_{\mathrm{shuffle}}\mathrm{Gen}$ inputs $k$ $\mathrm{tx}_{\mathrm{send}}$ and outputs a valid shuffle of them for block $blk_n$. Once this transaction is on-chain, the UTXO pool should be updated.

$\mathrm{tx}_{\mathrm{shuffle}}\mathrm{Gen}$

Input:
- public parameters $pp$
- $k$ $\mathrm{tx}_{\mathrm{send}}$ with $\mathrm{Gas}$
- SSP's knowledge regarding each $\mathrm{tx}_{\mathrm{send}}$
- header of previous block $blk_{n-1}$
- regulation parameter $rp$
- Regulator's signature $sig_R$

Output:
- $\mathrm{tx}_{\mathrm{shuffle}}$ which includes
  - $k$ $\mathrm{tx}'_{\mathrm{send}}$ with updated $\mathrm{Gas}'$
  - proof $\pi$
  - Regulator's signature $sig_R$

The previous block $blk_{n-1}$ in this algorithm is used to acquire the current regulation rule so that SSPs can get the pre-signed random number from the correct regulator. The gas cost for shuffle should be universally equal: the $\mathrm{Gas}$ in each input should equal with each other or only a limited number of $\mathrm{Gas}$ values are different from others. This is to guarantee that no adversary can find the correlation between input $\mathrm{tx}_{\mathrm{send}}$ and output $\mathrm{tx}_{\mathrm{send}}$ from the gas system.

The SSP's knowledge of the $k$ $\mathrm{tx}_{\mathrm{send}}$ is obtained from their receivers directly off-chain. We will introduce another alternative algorithm later so that some SSPs can obtain one $\mathrm{tx}_{\mathrm{send}}$'s knowledge based on on-chain information solely.

Similarly, in the scenario without regulation, the header of the previous block and regulator's pre-signed random number can be omitted from the input list. Accordingly, the regulator's signature can be omitted from the output list.

SSPs may also need to provide their plaintext account addresses to receive gas. Besides this, we can see that this algorithm does not require any information regarding the SSP itself, such that no authentication is needed for any SSP and the role of SSPs can be played by any user or peer.

$\mathrm{tx}_{\mathrm{shuffle}}$ **verification.** This algorithm $\mathrm{tx}_{\mathrm{shuffle}}\mathrm{Verify}$ outputs 1 iff the input $\mathrm{tx}_{\mathrm{shuffle}}$ is valid.

$\mathrm{tx}_{\mathrm{shuffle}}\mathrm{Verify}$

Input:
- public parameters $pp$
- $\mathrm{tx}_{\mathrm{shuffle}}$
- previous block $blk_{n-1}$

Output:
- 0 or 1

**Block generation.** This algorithm $\mathrm{BlkGen}$ outputs a block for peers to validate.

$\mathrm{BlkGen}$

Input:
- public parameters $pp$
- previous block $blk_{n-1}$
- a set of $\mathrm{tx}_{\mathrm{send}}$
- a set of $\mathrm{tx}_{\mathrm{recv}}$
- a set of $\mathrm{tx}_{\mathrm{shuffle}}$
- a strategy trie $H_s$

Output:
- $blk_n$

When a miner executes this algorithm, it first checks the correctness of all transactions and then the potential conflicts between transactions (i.e., multiple users can legally receive the same UTXO as long as they have the knowledge). The miner decides a particular user to receive the UTXO on a first-come, first-served basis.

Besides, a special kind of double spend attack needs to be detected. Considering that Bob owns one UTXO, he can perform two valid actions: receive it now or let an SSP shuffle it and receive the shuffled UTXO later. If in one block without double spend detection, Bob can actually duplicate this UTXO by receiving and shuffling the same UTXO. In general, the double spend detection ensures that one UTXO is deleted from the transaction pool for only once.

**Block verification.** This algorithm $\mathrm{BlkVerify}$ outputs 1 iff the input block is valid.

BlkVerify
Input:
- public parameters $pp$
- $blk_n$
- previous block $blk_{n-1}$
Output:
- 0 or 1

**Regulation setup.** This algorithm ReguSetup outputs parameters for users' regulation.

ReguSetup
Input:
- public parameters $pp$
- transaction type for regulation $type$
- regulator's database $db$
- regulator's secret key $sk_R$
Output:
- regulator's signature $sig_R$
- regulation parameters $rp$
- regulator's updated database $db'$

The transaction type in this algorithm is either $\mathrm{tx}_{\mathrm{send}}$ or $\mathrm{tx}_{\mathrm{shuffle}}$. The algorithm outputs different regulation parameters based on different transaction types. The regulator stores the $rp$ together with the signature in a database. W.L.O.G., the regulator stores them in a key-value database with signature as the key and $rp$ as the value.

$\mathrm{tx}_{\mathrm{send}}$ **regulation.** This algorithm $\mathrm{tx}_{\mathrm{send}}\mathrm{Regu}$ outputs the amount of one $\mathrm{tx}_{\mathrm{send}}$.

$\mathrm{tx}_{\mathrm{send}}\mathrm{Regu}$
Input:
- regulator's database $db$
- regulator's signature $sig_R$
- $\mathrm{tx}_{\mathrm{send}}$
Output:
- transaction amount $x'$

This algorithm first retrieves the $rp$ based on the signature. Then, this algorithm calculates the transaction amount based on $rp$ and $\pi$ within $\mathrm{tx}_{\mathrm{send}}$. This implies that this algorithm can only regulate the $\mathrm{tx}_{\mathrm{send}}$ sent on-chain by its sender and cannot regulate the output of any $\mathrm{tx}_{\mathrm{shuffle}}$ lacking $\pi$ or regulator's signature. Besides, this algorithm's regulation scope can be further subdivided. Since one $\mathrm{tx}_{\mathrm{send}}$ contains two range proof, one for transaction amount and the other for the updated account balance. This algorithm can actually regulate both of the two numbers. For simplicity, we describe the regulation of the transaction amount solely if not specially specified.

