Hurricane Mixer: The Eye in the Storm—Embedding Regulatory Oversight into Cryptocurrency Mixing Services

Zonglun Li, Wangze Ni, Shuhao Zheng, Junliang Luo, Weijie Sun, Lei Chen, *Fellow, IEEE,* Xue Liu, *Fellow, IEEE,* Tianhang Zheng, Zhan Qin, *Senior Member, IEEE,* Kui Ren, *Fellow, IEEE*

***Abstract*—While transaction transparency is fundamental, it introduces privacy vulnerabilities for blockchain users requiring confidentiality. Existing privacy mixers, intended to mitigate the issue by offering obfuscation of transactional links, have been leveraged to evade emerging financial regulations in DeFi and facilitate harmful practices within the community. Regulatory concerns, driven by prosocial intentions, are raised to ensure that mixers are used responsibly complying with regulations. The research challenge is to reconcile privacy with enforceable compliance by providing designated-only transaction traceability, blocking sanctioned actors and preserving honest-user anonymity.**

We tackle this challenge by introducing the *Hurricane Mixer*, the mixer framework that embeds compliance logic without forfeiting privacy of regular transactions. Hurricane comes in two deployable variants: *Cash* for fixed-denomination pools and *UTXO* for arbitrary-amount transfers. Both variants share the key components: a sanction list mechanism that prevents transactions involving sanctioned entities, and a mechanism that allows for possible regulatory access to encrypted transaction details for compliance purposes. We implement the full stack: Gnark Groth-16 circuits for deposit/withdraw proofs, contracts maintaining an on-chain sanction list, and dual public-key encryption for bidirectional tracing. The comprehensive evaluation illustrates the efficacy of Hurricane Mixer in ensuring privacy preservation, regulatory conformity, and cost efficiency. Experiments show that the *Cash* variant is more economical when the payment amount matches the denomination, the *UTXO* variant is better suited for large or fractional payments, and the framework overall sustains competitive gas efficiency without compromising regulator traceability.

***Index Terms*—Data privacy, blockchains, cryptocurrency, smart contracts, internet security, digital policy, regulation**

I. Introduction

THE transparency of blockchain allows users to see any transaction's details, leading to privacy leakage issues. As a result, mixers have become very popular recently as a way to protect user privacy, attracting widespread attention from both academia [1] and industry [2]. Yet mixers remain a double-edged sword: although they cloak legitimate user flows, they also create regulatory blind-spots routinely exploited for money laundering. After the U.S. imposed sanctions on the mixer Tornado Cash and the European Union released the Markets in Crypto-Assets (MiCA) regulation [3], [4], users began seeking regulatory-compliant mixers.

Zonglun Li and Wangze Ni contributed equally to this work. Zonglun Li, Shuhao Zheng, and Junliang Luo are with the School of Computer Science, McGill University, Montreal, H3A 2A7 (e-mail: zonglun.li@mail.mcgill.ca; shuhao.zheng@mail.mcgill.ca; junliang.luo@mail.mcgill.ca).

Wangze Ni, Tianhang Zheng, Zhan Qin, and Kui Ren are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310007 (e-mail: niwangze@zju.edu.cn; zthzheng@zju.edu.cn; qinzhan@zju.edu.cn; kuiren@zju.edu.cn).

Weijie Sun and Lei Chen are with The Hong Kong University of Science and Technology, Hong Kong SAR (e-mail: wsunan@cse.ust.hk; leichen@cse.ust.hk).

Xue Liu is with the School of Computer Science, McGill University, Montreal, H3A 2A7, and with the Mohamed bin Zayed University of Artificial Intelligence (MBZUAI) (e-mail: xueliu@cs.mcgill.ca)

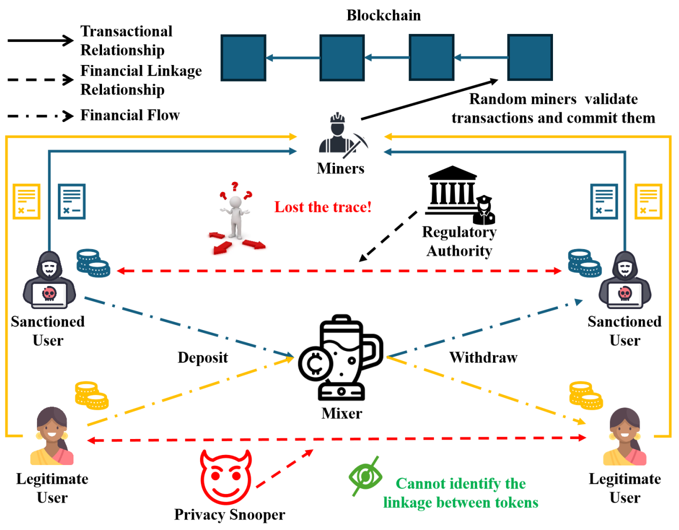


Fig. 1: The Outline of Mixers

Figure 1 shows the outline of mixers. Multiple users first send their original transactions to the mixer. These original transactions are then mixed within the mixer and output. Communication between users and the mixer can occur on-chain or off-chain. On-chain communication is stored on the blockchain by miners as part of transactions and is accessible to any user. The miners who package transactions are randomly selected through mining competitions, a process that is uncontrollable and unpredictable by anyone. Different mixers employ various methods to conceal the association between the mixed output transactions and the original transactions. For instance, in Tornado Cash [2], one of the world's most renowned mixers, users leverage zero-knowledge proofs to prove that the funds they intend to withdraw originated from a specific prior transaction deposited into Tornado Cash, without disclosing the details of that transaction. In other words, the mixer enables users to obscure the source of their funds, preventing fund tracing and achieving privacy protection.