$\mathrm{tx}_{\mathrm{shuffle}}$ **regulation.** This algorithm $\mathrm{tx}_{\mathrm{shuffle}}\mathrm{Regu}$ outputs the permutation used within the $\mathrm{tx}_{\mathrm{shuffle}}$.

Similarly, this algorithm first retrieves the $rp$ based on the signature. Then, this algorithm calculates the permutation used in the shuffle transaction based on the $rp$ and $\pi$ within $\mathrm{tx}_{\mathrm{shuffle}}$.

$\mathrm{tx}_{\mathrm{shuffle}}\mathrm{Regu}$
Input:
- regulator's database $db$
- regulator's signature $sig_R$
- $\mathrm{tx}_{\mathrm{shuffle}}$
Output:
- permutation in $\mathrm{tx}_{\mathrm{shuffle}}$

## VII. WaterCarver System

Based on preliminaries and algorithmic components, we present an anonymous confidential blockchain system with regulation, called WaterCarver. Before providing our concrete construction, we first present two regulation sub-systems.

### A. Bulletproof Regulation Sub-system

We provide a simple yet effective prototype for Bulletproof regulation in the scenario of blockchain. There are three kinds of entities in this sub-system: prover, verifier, and regulator. The prover puts its proof in a block being proposed. The verifier verifies the correctness of the proof based on the information in the block. Both the prover and the verifier knows the public key of the regulator.

Before constructing a proof, the prover requests from the regulator for two blinding vectors as well as a random number $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$ and a signature $sig_R$ on the corresponding $S$ which is calculated as $h^{\rho}\boldsymbol{g}^{\boldsymbol{s}_L}\boldsymbol{h}^{\boldsymbol{s}_R}$.

The regulator randomly and uniformly selects $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$ and signs on $S$ using its private signing key. Without losing of generality, the regulator stores the $\boldsymbol{s}_L$ and the corresponding signature locally in a key-value database with signature as key and $\boldsymbol{s}_L$ as value. Note that the requesting process should be encrypted.

The prover makes the proof using the retrieved $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$. It sends the proof together with regulator's signature and discards the used blinding vectors as well as random numbers. The verifier verifies the NIZK together with the signature.

When regulation is needed, the regulator checks the correctness of the proof and retrieves the corresponding $\boldsymbol{s}_L$ with signature in its database. Based on $\{A, S, T_1, T_2, \boldsymbol{l}\}$, the regulator computes:

$$y = \mathrm{Hash}(A||S) \qquad z = \mathrm{Hash}(A||S||y)$$
$$x = \mathrm{Hash}(T_1||T_2)$$
$$\boldsymbol{a}_L = \boldsymbol{l} + z \cdot \mathbf{1}^n - \boldsymbol{s}_L \cdot x$$

where $\boldsymbol{a}_L$ is the binary form of the committed value $v$.

For multi-regulator systems that regulate the same information, the regulators need to agree on one set of $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$. Then, they sign on the corresponding $S$ with their own private signing keys, store $\boldsymbol{s}_L$ in their own databases with their own signatures and exchange signatures with each other. Finally, when one prover requests $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$ to any one of the

regulators, that regulator responds with $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$ and all the signatures on $\boldsymbol{s}_L$ after noticing other regulators that this set of $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$ and corresponding signatures are voided for future use.

Using this regulation method, the prover does not directly communicate with the verifier. So, the regulator can make sure it receives the same proof as the verifier without additional communication overhead. Moreover, it is impossible for a single prover to circumvent regulation if the verifier requires the regulator's signature. Besides, there will be no divergence between the prover and the verifier even if the regulation strategy or the regulator's public key is updating periodically as long as they all follow the regulation rule in the previous block.

Essential improvements are needed for the above prototype if the number of regulators goes larger or the network condition gets worsen.

- Multi-signatures [31] are needed to shrink the signature size for lower bandwidth and storage overhead.
- The regulators need to precompute a large pool of $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$ and schedule them by block number. Each regulator only uses a mutually-exclusive subset of the pool as response to provers. This is to avoid duplicately providing one specific set of $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$ to multiple provers due to network delay or partition. Otherwise, the two provers will find out they used the same $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$ because of the same regulation signature and they are able to calculate each other's transaction amount based on the knowledge of $\boldsymbol{s}_L$.

**Theorem 1.** *The proposed Bulletproof regulation method achieves perfect forward secrecy.*

*Proof:* The long-term key is merely used for signing $S$, which is a public parameter in the proof. Even if an adversary gains access to the signing key, she cannot compute the previously selected random numbers which have no correlation with the signing key. ∎

### B. Verifiable Shuffle Regulation Sub-system

The regulation method for verifiable shuffle is the same as Bulletproof, except that what the verifier verifies are the parameters of verifiable shuffle.

Before making a proof, the prover queries the regulator for $\{\pi(), \boldsymbol{r}\}$ and a signature on the corresponding $x$ which is deterministic based on $\{\pi(), \boldsymbol{r}\}$. The regulator randomly and uniformly selects $\{\pi(), \boldsymbol{r}\}$ and signs on the corresponding $x$ with its private signing key. Then the regulator stores the $\pi$ and the corresponding signature locally. Without losing of generality, the regulator stores them in a key-value database with signature as key and $\pi()$ as value. Note that the requesting process should be encrypted.

The prover makes the proof using the retrieved $\{\pi(), \boldsymbol{r}\}$ instead of selecting by itself. The prover sends the proof together with the regulator's signature and discards the used $\pi()$ as well as random numbers. The verifier verifies the NIZK together with the signature.