Yet a deployable, practical system of regulation-aware mixer

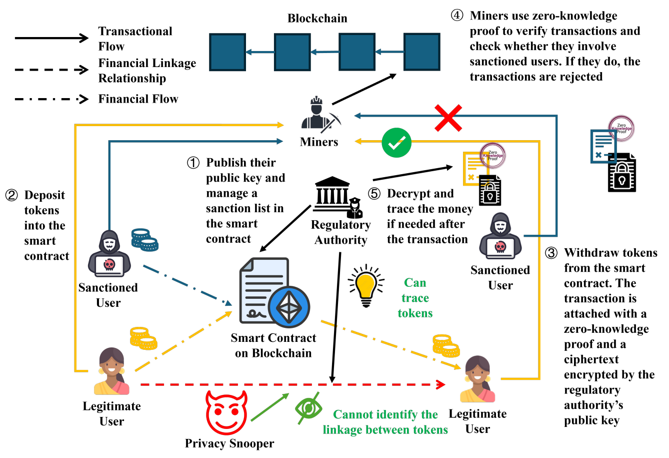

Blockchain
④ Miners use zero-knowledge proof to verify transactions and check whether they involve sanctioned users. If they do, the transactions are rejected
Transactional Flow
Financial Linkage Relationship
Financial Flow
Miners
① Publish their public key and manage a sanction list in the smart contract
② Deposit tokens into the smart contract
Regulatory Authority
⑤ Decrypt and trace the money if needed after the transaction
Sanctioned User
Sanctioned User
③ Withdraw tokens from the smart contract. The transaction is attached with a zero-knowledge proof and a ciphertext encrypted by the regulatory authority's public key
Can trace tokens
Smart Contract on Blockchain
Legitimate User
Privacy Snooper
Cannot identify the linkage between tokens
Legitimate User
Zero Knowledge Proof
Zero Knowledge Proof

Fig. 2: The Outline of Hurricane Mixer.

is still absent. Research has been examining the rationale behind this need [5], [6]. Burleson et al. discussed Tornado Cash as a privacy-enhancing protocol using zero-knowledge proofs to anonymize blockchain transactions, mentioning its legitimate privacy benefits for users while also noting its exploitation by illicit actors, which led to its sanction by the U.S. Treasury's OFAC [6]. In approaching the problem, the authors explored potential improvements, including deposit and withdrawal screening and selective de-anonymization, to balance privacy with regulatory compliance. In practice, effective compliance requires two complementary controls: (i) ex-ante screening that blocks deposits or withdrawals involving sanctioned funds, and (ii) ex-post tracing that reconstructs the flow of suspect funds after they transit the pool. No existing mixer design unifies both defences, chiefly because doing so confronts several system-level challenges:

Challenge 1. In the blockchain system, legitimate users should be able to access their funds, move or transfer their funds freely, and make decisions about their funds without interference or restrictions from other entities. While sanctioned users are restricted with interference or restrictions, particularly when it comes to using certain mixers. This weakening of decentralization must be auditable and occur under well-defined governance under system-level.

Challenge 2. To safeguard user privacy, we must restrict the accessibility of transaction information. However, regulators require users to disclose such information, posing a dilemma. This conflict becomes even more challenging in blockchain systems, where every transaction recorded on the blockchain must be verifiable by any user.

Challenge 3. Fixed-denomination mixers are gas-efficient but force users to split or pad transfers, whereas an denomination-free (UTXO-style) mixer could handle arbitrary amounts at higher per-proof cost. A practical solution must bridge these two transaction (fee) paradigms without sacrificing auditability or privacy.

To address the aforementioned challenges, we introduce *Hurricane Mixers*; its instantiation is sketched in Figure 2. The denomination-free *Hurricane UTXO* variant is introduced later in Section V. In our solution, the regulatory authority publishes a public key and secretly holds the corresponding private key. Additionally, the regulatory authority stores and updates a sanction list in smart contracts. Any transaction touching sanctioned funds will be automatically prohibited.

If a user wants to generate a privacy-preserving transaction, they need to first deposit the transfer amount into a smart contract instead of directly transferring it to the payee's account. After that, the payee uses a transfer transaction to move the money from the smart contract to their own account or to let the smart contract hold it. In the transfer transaction, the payee encrypts the transaction details with the regulatory authority's public key and records them. Furthermore, the payee also attaches a zero-knowledge proof in the transaction to prove ownership of the money used in the transfer, that the transaction participants are not on the sanction list, and that the encrypted information attached to the transaction is correctly generated using the transaction details.

Since any user can execute the on-chain logic that enforces sanction-list checks on every deposit, transfer, and withdrawal, **Challenge 1** (governance-auditable access control) can be resolved. Additionally, zero-knowledge proofs do not reveal the details of transactions and only the regulatory authorities possess the private keys to decrypt the ciphertext, users' privacy is protected. Furthermore, miners can utilize zero-knowledge proofs to verify whether users have accurately filled in the transaction details in ciphertext and whether they involve users on a sanction list. Hence, **challenge 2** has also been addressed. Toward Challenge 3. Hurricane Cash achieves compliance with minimal gas overhead when payments match a fixed pool size. To accommodate irregular or high-value amounts, and thus remove the denomination rigidity of **Challenge 3**, we extend the same framework in §V with a second variant, *Hurricane UTXO*, which accepts arbitrary values under a single proof.

The contributions of this study can be delineated as:

Regulation-Compatible Mixers. We propose the Hurricane Mixer framework and deliver two concrete variants: *Hurricane Cash* (fixed-denomination pools) and *Hurricane UTXO* (denomination-free, arbitrary-amount transfers), all proved under one unified security model.

Sanction List Verification & Bidirectional Tracing. Every mixer operation is checked against an updatable on-chain sanction list, while "forward" and "backward" *Eyes* let an authorized regulator decrypt and trace fund flows in either direction without exposing honest-user data.

Dual-Mode for Cost Reduced Flex Denomination. *Hurricane Cash* pools fixed denominations, e.g., separate pools for a unit (e.g., 1) ETH, so a transfer that matches one of these preset amounts inserts no extra Merkle leaf and requires only a single proof. *Hurricane UTXO* removes denomination rigidity: a single proof can move, e.g., multiple ETH without splitting into multiple unit ETH notes, or padding sub-denomination amounts up to a unit ETH, therefore saving both user fees and on-chain state when amounts exceed one pool unit or are fractional (e.g., < 1 ETH). Our empirical study shows the crossover point: Cash is cheaper up to one denomination; UTXO becomes cheaper (and lower-latency) once the payment exceeds two denominations or is sub-denomination. Unlike Tornado Cash Nova, our UTXO variant retains full regulator compatibility.

The remainder of the paper is organised as follows. Section II reviews background and related work. Section III formalises our system and adversary model. Section IV details the Hurricane Cash protocol, and Section V extends it to the denomination-free Hurricane UTXO variant. Section VI explains the bidirectional tracing mechanism. Section VII provides the unified security proofs. Section VIII reports our empirical evaluation, and Section IX concludes.

II. Background

Smart Contract. The account-based blockchain model has gained significant traction in various real-world applications, notably with platforms like Ethereum [7] and decentralized finance applications [8], [9], thanks to its support for smart contracts. In this model, there are two main account types: externally owned accounts (EOAs), which are managed through private keys, and contract accounts, which function as self-executing Turing-complete programs [10]. EOAs allow users to perform transactions, such as sending tokens or triggering smart contract operations. Meanwhile, contract accounts, created by EOAs, house publicly accessible code that executes automatically when transactions are received, facilitating intricate logic and state modifications on the blockchain.

Zero-knowledge Proof. Zero-Knowledge Proofs (ZKPs) are cryptographic protocols that facilitate secure interaction between two or more parties. Introduced by Goldwasser, Micali, and Rackoff [11], ZKPs enable a prover to demonstrate the truth of a statement to a verifier without conveying any information beyond the statement's validity. In formal terms, a ZKP is characterized by three algorithms: P, the prover; V, the verifier; and S, the simulator. The key attributes of a ZKP are as follows:

- **Completeness:** For a true statement, an honest verifier adhering correctly to the protocol will be assured of its truth by an honest prover. Specifically, for any x and w where $x \in L_w$ and under any verifier strategy V^*, $\Pr[(P(x,w), V^*(x)) = 1] = 1$.
- **Soundness:** If the statement is incorrect, no deceptive prover can persuade the honest verifier of its truth, except with a negligible probability. Specifically, for any $x \notin L_w$, any prover strategy P^*, and any w', $\Pr[(P^*(x,w'), V(x)) = 1] \leq \text{negl}(\lambda)$, where $\text{negl}(\lambda)$ denotes a negligible function.
- **Zero-knowledge:** For a true statement, the verifier gains no additional knowledge apart from the truth of the statement. This is demonstrated by the existence of a simulator S that, without the prover's input but only the statement, can generate a transcript that mirrors the interaction between the honest prover and the verifier. Formally, for any x, w where $x \in L_w$, any verifier strategy V^*, and any w', $\{(P(x,w), V^*(x))\} \approx \{(S(x,w'), V^*(x))\}$, where the $\approx$ indicates computational indistinguishability.

For instance, a prover can validate the existence of a preimage for a given hash value without disclosing the preimage itself. Subsequent research [12], [13], [14], [15], [16] has focused on refining ZKP schemes to reduce proof sizes, decrease computational overhead, and eliminate the need for a trusted setup. Due to its robust privacy guarantees and scalability, ZKP technology has been widely adopted in various applications, including cryptocurrency transactions in platforms like Tornado Cash [2] and Zcash [17], bridging mechanisms between different blockchains [18], off-chain data verification via decentralized oracles [19], decentralized identity systems [20], and decentralized data retrieval solutions [21].

Tornado Cash - Mixing Solution in Cryptocurrency. Notable examples of privacy-preserving technologies in cryptocurrency include ring signatures, which enable transaction signing by any member of a group without disclosing the specific signer, obfuscating the transaction's origin to complicate traceability [22]; stealth addresses, which generate unique, one-time addresses for each transaction, preventing the linkage of multiple transactions to a single public key to hinder traceability of the recipient's transactions [23]; and mixers such as CoinJoin [24]. Among these, Tornado Cash [2], a protocol on the Ethereum blockchain, stands out by utilizing zero-knowledge proofs to break the on-chain link between sender and receiver addresses, preserving transaction privacy.

When a user deposits cryptocurrency into Tornado Cash, the amount is standardized to specific denominations (e.g., 0.1, 1, 10 ETH) to obfuscate transaction patterns. Each deposit requires a secret s, from which the user derives a Leaf $L = hash(s, 1)$ and a Nullifier $N = hash(s, 2)$, where $hash$ is a collision resistance hash function. Since the character of non-collision, each pair of L and N is unique. When the token is deposited into the smart contract, the Leaf L will be added as a leaf in the on-chain stored Merkle Tree. Withdrawing tokens stored in Leaf L will consume the user's corresponding unique Nullifier N. During withdrawal, Tornado Cash will require the user to prove the knowledge about the secret s and the correctness of the provided Nullifier N. The user will generate a zero-knowledge proof π, which asserts the following statements are true: $N == hash(s, 2) \wedge L == hash(s, 1) \wedge L \in MerkleTree$, to prove the correctness of N and knowledge about s. To preserve privacy, the proof π takes s, L, and Merkle proof for L as the private input; meanwhile, it takes N and Merkle Root as the public input. Since π will show nothing about L, no one else can know the relationship between the N and L. Thus, no one except the user herself knows where the withdrawal token comes from.

III. System & Security Model

This section provides the formal definition of the Hurricane Mixer protocol along with Regulation-Compatible Mixing.

Definition 1 (Hurricane Mixer). *The Hurricane Mixer is defined as the tuple $(\mathcal{H}, Enc, P_{enc}, V_{enc}, P_w, V_w, P_t, V_t)$. It consists of two variants: Hurricane Cash and Hurricane UTXO. Hurricane Cash supports fixed-denomination mixing, while Hurricane UTXO supports denomination-free (arbitrary amount) mixing. Specifically:*

- *Hurricane Cash is defined as $(\mathcal{H}, Enc, P_d^{cash}, V_d^{cash}, P_w^{cash}, V_w^{cash}, P_t^{cash}, V_t^{cash})$*
- *Hurricane UTXO is defined as $(\mathcal{H}, Enc, P_d^{UTXO}, V_d^{UTXO}, P_w^{UTXO}, V_w^{UTXO}, P_t^{UTXO}, V_t^{UTXO})$*

Here, $\mathcal{H}$ denotes a cryptographic hash function, and Enc is a public-key encryption scheme. The terms P_{enc}, P_w, P_t

refer to the zero-knowledge proof provers for encryption, withdrawal, and transfer respectively, while V_{enc}, V_w, V_t are the corresponding verifiers.

Before presenting the formal definition of Regulation-Compatible Mixing, several terminologies must first be clarified. This paper borrows the concept of Nullifier N and Leaf L in Tornado Cash. Nullifier N and Leaf L are computed by a secret s and two one-way function f_N, and f_L, such that $N = f_N(s), L = f_L(s)$. Let $\mathcal{N}$ represent the set of all used Nullifiers, and $\mathcal{L}$ represent the set of all observed Leaves (appearing in successfully executed and finalized transactions), with L_{sanc} denoting the set of leaves in the sanction list. Additionally, let $\mathcal{T}$ denote the set of all valid transactions.

Definition 2 (**Mixing Distinguisher**). *A Mixing Distinguisher is a Probabilistic Polynomial-Time (PPT) function* $D : \{0,1\}^{|L|} \times \{0,1\}^{|L|} \times \{0,1\}^{|N|} \rightarrow \{0,1\}$ *such that:*

- *D has access to read all blockchain history states.*
- $D(L_1, L_2, N) = \begin{cases} 0 & \exists s : f_L(s) = L \wedge f_N(s) = N \\ 1 & \nexists s : f_L(s) = L \wedge f_N(s) = N \\ rand\{0,1\} & \textit{otherwise} \end{cases}$

The formal definition of Regulation-Compatible Mixing is provided here based on the Mixing Distinguisher.