When regulation is needed, the regulator checks the correctness of the proof and retrieves the corresponding $\pi()$ using the signature as the key in its database.

The method of multi-regulator regulation directly follows the Bulletproof regulation part.

**Theorem 2.** *The proposed verifiable shuffle regulation method achieves perfect forward secrecy.*

Similar to Bulletproof regulation, the verifiable shuffle regulation shows the characteristics of perfect forward secrecy.

### C. $\mathrm{tx}_{\mathrm{send}}$ Construction

The system setup follows the requirements in [12], [29], [30]. We refers to [12] for key generation and general signatures. We construct the confidential account balance using Pedersen commitment. The update of confidential account balance utilises the additive homomorphic property of Pedersen commitment.

Suppose Alice sends $x$ to Bob through $\mathrm{tx}_{\mathrm{send}}$. She first synchronizes previous block's strategy trie $H_s^{n-1}$ to get regulation information and requests the regulator for a set of regulation parameter $rp$ for Bulletproof: $\{\boldsymbol{s}_L, \boldsymbol{s}_R, \rho\}$ and a signature $sig_R$ on the corresponding $S$.

Then, it carries out the following proof:

$$\begin{aligned}\mathrm{NIZK}_1 : \{(C_x, C_{A_0}, C_{A_1}, n; x, r_0, A_1, r_A) : C_x = g^x h^{r_0} \wedge \\ C_{A_1} = C_{A_0}/C_x = g^{A_1} h^{r_A} \wedge C_x = g^x h^{r_0} \wedge x \in [0, 2^n - 1] \\ \wedge A_1 \in [0, 2^n - 1]\}.\end{aligned}$$

$\mathrm{NIZK}_1$ can be implemented using two individual Bulletproofs or one aggregated Bulletproof which is suggested due to its efficiency. Alice uses the $rp$ for the Bulletproof on the transaction amount and uses its random number for her updated account balance proof. Note that regulating the updated account balance is also possible, we only take the amount-only regulation for example.

Given Bob's public key as $pk_B$, Alice uses ECIES to encrypt the spending of $C_x$. More specifically, she generates a random one-time key pair $(pk_{rnd}, sk_{rnd})$ where $pk_{rnd} = g^{sk_{rnd}}$ and calculates the shared key as $pk_B^{sk_{rnd}}$. Then she gets encrypted message $msg$:

$$msg = Enc(pk_B^{sk_{rnd}}, x||r_0).$$

Finally, Alice outputs the $\mathrm{tx}_{\mathrm{send}}$ which contains $\mathrm{NIZK}_1$, ECIES encrypted message $msg$ for Bob, one-time public key $pk_{rnd}$, an appropriate Gas and the regulator's signature $sig_R$. She further signs on the $\mathrm{tx}_{\mathrm{send}}$ using her private signing key and sends the signed $\mathrm{tx}_{\mathrm{send}}$ to miners.

### D. $\mathrm{tx}_{\mathrm{shuffle}}$ Construction

The presence of transactions to Bob is either notified by Alice through private communication or checked by Bob via calculating the shared key and decrypt the message. After that, Bob gains the spending of the $\mathrm{tx}_{\mathrm{send}}$ which is $x$ and $r_0$ .

Bob selects one SSP he trusts, e.g., $SSP_1$. He selects a random number $r_1$ and computes the updated $\mathrm{Gas}'$, which is $\mathrm{Gas}$ minus the unit price of one round of shuffle. Then, he carries out the following proof:

$$\begin{aligned}\text{NIZK}_2 : \{(C'_x, C_\alpha, \mathrm{Gas}'; x, r_0, r_1) : C'_x = g^x h^{r_0+r_1} \wedge \\ \alpha = \text{Hash}(C'_x || \mathrm{Gas}') \wedge C_\alpha = g^\alpha h^{r_0+r_1}\}.\end{aligned}$$

Bob sends $(C_x, C'_x, C_\alpha, \mathrm{Gas}', \text{NIZK}_2)$ to $SSP_1$ as the shuffle request through private communication.

When enough shuffle request are accumulated[6], $SSP_1$ first synchronizes previous block's strategy trie $H_s^{n-1}$ to get regulator information and requests the regulator for a set of regulation parameters $rp$ for verifiable shuffle: $\{\pi(), \boldsymbol{r}\}$ and a signature $sig_R$ on the corresponding $x$.

Then, he carries out the following proof:

$$\begin{aligned}\text{NIZK}_3 : \{(\boldsymbol{C_x}, \boldsymbol{C'_x}, \mathbf{Gas'}; \pi(), \boldsymbol{\rho}, \boldsymbol{\beta}, \boldsymbol{\gamma}) : \\ \{C'_i = C_{\pi(i)} h^{\rho_{\pi(i)}}\}_{i=1}^k \wedge C'_{x_i} = g^\beta h^\gamma \wedge \\ \alpha_i = \text{Hash}(C'_{x_i} || \mathrm{Gas}'_i) \wedge C_{\alpha_i} = g^{\alpha_i} h^\gamma\}.\end{aligned}$$

Note that $SSP_1$ does not have knowledge of $\boldsymbol{\beta}, \boldsymbol{\gamma}$. It constructs $\text{NIZK}_3$ by conducting a verifiable shuffle of $\boldsymbol{C}_x$ using the regulator's permutation and each $C_{x_i}$ owner's random number, then binds each $C'_{x_i}$ with their owner's $\text{NIZK}_2$.

Finally, $SSP_1$ outputs the $\mathrm{tx}_{\mathrm{send}}$ which contains $\boldsymbol{C}_x, \boldsymbol{C'_x}, \mathbf{Gas'}$, $\text{NIZK}_3$ and one account address to receive gas. Once this $\mathrm{tx}_{\mathrm{send}}$ is on-chain, the transaction pool is updated simultaneously which means that Bob does not own $C_x$ any more but a new UTXO $C'_x$.

Bob can further shuffle the UTXO $C'_x$ with other SSPs as long as the gas is sufficient. We further provided another construction at the end of this section, which allows users to shuffle UTXOs non-interactively with SSPs but requires more storage as a trade-off.

The gas is bounded with each UTXO during the shuffle. If there is a clear map between input UTXOs' gas and output UTXOs' gas, then the permutation can be inferred. Therefore, this construction requires that the gas cost of one round of shuffle should be universally equal and only UTXOs with the same input gas can be shuffled together.

### *E.* $\mathrm{tx}_{\mathrm{recv}}$ *Construction*

When Bob decides to receive $C'_x$, he generates a simple proof of knowledge of the spending:

$$\text{NIZK}_4 : \{(C'_x; x, r') : C'_x = g^x h^{r'}\}.$$

He sends $\mathrm{tx}_{\mathrm{recv}}$ which contains $C'_x, \text{NIZK}_4$ and Bob's signature on-chain to receive $C'_x$ into his account balance. Since Bob has proved his knowledge against $C'_x$ and signed his proof, his account balance can be directly updated by adding $C'_x$ using addictive homomorphic property of Pedersen commitment.

[6]The shuffle request threshold number is determined by users and the SSP dynamically.

### *F. Non-interactive WaterCarver*

Here we provide a non-interactive variation of WaterCarver where users do not need to interact directly with SSPs. This method clearly improves the usability of the system but requires more storage on-chain as a trade-off. The idea of this variation is to piggyback the shuffle request and information needed in an additional field $\mathrm{Addi}$ in $\mathrm{tx}_{\mathrm{send}}$.

Suppose Alice sends $x$ to Bob through $\mathrm{tx}_{\mathrm{send}}$. Unlike selecting SSPs by the receiver, Alice selects SSPs she trusts, e.g., she selects $SSP_1$ to shuffle for the first round and $SSP_2$ to shuffle for the second round. She randomly selects two random numbers $r_1, r_2$ and computes the expected gas $\mathrm{Gas}', \mathrm{Gas}''$ for the two rounds respectively. She further generates three random one-time key pair $(pk^i_{rnd}, sk^i_{rnd}), i \in \{1, 2, 3\}$ where $pk^i_{rnd} = g^{sk^i_{rnd}}$ and calculates the shared keys. For simplicity, we let $key^B$ to represent the shared key with Bob and let $key^1, key^2$ to represent the shared key with $SSP_1, SSP_2$ respectively.

Alice first calculates the $\{C'_x = g^x h^{r_0+r_1}, \mathrm{Gas}'\}$ which is the UTXO after the first round of shuffle. Then she calculates $\{C''_x = g^x h^{r_0+r_1+r_2}, \mathrm{Gas}''\}$ which is the UTXO after the second round of shuffle. She carries the following proof:

$$\begin{aligned}\text{NIZK}''_2 : \{(C''_x, C''_\alpha, \mathrm{Gas}''; x, r_0 + r_1 + r_2) : C''_x = g^x h^{r_0+r_1+r_2} \\ \wedge \alpha'' = \text{Hash}(C''_x || \mathrm{Gas}'') \wedge C''_\alpha = g^\alpha h^{r_0+r_1+r_2}\}.\end{aligned}$$

Alice calculates the message for $SSP_2$:

$$msg'' = Enc(key^2, C'_x || C''_x || C''_\alpha || \mathrm{Gas}'' || \text{NIZK}''_2)).$$

She uses $msg'' || pk^2_{rnd}$ as the additional information $\mathrm{Addi}'$ for $SSP_1$ to piggyback in its shuffle output. She carries the following proof:

$$\begin{aligned}\text{NIZK}'_2 : \{(C'_x, C'_\alpha, \mathrm{Gas}'; x, r_0 + r_1) : C'_x = g^x h^{r_0+r_1} \\ \wedge \alpha' = \text{Hash}(C'_x || \mathrm{Gas}' || \mathrm{Addi}') \wedge C'_\alpha = g^\alpha h^{r_0+r_1}\}.\end{aligned}$$

Similarity, Alice calculates the message for $SSP_1$:

$$msg' = Enc(key^1, C_x || C'_x || C'_\alpha || \mathrm{Gas}' || \mathrm{Addi}' || \text{NIZK}'_2)).$$

Alice carries out $\text{NIZK}_1$ as described before. She calculates the encrypted message for Bob as

$$msg = Enc(key^B, x || r_0 || r_1 || r_2).$$

Finally, Alice outputs the $\mathrm{tx}_{\mathrm{send}}$ which contains $\text{NIZK}_1$, $msg$ for Bob, $msg'$ for $SSP_1$, one-time public key $key^B, key^1$, Gas, the regulator's signature $sig_R$. She further signs on the $\mathrm{tx}_{\mathrm{send}}$ using her private key and sends the signed $\mathrm{tx}_{\mathrm{send}}$ to miners.

In this variant, shuffles are triggered by the sender. If it did not, the receiver can also use this piggyback method to trigger shuffles for multiple rounds with only one time direct interaction with one SSP.

## VIII. Experiments

Simulations and benchmarks are conducted to show the efficiency of our proposed scheme in different scenarios[7].

### A. Benchmarks

We used the benchmark framework provided by Ethereum [12] in Golang to test the TPS for different transactions. We implemented Bulletproof based on Dero project[8] which is based on Ed25519 curve and further improved it with aggregation optimization and multi-exponentiation optimization. We implemented verifiable shuffle based on the Stadium project [9] provided by [32]. Stadium implemented a verifiable shuffle for ElGamal encryption also based on Ed25519 curve. We modified it to shuffle for Pedersen commitment. Stadium is written in C++, we used cgo as the interaction interface between C++ and Golang. We used Ed25519 curve to implement the NIZKs in this paper.