Definition 3 (**Regulation-Compatible Mixing**). *If a tornado-cash-based mixing protocol is a Regulation-Compatible Mixing protocol, then it must satisfy the following properties:*

- ***Regulation Compatible Transaction Legitimacy Preserving:*** *For a transaction txn that needs to consume Nullifier N whose corresponding leaf is L:*

$$\boldsymbol{Pr}\{txn \in \mathcal{T} | N \notin \mathcal{N} \cap L \notin L_{sanc}\} = 1 \wedge$$

$$\boldsymbol{Pr}\{txn \in \mathcal{T} | N \in \mathcal{N} \cup L \in L_{sanc}\} \leq negl(\lambda)$$

- ***Regulation-Compatible Privacy Preserving:*** *For any Mixing Distinguisher D without knowing the regulator private key, a nullifier N consumed on block height i with its corresponding leaf L :*

$$\boldsymbol{Pr}\{D(L, L', N) = L | L' \sim \mathcal{L}, \forall D\} \leq \frac{1}{2} + negl(\lambda)$$

On the contrary, for any Mixing Distinguisher D' knowing regulator private key:

$$\boldsymbol{Pr}\{D(L, L', N) = L | L' \sim \mathcal{L}, \forall D'\} \geq 1 - negl(\lambda)$$

Hurricane Mixer functions under the same security assumptions as the Tornado Cash system. This implies that, within the context of Hurricane Mixer, any participant except the regulator, including those who have access to blockchain data, can act maliciously. Furthermore, the blockchain itself is assumed to be secure, ensuring that all on-chain finalized transactions are correctly executed via the smart contract. Lastly, the system presumes that all entities are limited to PPT computational power.

IV. Hurricane Cash

Hurricane Cash is a regulation-compatible mixing protocol built on Tornado Cash. It allows a regulator to use a public-key encrypted message, called Eye, to link the Nullifier and Leaf, enabling fund traceability. The protocol also supports a regulator-maintained sanction list to freeze tokens suspected of illicit use. Hurricane Cash retains the security and privacy of Tornado Cash while enabling regulatory tracking and intervention.

Overview. Hurricane Cash, like Tornado Cash, employs the concepts of *Nullifier* and *Leaf* (Fig. 3). Only a user with the correct secret for a Nullifier-Leaf pair can withdraw mixed tokens. Unlike Tornado Cash, given a secret s, Hurricane Cash first computes $h = \mathcal{H}(s)$, then derives the Leaf as $L = \mathcal{H}(h, 1)$ and the Nullifier as $N = \mathcal{H}(h, 2)$. The regulator can trace the link between L and N via h (revealing fund flow in Tornado Cash [25]). However, h is hidden from the public by encrypting it as $Eye = \text{Enc}(\phi, h)$, where ϕ is the regulator's public key. In practice, Hurricane Cash uses two keys: a forward tracing key ϕ_f and a backward tracing key ϕ_b, producing Eye^f and Eye^b, respectively. This enables constant-time tracing, discussed later. Importantly, regulators cannot withdraw or spend funds, as they never learn s. Additionally, Hurricane Cash blocks withdrawals using any L in the regulator's sanction list $sanc$.

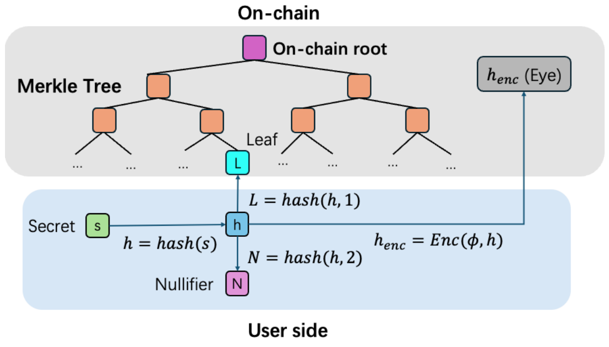


Fig. 3: Overview of Hurricane Cash

Deposit. When a user wants to mix tokens, they must deposit them into the Hurricane Cash contract via the **Deposit** protocol (Algorithm 1). The user generates a random secret s and computes $h = \mathcal{H}(s)$. Then, h is encrypted as $Eye^f = \text{Enc}(\phi_f, h)$ using the regulator's forward key ϕ_f. The use of Eye^f is explained in Section VI. The user then derives $L = \mathcal{H}(h, 1)$ and $N = \mathcal{H}(h, 2)$. A zero-knowledge proof π_d^{cash} is generated using prover P_d^{cash}, proving that $Eye^f = Enc(\phi_f, \mathcal{H}(s)) \wedge L = \mathcal{H}(\mathcal{H}(s), 1)$, with Eye^f, ϕ_f, and L as public inputs and s as a secret. The user then submits a transaction calling the **Deposit** function with the tokens, proof π_d^{cash}, L, and Eye^f. The contract verifies π_d^{cash} via verifier V_d. If verification fails, the transaction is reverted. If L is already present, it is also reverted. On success, L is inserted into the Merkle Tree $\mathcal{MT}$ and the tokens are accepted.

Withdraw. Hurricane Cash allows users to withdraw mixed tokens using the withdrawal process in Algorithm 2. Suppose the user aims to withdraw tokens corresponding to Leaf L and Nullifier N, and that a public sanction list $sanc$ exists, with its commitment R_{sanc} stored in the contract. Let R denote the Merkle root of $\mathcal{MT}$. Before withdrawal, the user computes the backward Eye $Eye^b = \text{Enc}(\phi_b, \mathcal{H}(s))$ and obtains the Merkle path M for L (as in Tornado Cash). The user then uses prover P_w^{cash} to generate a zero-knowledge proof π_w^{cash} attesting the

Algorithm 1: Deposit

```
USER:
  secret $s \sim \mathbb{F}_P$                         // Generate secret
  $h \leftarrow \mathcal{H}(s)$                                // Hash secret
  $Eye^f \leftarrow \textbf{Enc}(\phi_f, h)$                       // Generate Eye
  $L \leftarrow \mathcal{H}(h, 1)$                              // Generate Leaf
  $\pi_d^{cash} \leftarrow P_d^{cash}.\textbf{Prove}(Eye^f, \phi_f, s, L)$
  Send Transaction call $\textbf{Deposit}(L, \pi_d^{cash}, Eye^f)$
HURRICAN CONTRACT
  Deposit:
    Require: $L$, where $L = \mathcal{H}(\mathcal{H}(s), 1)$, $\pi_d^{cash}$, $Eye^f$
      $V_d.\textbf{Verify}(\pi_d^{cash}, Eye^f, L, \phi_f)$              // Verify $\pi_d^{cash}$
      if $L$ not Existed then
        $\mathcal{MT}.\textbf{Insert}(L)$, Receive Token
      else Abort                                   // Abort on failure
```