### B. Hardware Settings

All experiments are conducted on a desktop with 64-bit Win 7 operating system and 16GB RAM. The processor is Intel(R) Core(TM) i7-8700 CPU @ 3.2 GHz with 6 cores. For the simplicity of repetition and verification, we ran our experiments in a docker image within a Ubuntu virtual machine using VMware.

### C. Background Knowledge

Before getting into our experiments, we introduce some background knowledge about Bulletproof and verifiable shuffle implementation and optimization.

In our Bulletproof implementation, three different variants are provided and benchmarked:

- A Bulletproof with a single input and optimized verification speed.
- A Bulletproof optimized using the aggregation optimization [29] and multi-exponentiation optimization [33]. This variant allows inputting multiple values and proving that they are in the same range based on aggregation optimization. Multiple users' proofs can be verified together to speed up the verification process based on multi-exponentiation optimization.
- An optimized Bulletproof with modification to suit regulation. In order to preserve the regulated value in the proof, the logarithmic range proof suggested and used in [29] is omitted.

The theoretical size of Bulletproof is $(2\log_2 n + \log_2 m)\mathbb{G} + 5\mathbb{Z}_q$ where $n$ is the range bit length and $m$ is the proof number. However, the theoretical size of Bulletproof with regulation is $(2n + \log_2 m)\mathbb{G} + 1\mathbb{Z}_q$ due to omitting logarithmic range proof. In our implementation, we set $n = 64$ and $m = 2$.

[7]https://github.com/jiajun1992/WaterCarver

[8]https://dero.io

[9]https://github.com/nirvantyagi/stadium

To shuffle $N$ commitments, we need to arrange the $N$ commitments into a $i \times j$ matrix for efficiency. The theoretical size of verifiable shuffle is $11i\mathbb{G} + 5j\mathbb{Z}_q$. According to [30], the most efficient implementation is using multi-exponentiation and a round of the interactive technique. To suit this optimization, we set $i = 16$ and tested conditions for $j \in \{16, 24, 32\}$.

### D. Transaction Size

In Table I, we illustrate the size of different transactions. For $\mathrm{tx}_{\mathrm{send}}$, the size of an optimized Bulletproof is 754B which contributes most of its space. We can see a clear increase in size for regulated Bulletproof. Because logarithmic range proof is omitted, the size of the Bulletproof become linear with $n$. Although the size of one $\mathrm{tx}_{\mathrm{shuffle}}$ apparently overwhelms the size of other transaction, the average size per UTXO for the $\mathrm{tx}_{\mathrm{shuffle}}$ is only around 538 Byte. Compared to $\mathrm{tx}_{\mathrm{shuffle}}$ and $\mathrm{tx}_{\mathrm{send}}$, the size of $\mathrm{tx}_{\mathrm{recv}}$ is relatively small, which is only slightly larger than the size of an original Ethereum transaction.

TABLE I
Size for different transaction

| Size | Descriptions |
|---|---|
| 342 Byte | Original Ethereum |
| 1.06 KB | $\mathrm{tx}_{\mathrm{send}}$ with optimized Bulletproof |
| 8.62 KB | $\mathrm{tx}_{\mathrm{send}}$ with optimized regulated Bulletproof |
| 137.71 KB | $16 \times 16$ $\mathrm{tx}_{\mathrm{shuffle}}$ |
| 440 Byte | $\mathrm{tx}_{\mathrm{recv}}$ |

In Table II, we further compare the size of $\mathrm{tx}_{\mathrm{shuffle}}$ under different matrix settings. We can see that the average size decreases with the increase in the total number of input UTXOs. This is because the size of verifiable shuffle increases sub-linearly.

TABLE II
Size for different transaction

| Total Size | Average Size | Matrix Setting |
|---|---|---|
| 137.71 KB | 538 Byte | $16 * 16$ $\mathrm{tx}_{\mathrm{shuffle}}$ |
| 187.96 KB | 489 Byte | $16 * 24$ $\mathrm{tx}_{\mathrm{shuffle}}$ |
| 238.19 KB | 465 Byte | $16 * 32$ $\mathrm{tx}_{\mathrm{shuffle}}$ |

### E. TPS

In Table III, we illustrate the TPS of different transactions under 6 cores. The TPS of the original Ethereum transaction is also provided for comparison. The experiments of $\mathrm{tx}_{\mathrm{send}}$ and $\mathrm{tx}_{\mathrm{recv}}$ are conducted for 25600 times in average. The experiments for $\mathrm{tx}_{\mathrm{shuffle}}$ containing $m * n = 16 * 16$ individual UTXOs are conducted for 100 times in average. The results show that the optimization of Bulletproof greatly benefits the efficiency of $\mathrm{tx}_{\mathrm{send}}$ verification, while the regulation of Bulletproof degrading the efficiency a little bit due to logarithmic range proof. The total number of UTXOs shuffled is $256 * 11.0 = 2816$.

TABLE III
TPS FOR DIFFERENT TRANSACTION

| TPS | Descriptions |
|---|---|
| 3244 | Original Ethereum |
| 126.8 | $\mathrm{tx}_{\mathrm{send}}$ with naive Bulletproof |
| 1176.1 | $\mathrm{tx}_{\mathrm{send}}$ with optimized Bulletproof |
| 1071.5 | $\mathrm{tx}_{\mathrm{send}}$ with optimized regulated Bulletproof |
| 11.0 | $\mathrm{tx}_{\mathrm{shuffle}}$ |
| 1818.4 | $\mathrm{tx}_{\mathrm{recv}}$ |

We further estimate the overall TPS and upper bound of anonymity for the system under different shuffle rounds. We first calculate the average processing time for each kind of transactions. We use the optimized regulated Bulletproof TPS solely for simplicity.

- $T_{\text{send}}$, time for verifying one $\mathrm{tx}_{\mathrm{send}}$ with optimized regulated Bulletproof: 0.933 ms.
- $T_{\text{shuffle}}$, averaged time for verifying one UTXO within a $\mathrm{tx}_{\mathrm{shuffle}}$ containing 256 UTXOs: 0.355 ms.
- $T_{\text{recv}}$, time for verifying one $\mathrm{tx}_{\mathrm{recv}}$: 0.550 ms.

Then, we estimate the overall TPS in this way:

$$\text{Overall TPS} = 1/(T_{\text{send}} + T_{\text{recv}} + T_{\text{shuffle}} \times \text{shuffle rounds}).$$

In Table IV, we provide the overall TPS together with the estimated time needed under different shuffle rounds. In the fourth column, we provide the estimated transaction pool size under different shuffle rounds. If we naively and optimistically consider that one round of shuffle provides $k$-anonymity, where $k = 256$, two rounds of shuffle provide $k^2$-anonymity, which significantly exceed the upper bound of the pool size. Therefore, the higher level of anonymity is bounded by the TPS instead of the number of shuffle rounds. Note that in real-world scenarios, $k$ is a user-defined parameter. We only use $k = 256$ as an example to explore the limitations and boundaries of this system.

TABLE IV
OVERALL TPS UNDER DIFFERENT SHUFFLE ROUNDS

| Rounds | TPS | Estimated time | Pool Size |
|---|---|---|---|
| 0 | 674.3 | 1.483 ms | 0 |
| 1 | 544.1 | 1.838 ms | 544.1 |
| 2 | 456.0 | 2.193 ms | 912.0 |
| 3 | 392.5 | 2.548 ms | 1177.5 |
| 4 | 344.5 | 2.903 ms | 1382.0 |

We compare TPS under different number of cores in Figure 2. In this figure, we compare three different kinds of transactions: $\mathrm{tx}_{\mathrm{send}}$ using optimized regulated Bulletproof, $\mathrm{tx}_{\mathrm{shuffle}}$ using $16 \times 16$ matrix, and $\mathrm{tx}_{\mathrm{recv}}$. We can see that the TPS for $\mathrm{tx}_{\mathrm{shuffle}}$ grows linearly with the number of cores, while the TPSs for $\mathrm{tx}_{\mathrm{send}}$ and $\mathrm{tx}_{\mathrm{recv}}$ remain steady. This is because Ethereum benchmark framework requires operations that can alter user balance to be processed in sequence while

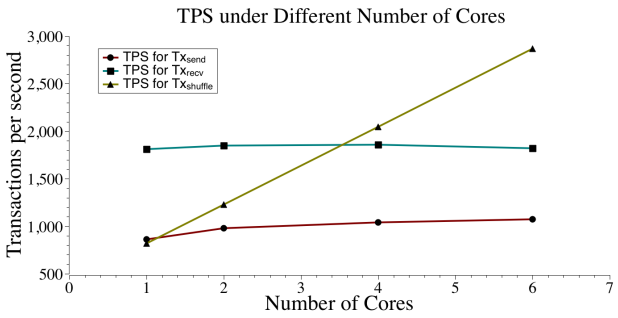


Fig. 2. TPS under different number of cores.

$\mathrm{tx}_{\mathrm{shuffle}}$ is not influential to any user's balance. Therefore, more cores do not improve the TPS of $\mathrm{tx}_{\mathrm{send}}$ and $\mathrm{tx}_{\mathrm{recv}}$ in the experiment. This restriction of Ethereum is reasonable because transactions in Ethereum alter both the sender's and the receiver's account balances and paralleling the account balance updates causes data conflicts and intricate problems. However, unlike the original Ethereum, updates of the sender's and receiver's account balances in the system are separated in two transactions, so $\mathrm{tx}_{\mathrm{send}}$ and $\mathrm{tx}_{\mathrm{recv}}$ are only effective on one users' account balance. As long as we process transactions from the same user in sequence, most transactions in this system can be processed in parallel. Hence, we can foresee that the TPS of this system grows linearly with core numbers in real implementations.

## IX. RELATED WORK

**Privacy-preserving blockchain system.** There are several solid works [15], [34]–[36] solving the confidentiality problem thoroughly. Most of them use commitments to hide the transaction amounts and apply range proofs like [27], [29] to prove the amount is valid.

In Bitcoin and Ethereum, anonymity is only insufficiently ensured by pseudonymity. There are several mixing schemes [37], [38] proposed for better anonymity and most of them achieve $k$-anonymity. CoinJoin [37] is the first coin mixer to shuffle Bitcoins. Its real-world implementation depends on a trusted party to resist attacks which causes users' discontent. Coinshuffle [38] improves CoinJoin in a distributed way with a heavy off-chain computation. Several works [21], [39], [40] further improves coin mixing protocols by depositing coin to a mixer and sending the withdraw information to coin receiver off-chain through private communication. The off-chain communication clearly influences the usability of these schemes. Besides, these mixing schemes only protect the anonymity of transactions where the transaction amount is public or settled.