following: $Eye^b = Enc(\phi_b, \mathcal{H}(s)) \wedge \mathcal{H}(\mathcal{H}(s), 2) = N \wedge \mathcal{H}(\mathcal{H}(s), 1) = L \wedge \mathcal{MT}.Verify(R, M, L) = 1 \wedge L \notin sanc \wedge \mathcal{H}(sanc) = R_{sanc}$, with public inputs $Eye^b, \phi_b, N, R, R_{sanc}$, and secret inputs $s, L, M, sanc$. This proves the user knows s corresponding to L and N, that $L \in \mathcal{MT}$, $L \notin sanc$, and that Eye^b is correctly computed. Using a hash commitment R_{sanc} instead of the full $sanc$ list as input reduces on-chain gas costs. The user then submits a transaction invoking **Withdraw** with inputs $\pi_w^{cash}, R, R_{sanc}, N$, and Eye^b. The contract checks R and R_{sanc} against stored values, ensures N is unused, and verifies π_w^{cash} with V_w. If any check fails, the transaction reverts; otherwise, the contract transfers the token to the user.

Algorithm 2: Withdraw

```
RECEIVER:
  $Eye^b \leftarrow \textbf{Enc}(\phi_b, \mathcal{H}(s))$
  $\pi_w^{cash} \leftarrow P_w^{cash}.\textbf{Prove}(N, R, s, L, M, R_{sanc}, sanc, \phi_b,$
                                  $Eye^b)$
  Send Transaction call $(\pi_w^{cash}, R, R_{sanc}, N, Eye^b)$
HURRICAN CONTRACT:
  Withdraw:
    Require: $\pi_w^{cash}$, $R$, $N$, $R_{sanc}, Eye^b$
      $\textbf{ValidateSanc}(R_{sanc})$, $\textbf{ValidateRoot}(R)$,
      $\textbf{ValidateN}(N)$
      $V_w.\textbf{Verify}(\pi_w^{cash}, R, N, R_{sanc}, Eye^b)$
      if all verify and validate success then
        Transfer token, $\textbf{Record}(N)$
      else Abort                                   // Abort on failure
```

Transfer. Users can transfer mixed tokens using the **Transfer** process. The sender consumes a Nullifier N, while the receiver prepares a new Leaf L' to receive the tokens. The receiver first generates a new secret s', computes the intermediate hash h, Nullifier N', Leaf L', forward Eye $Eye^{f'}$, and a zero-knowledge proof π_d^{cash} (as in the *Deposit* process). Then the receiver sends L', $Eye^{f'}$, and π_d^{cash} to the sender. Upon receiving them, the sender computes Eye^b and generates a zero-knowledge proof π_t^{cash} using prover P_t^{cash}. This proof mirrors π_w^{cash}, confirming the sender's Nullifier N and corresponding Leaf L are valid. Additionally, π_t^{cash} includes L' as a public witness and applies a dummy constraint (e.g., $L' = L'$) to commit to L' and prevent front-running [25]. The sender then submits a transaction calling **Transfer**, including π_d^{cash}, π_t^{cash}, R, N, L', $Eye^{f'}$, Eye^b, and R_{sanc}. The contract first validates R, R_{sanc}, and ensures N is unused. It then verifies the proofs. On success, it marks N as spent and inserts L' into the Merkle Tree. Any failure causes the transaction to revert. After settlement, the receiver can use s' to withdraw or transfer the tokens. This process preserves privacy, as the sender only learns L', $Eye^{f'}$, and π_d^{cash}, none of which reveal the receiver's Nullifier N'.

Algorithm 3: Transfer

```
RECEIVER:
  secret $s' \sim \mathbb{F}_P$                        // Generate secret
  $h' \leftarrow \mathcal{H}(s')$                               // Hash secret
  $L' \leftarrow \mathcal{H}(h', 1)$                               // New leaf
  $Eye^{f'} \leftarrow \textbf{Enc}(h', \phi_f)$                     // Generate Eye
  $\pi_d^{cash} \leftarrow P_d.\textbf{Prove}(Eye^{f'}, \phi_f, s', L')$
  Send $L', Eye'$ and $\pi_d^{cash}$ to Sender
SENDER:
 Upon receiving $L', Eye^{f'}$ and $\pi_d^{cash}$:
  $Eye^b \leftarrow \textbf{Enc}(\mathcal{H}(s), \phi_b)$
  $\pi_t^{cash} \leftarrow P_t.\textbf{Prove}(N, R, L', L, M, s, Eye^b,$
                         $R_{sanc}, sanc)$
  Call $\textbf{Transfer}(\pi_d^{cash}, \pi_t^{cash}, R, N, L', Eye^{f'},$
                 $Eye^b, R_{sanc})$
HURRICAN CONTRACT:
 Transfer:
  Require: $\pi_d^{cash}, \pi_t^{cash}, R, N, L', Eye^b, Eye^{f'}, R_{sanc}$
    $\textbf{ValidateSanc}(R_{sanc})$, $\textbf{ValidateRoot}(R)$,
    $\textbf{ValidateN}(N)$
    $V_d.\textbf{Verify}(\pi_d^{cash}, Eye^{f'}, L')$
    $V_t.\textbf{Verify}(\pi_t^{cash}, N, R, L', R_{sanc}, Eye^b)$
    if all validate and verify passed then
      $\mathcal{MT}.\textbf{Insert}(L')$, $\textbf{Record}(N)$
    else Abort                                   // Abort on failure
```

V. HURRICANE UTXO

The Hurricane Cash protocol described in the previous section supports only fixed-denomination token mixing, which imposes limitations that increase the cost of transferring funds. For example, in a protocol restricted to denominations of 1 ETH, 5 ETH, and 10 ETH, transferring 19 ETH from Alice to Bob would require at least six separate transactions. Moreover, this fixed-denomination model prevents the transfer of amounts smaller than the minimum denomination; if the smallest supported unit is 1 ETH, it becomes impossible to transfer 0.1 ETH. In contrast, the UTXO model addresses this limitation by allowing each UTXO to represent an arbitrary value, thereby enabling greater flexibility. While Tornado Cash Nova proposed a UTXO-based mixer to overcome this issue, it lacks regulatory compliance. Consequently, this paper explores the design of a UTXO-based mixing protocol that also supports regulatory oversight. In the following section, we will first introduce a new construction of UTXO-based mixer, UTXO Mixer. Then we introduce **Hurricane UTXO**, a mixing service based on UTXO Mixer that adheres to the regulatory framework outlined previously.

A. UTXO Mixer

To easily construct Eye in the future modification, we introduce the UTXO Mixer. Each UTXO consists of a secret value s and an integer n representing the token amount. The corresponding leaf L is computed as $L = \mathcal{H}(\mathcal{H}(s), n, 1)$, and the nullifier N as $N = \mathcal{H}(\mathcal{H}(s), n, 2)$. This design enables a user with (s, n) to withdraw or transfer the associated tokens. Leveraging cryptographic hash function (CHF) properties [26], pre-image resistance prevents recovery of (s, n) from L and N, while collision resistance ensures that each (L, N) uniquely corresponds to one (s, n). A formal security analysis follows in later sections.

To mix tokens, users must first deposit them. The user generates a random secret s and, with a chosen amount n, computes the corresponding Leaf and Nullifier as $L = \mathcal{H}(\mathcal{H}(s), n, 1), \quad N = \mathcal{H}(\mathcal{H}(s), n, 2)$. Since the blockchain lacks access to s, the user must prove that L correctly encodes n by generating a zero-knowledge proof π_d^{Mix} for the statement: $L = \mathcal{H}(\mathcal{H}(s), n, 1)$, with L and n as public inputs and s as a secret. The user includes π_d^{Mix}, L, and n in a transaction calling the Deposit function of the UTXO Mixer contract, along with n tokens. The contract verifies π_d^{Mix} and checks that the transferred token amount matches n. If either check fails, the transaction is reverted. On success, the contract inserts L into the Merkle tree.

The UTXO Mixer allows users to withdraw an arbitrary amount of tokens, provided it does not exceed their total mixed balance. Suppose a user wishes to withdraw k tokens by selecting l owned UTXOs totaling $m = \sum_{i=1}^{l} n_i$ tokens, where $k \leq m$. Each selected UTXO has Leaf L_i, Nullifier N_i, secret s_i, and amount n_i. Let R be the current Merkle root and M_i the Merkle path for L_i. To preserve privacy, the user generates a new secret s' and computes a new Leaf $L' = \mathcal{H}(\mathcal{H}(s'), m - k, 1)$ for the remaining balance. The user constructs a zero-knowledge proof π_w^{Mix} proving ownership of the selected UTXOs and correctness of L', by attesting to the following:

$$\begin{aligned}\bigcap_{i=1}^{l} \Big(&\mathcal{MT}.Verify(R, M_i, L_i) = 1 \\ &\wedge\ L_i = \mathcal{H}(\mathcal{H}(s_i), n_i, 1) \\ &\wedge\ N_i = \mathcal{H}(\mathcal{H}(s_i), n_i, 2) \Big) \bigcap \\ \Big(\mathcal{H}(\mathcal{H}(s'), &\sum_{j=1}^{l} n_i - k, 1) = L' \Big) \bigcap \Big(\sum_{j=1}^{l} n_i - k \geq 0 \Big)\end{aligned}$$

Here, R, k, L', and all N_i are public inputs; M_i, L_i, s_i, n_i, and s' are private. The user submits π_w^{Mix} and public inputs to the Withdraw function. The contract checks that all N_i are unused, and then verifies π_w^{Mix} using V_w. If all checks pass, it transfers k tokens and marks N_i as spent; otherwise, the transaction is reverted.

The UTXO Mixer supports a transfer function that enables users to send mixed tokens to others. The sender consumes sufficient UTXOs to cover the transfer amount k, after which two new leaves are inserted into the Merkle tree: one for the sender's change (L') and one for the receiver's received tokens (L''). The receiver generates a new Leaf $L'' = \mathcal{H}(\mathcal{H}(s''), k, 1)$ along with a proof π_d^{Mix}, similar to the deposit case, and sends them to the sender. Upon receipt, the sender firstly selects UTXOs totaling n tokens ($n \geq k$). Then the sender computes $L' = \mathcal{H}(\mathcal{H}(s'), n - k, 1)$ and constructs a transfer proof π_t^{Mix} similar to the withdrawal proof π_w^{Mix}, with one additional constraint: a dummy constraint $L'' = L''$, allowing inclusion of L'' as a public input and protecting against front-running. The sender submits a transaction invoking the Transfer function, including π_t^{Mix}, π_d^{Mix}, and public inputs (e.g., R, k, L', L'', and all N_i). The contract first checks that all N_i are unspent, then verifies both proofs using V_t^{Mix} and V_d^{Mix}. If successful, it inserts both L' and L'' into the Merkle tree and consumes all N_i. Otherwise, the transaction is reverted.

B. Hurricane UTXO

The Hurricane UTXO protocol requires users to encrypt the pair $(\mathcal{H}(s), n)$ that appears in the UTXO Mixer, referred to as the *Eye*, using the regulator's public keys. Similar to Hurricane Cash, the Hurricane UTXO introduces the concepts of a forward Eye Eye^f and a backward Eye Eye^b. The regulator is equipped with two distinct public encryption keys: a forward key ϕ_f and a backward key ϕ_b. Accordingly, the forward Eye is computed as $Eye^f = Enc(\phi_f, \mathcal{H}(s) \,\|\, n)$, while the backward Eye is obtained as $Eye^b = Enc(\phi_b, \mathcal{H}(s) \,\|\, n)$.

Once the information to be encrypted as the *Eye* is clearly defined, it is necessary to ensure that the user provides the correct ciphertext. As specified in the Hurricane Cash protocol, during both deposit and transfer operations, the user must provide the forward and backward Eyes, denoted as Eye^f and Eye^b, corresponding to the newly generated leaf and the consumed nullifier. Additionally, during both withdrawal and transfer, the user must prove that the leaf associated with the consumed nullifier is not present in the sanction list.

The Hurricane UTXO protocol adopts this approach by modifying the **Deposit**, **Withdraw**, and **Transfer** procedures of the UTXO Mixer. The primary changes lie in the zero-knowledge proof (ZKP) statements that the user must generate and prove.

In the **Deposit** phase, the user submits a proof π_d^{UTXO} that guarantees the correctness of the following statement: $L = \mathcal{H}(\mathcal{H}(s), n, 1) \wedge Enc(\phi_f, \mathcal{H}(s) \,\|\, n) = Eye^f$ Here, L, n, ϕ_f, and Eye^f are public inputs, while $\mathcal{H}(s)$ is treated as a private (secret) input. After constructing the proof, the user proceeds as in the original UTXO Mixer: submitting the zero-knowledge proof along with the tokens to be mixed to the smart contract.