Monero[10] uses ring signature [15], [35] to achieve $k$-anonymity and uses confidential transaction to achieve confidentiality. Due to the overhead of ring signature grows

[10]https://monero.org/

with $k$, a relative small $k$ is used[11]. Because of the small $k$ and its naive decoy inputs selection method, Monero's $k$-anonymity is vulnerable. Yu *el al.* [41] achieved to identified 5,000 transactions in Monero with inference attack which only observes the decoy inputs selections. Although a sophisticated decoy inputs selection method reduces the chances of being identified in this case, some other methods can also help adversaries to infer [42], [43]. For example, a malicious user can generate lots of transactions or accounts in order to be chosen into some transaction sender's $k$ clocking groups and gain advantage in inference since it knows itself is not the sender. Several works [18], [44] have been proposed to provide regulation or traceability for Monero.

**Verifiable Shuffle.** There are several works [21], [45] used verifiable shuffle as their key components to provide anonymity. Verifiable shuffle proves that one list of hidden values is permuted from the other. It is usually applied in untraceable electronic mail [46] and e-voting [1] where anonymity is required and no single third-party is trusted. There are different kinds of verifiable shuffles in different settings. The majority of works try to prove two lists of multiplicative homomorphic ciphertexts (e.g. ElGamal encryption) have the same plaintexts where the plaintexts are unknown to the prover [30], [47]. In this case, a single prover only has knowledge of the permutation of its own round and by involving multiple rounds of shuffles, no party has knowledge of the overall permutation unless all provers collude with each other. Neff [1] first introduced the K-shuffle which proves two lists of elements $X_1, \ldots, X_N$ and $Y_1, \ldots, Y_N$ in $\mathbb{Z}_q$ have the property that $Y_i^d = X_{\pi(i)}^c$ where $c, d$ are committed. [1] also defines that if the prover has knowledge of $\log_g X_i$ as well as $\log_g Y_i$, the problem is called simple K-shuffle; if not, the problem is called general K-shuffle.

**Concurrent Work.** Zether [13] is state of the art of privacy-preserving technique based on Ethereum. It not only provides confidentiality but also anonymity to account-based transactions in a fully-decentralized way. It is built in a smart contract of Ethereum which clearly makes it easy to be implemented. However, every operation in a smart contract is costly. The high cost of Zether, which can be approximately $\$1.51$ per confidential transaction according to [13], limits its further adoption. In the meantime, our work is a modification of Ethereum. This makes our work harder to be implemented, but more flexible and cost-friendly.

Besides this difference, Zether and our work can be integrated together for better privacy and regulation support. The anonymity of Zether is also $k$-anonymity, like Monero. Each user in Zether makes its own selection of decoy inputs as a cloak. This process can be potentially combined together with our verifiable shuffle for a very large cloaking group. We leave this interesting problem as our future work.

[11] $k$ varies in different transactions, it is mostly set to 7.

## X. Discussions and Conclusion

**Discussions.** Based on the experimental results, the TPS of this system, which is already considerably high, is the bottleneck for better usability and anonymity. To improve the TPS, there are several potential methods. Firstly, batching the Bulletproof using MPC as described in [29]. Secondly, more multi-exponentiation techniques [33], [48] can be applied to further optimize NIZKs. Thirdly, aggregate signature [49]–[51] can be applied to the system for faster signature verification. Fourthly, some common optimizations like faster elliptic curves calculations using GPU [52], [53] or simply involving more cores for verification are clearly beneficial to this system's TPS.

Moreover, changing the assumptions of SSPs to semi-honest can greatly benefit the TPS of $\mathrm{tx}_{\mathrm{shuffle}}$. For now, we bind each UTXO in the $\mathrm{tx}_{\mathrm{shuffle}}$ with two Chaum-Pedersen proofs. One Chaum-Pedersen proof is to show the shuffle is triggered by users with spending of the UTXO. The other is to proof the gas is correctly calculated by the SSP and we only allow input gas to be equal. If we assume SSPs correctly calculate the gas cost, we can redesign the $\mathrm{tx}_{\mathrm{shuffle}}$ as follows: SSPs deduct gas of each UTXOs randomly following the principle of differential privacy [54], [55]. Users only need to bind one Chaum-Pedersen proof with each UTXO to show that the shuffle is triggered by the UTXO owner. In this way, UTXOs with different gas can be shuffled together and the verification process is faster due to omitting one Chaum-Pedersen proof.

**Conclusion.** Privacy and regulation are the two major challenges of an account-model-based blockchain system. Our proposed scheme, WaterCarver, solves them both in the meantime. In our system, we provide robust anonymity and confidentiality. We also leave space for a flexible and perfect forward secure regulation. Experiments have demonstrated the usability of our system. We expect a much higher TPS in the real implementation with more computing power to meet large-scale adoption needs.

## Acknowledgement

We thank Benedikt Bünz, Cathie Yun for helpful discussions. We thank Nirvan Tyagi for help with experiments. The name of this work comes with an ancient Chinese proverb "Ke Zhou Qiu Jian". We refer to that transactions are submerged and mixed under the water, while their locations are marked with notches in a boat floating on the surface. Later, the transactions are retrieved with the help of the notches.

## References

[1] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM, 2001, pp. 116–125.

[2] T. Moran and M. Naor, "Receipt-free universally-verifiable voting with everlasting privacy," in *Annual International Cryptology Conference*. Springer, 2006, pp. 373–392.

[3] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," *EC*, vol. 99, pp. 129–139, 1999.

[4] Q. Huang, Y. Tao, and F. Wu, "Spring: A strategy-proof and privacy preserving spectrum auction mechanism," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 827–835.

[5] S. Liu, H. Zhu, R. Du, C. Chen, and X. Guan, "Location privacy preserving dynamic spectrum auction in cognitive radio network," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 256–265.

[6] A. Rial and G. Danezis, "Privacy-preserving smart metering," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*. ACM, 2011, pp. 49–60.

[7] A. Rial, G. Danezis, and M. Kohlweiss, "Privacy-preserving smart metering revisited," *International Journal of Information Security*, vol. 17, no. 1, pp. 1–31, 2018.