In the **Withdraw** phase, it is assumed that a sanction list sanc already exists, along with its commitment R_{sanc} stored in the smart contract. All other terminology remains consistent with the UTXO Mixer section. The zero-knowledge proof π_w^{UTXO} must demonstrate the correctness of the following statements:

$$\bigcap_{i=1}^{l}\Big(\mathcal{MT}.Verify(R, M_i, L_i) = 1$$
$$\wedge\ L_i \notin sanc \wedge \mathcal{H}(sanc) = R_{sanc}$$
$$\wedge\ L_i = \mathcal{H}(\mathcal{H}(s_i), n_i, 1)$$
$$\wedge\ Eye^b_i = Enc(\phi_b, \mathcal{H}(s_i)||n_i)$$
$$\wedge\ N_i = \mathcal{H}(\mathcal{H}(s_i), n_i, 2)\Big)\bigcap$$
$$\Big(Eye^{f'} = Enc(\phi, \mathcal{H}(s')||\sum_{j=1}^{l} n_i - k)\Big)\bigcap$$
$$\Big(\mathcal{H}(\mathcal{H}(s'), \sum_{j=1}^{l} n_i - k, 1) = L'\Big)\bigcap\Big(\sum_{j=1}^{l} n_i - k \geq 0\Big)$$

In this construction, the public inputs include R, k, L', all N_i, all Eye^b_i, the public key ϕ_b, and the sanction list commitment R_{sanc}. The secret inputs include the Merkle paths M_i, secrets s_i and s', token amounts n_i, leaves L_i, and the sanction list sanc. After receiving the proof π^{UTXO}_w, the smart contract verifies it. If the proof is valid—i.e., all conditions are satisfied and none of the nullifiers N_i have been used—the contract proceeds with processing the withdrawal.

In the **Transfer** phase, as in the UTXO Mixer, the receiver must first generate a new Leaf L'' to store the received tokens. The receiver also constructs a zero-knowledge proof π^{UTXO}_d attesting to the correctness of L'', and sends both the leaf and the proof to the sender. Next, the sender generates a transfer proof π^{UTXO}_t, which consumes sufficient nullifiers to cover the transfer amount and computes the sender's change. As in the UTXO Mixer, the structure of π^{UTXO}_t is identical to that of the withdrawal proof π^{UTXO}_w, except for the inclusion of an additional public input: the received leaf L'', along with a dummy constraint asserting $L'' = L''$. This ensures that L'' is bound into the proof without revealing further information. The sender then submits both π^{UTXO}_d and π^{UTXO}_t, along with all required public inputs, to the blockchain. After the smart contract verifies both proofs and confirms that none of the nullifiers have been previously spent, it proceeds to insert the new leaves into the Merkle tree and records all consumed nullifiers to prevent double-spending.

VI. Tracing

Both the Hurricane Cash and Hurricane UTXO constructions ensure that a valid forward Eye Eye^f and a valid backward Eye Eye^b are generated when a new leaf is created or a nullifier is consumed, respectively. This design enables the regulator to trace transactions *bidirectionally*. Specifically, given a suspicious transaction that either consumes a nullifier or creates a new leaf, the regulator can identify the corresponding leaf associated with the nullifier, or conversely, identify the nullifier that will eventually consume the specified leaf.

When the regulator identifies a suspicious fund and wishes to trace it from a deposit into the mixer to a corresponding withdrawal, she performs a *forward trace*. Suppose the regulator has already located the deposit transaction tx_0 that introduced the suspected fund into the Hurricane Cash or Hurricane UTXO mixer. The first step is to decrypt the forward Eye Eye^f using her forward encryption key ϕ_f, obtaining the intermediate hash h (where $h = \mathcal{H}(s)$ in Hurricane Cash and $h = \mathcal{H}(s) \,\|\, n$ in Hurricane UTXO). With this value, the regulator can compute the corresponding nullifier: $N = \mathcal{H}(\mathcal{H}(s), 2)$ for Hurricane Cash, or $N = \mathcal{H}(\mathcal{H}(s), n, 2)$ for Hurricane UTXO. By identifying the transaction that consumes this nullifier, the regulator can determine the next step in the fund's movement within the mixer. By recursively applying this procedure—decrypting each forward Eye, computing the corresponding nullifier, and identifying the transaction that consumes it—the regulator can trace the complete flow of the suspected funds. Once a withdrawal transaction is encountered, the regulator learns which external address ultimately received the funds.

The same logic applies when the regulator performs a *backward trace*, i.e., tracing a fund from a withdrawal transaction back to its originating deposit. In this case, the regulator begins by decrypting the backward Eye Eye^b, which is included in the withdrawal transaction, using her backward encryption key ϕ_b. This decryption reveals the intermediate hash h. Using this value, the regulator can compute the corresponding leaf L that is associated with the nullifier consumed during the withdrawal. Once L is known, she can identify the transaction that created it. By recursively repeating this process—decrypting each backward Eye, computing the corresponding leaf, and locating the transaction that generated it—the regulator can trace the complete transaction chain involved in the suspected fund flow. This backward tracing continues until a deposit transaction is reached, which reveals the original address from which the funds entered the mixer.

Discussion on Regulator Assumption. The above system design assumes an honest regulator, which may be considered a strong assumption. To mitigate this, the regulator's role can be decentralized. Specifically, the regulator can consist of a group of n independent entities, each holding a private key sk_i. The system then adopts a (t, n)-threshold public key encryption scheme [27], where a shared public key ϕ is used to encrypt the Eye, and decryption requires collaboration from at least t of the n entities. This design removes the need to trust a single regulator and instead assumes that no more than $t - 1$ of the n entities are malicious. Such decentralization enhances system robustness and reduces trust assumptions in adversarial environments.

VII. Security Argument

To validate that Hurricane Mixer balances privacy and compliance, we now formally prove it satisfies the definition of Regulation-Compatible Mixing. This ensures only legitimate transactions are accepted while protecting honest users' privacy.

Theorem 1 (Regulation-Compatible Mixing)**.** *The Hurricane Mixer—comprising the fixed-denomination variant (**Hurricane Cash**) and its variable-denomination extension (**Hurricane UTXO**)—satisfies the two properties of Regulation-Compatible Mixing (Definition 3):*

1) *Regulation-Compatible (RC) Transaction Legitimacy Preserving, and*
2) *Regulation-Compatible (RC) Privacy Preserving.*

This theorem formalizes the system's core security and privacy guarantees: it ensures that sanctioned users are blocked, honest users maintain their privacy, and regulators retain the ability to trace misuse. Together, these properties establish a solid foundation for a regulation-compliant mixer.

Proof. We treat each property in turn, following the game-based style of the original proof.