[8] H. Zhu, X. Lin, M. Shi, P.-H. Ho, and X. Shen, "Ppab: A privacy-preserving authentication and billing architecture for metropolitan area sharing networks," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 5, pp. 2529–2543, 2008.

[9] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *2010 proceedings ieee infocom*. Ieee, 2010, pp. 1–9.

[10] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *European symposium on research in computer security*. Springer, 2009, pp. 355–370.

[11] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *2010 Proceedings IEEE INFOCOM*. Ieee, 2010, pp. 1–9.

[12] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[13] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world." *IACR Cryptology ePrint Archive*, vol. 2019, p. 191, 2019.

[14] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 839–858.

[15] S.-F. Sun, M. H. Au, J. K. Liu, and T. H. Yuen, "Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 456–474.

[16] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," 2019.

[17] L. Association *et al.*, "Libra white paper," *Internet access:¡ https://libra.org/en-US/whitepaper/¿,[accessed June 19, 2019]*, 2019.

[18] Y. Li, G. Yang, W. Susilo, Y. Yu, M. H. Au, and D. Liu, "Traceable monero: Anonymous cryptocurrency with enhanced accountability," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[19] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," *arXiv preprint arXiv:1903.07602*, 2019.

[20] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989.

[21] I. A. Seres, D. A. Nagy, C. Buckland, and P. Burcsi, "Mixeth: efficient, trustless coin mixing service for ethereum." *IACR Cryptology ePrint Archive*, vol. 2019, p. 341, 2019.

[22] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, 2017.

[23] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[24] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual International Cryptology Conference*. Springer, 1991, pp. 129–140.

[25] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 186–194.

[26] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *Annual International Cryptology Conference*. Springer, 1997, pp. 410–424.

[27] F. Boudot, "Efficient proofs that a committed number lies in an interval," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 431–444.

[28] V. G. Martínez, L. H. Encinas, and C. S. Ávila, "A survey of the elliptic curve integrated encryption scheme," *ratio*, vol. 80, no. 1024, pp. 160–223, 2010.

[29] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 315–334.

[30] S. Bayer and J. Groth, "Efficient zero-knowledge argument for correctness of a shuffle," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 263–280.

[31] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple schnorr multi-signatures with applications to bitcoin," *Designs, Codes and Cryptography*, pp. 1–26, 2018.

[32] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A distributed metadata-private messaging system," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 423–440.

[33] C. H. Lim, "Efficient multi-exponentiation and application to batch verification of digital signatures, 2000," *Manuscript available at http://dasan. sejong. ac. kr/ chlim/pub/multi exp. ps*, 2014.

[34] G. Maxwell, "Confidential transactions," https://people.xiph.org/~greg/confidential_values.txt, 2016.

[35] S. Noether, "Ring signature confidential transactions for monero." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1098, 2015.

[36] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 459–474.

[37] G. Maxwell, "Coinjoin: Bitcoin privacy for the real world," in *Post on Bitcoin forum*, 2013.

[38] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 345–364.

[39] S. Meiklejohn and R. Mercer, "Möbius: Trustless tumbling for transaction privacy," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 2, pp. 105–121, 2018.

[40] O. Shlomovits and I. A. Seres, "Sharelock: Mixing for cryptocurrencies from multiparty ecdsa," *IACR Cryptology ePrint Archive*, 2019.

[41] J. Yu, M. H. A. Au, and P. Esteves-Verissimo, "Re-thinking untraceability in the cryptonote-style blockchain," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE, 2019, pp. 94–9413.

[42] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan *et al.*, "An empirical analysis of traceability in the monero blockchain," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 143–163, 2018.

[43] Z. Yu, M. H. Au, J. Yu, R. Yang, Q. Xu, and W. F. Lau, "New empirical traceability analysis of cryptonote-style blockchains," in *Proceedings of the 23rd International Conference on Financial Cryptography and Data Security (FC)*, 2019.

[44] L. C. Wulu Li, X. Lai, X. Zhang, and J. Xin, "Applications on traceable range proofs from fully regulatable privacy-preserving blockchains."

[45] U. Sarfraz, M. Alam, S. Zeadally, and A. Khan, "Privacy aware iota ledger: Decentralized mixing and unlinkable iota transactions," *Computer Networks*, vol. 148, pp. 361–372, 2019.

[46] R. Rivest, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications*, 1981.

[47] J. Groth and Y. Ishai, "Sub-linear zero-knowledge argument for correctness of a shuffle," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2008, pp. 379–396.

[48] N. Pippenger, "On the evaluation of powers and monomials," *SIAM Journal on Computing*, vol. 9, no. 2, pp. 230–250, 1980.

[49] H. Xiong, Z. Guan, Z. Chen, and F. Li, "An efficient certificateless aggregate signature with constant pairing computations," *Information Sciences*, vol. 219, pp. 225–235, 2013.

[50] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *International*

*Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2003, pp. 416–432.

[51] S. Chatterjee, D. Hankerson, E. Knapp, and A. Menezes, "Comparing two pairing-based aggregate signature schemes," *Designs, Codes and Cryptography*, vol. 55, no. 2-3, pp. 141–167, 2010.

[52] S. Cui, J. Großschädl, Z. Liu, and Q. Xu, "High-speed elliptic curve cryptography on the nvidia gt200 graphics processing unit," in *International Conference on Information Security Practice and Experience*. Springer, 2014, pp. 202–216.

[53] J. W. Bos, "Low-latency elliptic curve scalar multiplication," *International Journal of Parallel Programming*, vol. 40, no. 5, pp. 532–550, 2012.

[54] C. Dwork, "Differential privacy," *Encyclopedia of Cryptography and Security*, pp. 338–340, 2011.

[55] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.