RC Transaction Legitimacy Preserving. For any transaction txn that consumes a set of nullifiers $\{N_i\}$ linked to leaves $\{L_i\}$: $\Pr[txn \in \mathcal{T} \mid \forall i: N_i \notin \mathcal{N} \wedge L_i \notin L_{\text{sanc}}] = 1$, $\Pr[txn \in \mathcal{T} \mid \exists i: N_i \in \mathcal{N} \vee L_i \in L_{\text{sanc}}] \leq \text{negl}(\lambda)$.

a) Legitimate case.: For withdrawals or transfers, the user provides a non-interactive zero-knowledge (NIZK) proof $\pi \in \{\pi_w, \pi_t\}$ that certifies:

- Knowledge of secrets s_i (and values n_i for UTXO) such that: $L_i = \mathcal{H}(\mathcal{H}(s_i), 1), \quad N_i = \mathcal{H}(\mathcal{H}(s_i), 2)$ in Hurricane Cash, $L_i = \mathcal{H}(\mathcal{H}(s_i), n_i, 1), \quad N_i = \mathcal{H}(\mathcal{H}(s_i), n_i, 2)$ in Hurricane UTXO.
- Every L_i is in the Merkle tree: the proof includes a Merkle path M_i such that the on-chain root R satisfies $\text{MerkleVerify}(R, M_i, L_i) = 1$
- Every L_i is not in the sanction list, and the hash of the list matches R_{sanc}: $L_i \notin \text{sanc}, \quad \mathcal{H}(\text{sanc}) = R_{\text{sanc}}$.
- Encrypted tracing data is correct. For example: $Eye_i^b = \text{Enc}(\phi_b, \mathcal{H}(s_i))$ in Hurricane Cash, $Eye_i^b = \text{Enc}(\phi_b, (\mathcal{H}(s_i), n_i))$ in Hurricane UTXO.

  For Hurricane UTXO, the proof additionally enforces:

$$\sum_i n_i - k \geq 0, \quad L' = \mathcal{H}(\mathcal{H}(s'), \sum_i n_i - k, 1)$$

All conditions are satisfied, so the verifier accepts and the contract records $txn \in \mathcal{T}$ with probability 1.

b) Illegitimate case.: If an adversary submits a transaction violating any condition (e.g., double-spending $N_i \in \mathcal{N}$, using $L_i \in L_{\text{sanc}}$, or claiming a false token amount), then any valid proof must either:

- Break soundness of the zero-knowledge proof,
- Find a preimage or collision of $\mathcal{H}$,
- Decrypt or forge a ciphertext under the semantically secure encryption scheme.

All of these events have negligible probability in the security parameter λ, thus: $\Pr[txn \in \mathcal{T} \mid \text{illegitimate input}] \leq \text{negl}(\lambda)$

RC Privacy Preserving. We analyze two games following the original definition.

c) Game 1 (no regulator key).: Let a probabilistic polynomial-time adversary D select two leaves $L_0, L_1 \in \mathcal{L}$. The challenger chooses a random bit $b \in \{0, 1\}$ and creates a transaction using L_b with corresponding nullifier N_b, published along with Eye^b and ZK proof.

Since D cannot decrypt Eye^b nor link N_b to L_b, and all ZK proofs reveal zero knowledge, D cannot guess b with probability better than random: $\Pr[b' = b] \leq \frac{1}{2} + \text{negl}(\lambda)$.

d) Game 2 (with regulator key).: A distinguisher D' with access to the regulator's private key can decrypt Eye^b to learn $\mathcal{H}(s_b)$ (and n_b in UTXO), and recompute the corresponding L_b. Thus, $\Pr[b' = b] \geq 1 - \text{negl}(\lambda)$.

e) Conclusion.: The combined Hurricane protocol satisfies both Regulation-Compatible properties:

- Only legitimate transactions are accepted (ensured by cryptographic soundness, preimage resistance, and proof validation),
- User privacy is preserved against all external observers (except the regulator), with tracing possible only via decryption using the regulator's private key. □

VIII. Experimental Studies

In this section, we perform experimental studies to answer the following three questions, which are widely studied by many other works [20], [28], [18], [19]:

- **Q1.** What's the cryptographic computation cost for our proposed scheme Hurricane Cash (HC) and Hurricane UTXO (HU)? (§VIII-B)
- **Q2.** How much on-chain transaction gas is required for our proposed HC and HU? (§VIII-C)
- **Q3.** How efficient are HC and HU with regard to the system performance? (§VIII-D)

We summarize our conclusions and findings in §VIII-E.

TABLE I: Experimental Settings.

Parameters	Values
Merkle Depth dmt	2, 4, 8, 16, **32**
Sanction List Size ssl	64, 128, **256**, 512, 1024
Number of UTXO Inputs n_{in}	1, 2, **3**, 4, 5
Transaction Amount amt	**1**, 3, 5, 7, 9

A. Experimental Configuration

Configurable Parameters. We summarize our parameter settings in Table I with default values in bold. *(1)* As introduced in Section IV, Hurricane Cash is mainly related to two parameters: the depth of a Merkle tree dmt and the size of a sanction list ssl. Thus, we evaluate the computation cost under the effect of both dmt and ssl. Specifically, we adopt the default setting $dmt = 32$, same as the actual configuration in Tornado Cash [2], where it can store more than 4 billion transactions, sufficient to meet practical application requirements. *(2)* Moreover, for Hurricane UTXO, we also test its performance under the varying number of UTXO inputs n_{in}, where we set the default $n_{in} = 3$ as the empirical average number on the Bitcoin network [29]. *(3)* Finally, we compare the system performance under different transaction amount amt, mirroring the real world transaction scenarios.

Evaluation Metric. Privacy-preserving transaction mechanism incurs costs across multiple dimensions, including ZKP circuit size, temporal costs (e.g., verification time, proof generation time), and monetary costs (e.g., transaction fees), etc., as the conventions in the related papers [20], [28], [18], [19]. Specifically, we report the *(1) cryptographic cost* including metrics of the circuit size, proof time and verify time; *(2) monetary cost* of the gas consumption; and *(3) system performance* consisting of the throughput and transaction latency.

- The **circuit size** refers to the number of constraints included in a zero-knowledge proof. The higher the circuit size, the more complex the zero-knowledge proof, resulting in higher costs associated with its resource demand and computation.
- The **proof time** of a method refers to the time taken for the method to generate a zero-knowledge proof. The shorter the proof time of a method, the more efficient it is at generating a zero-knowledge proof, resulting in lower latency for transaction generation using this method.

- The **verify time** of a method is the time required to verify a zero-knowledge proof generated by this method. The shorter the verification time of a method, the more efficient it is in verifying the zero-knowledge proofs it generates, thereby reducing transaction latency.
- The **gas cost** represents the amount of computational effort required to execute a transaction on Ethereum, involving deposit, transfer and withdraw. The higher the gas cost, the more expensive the transaction fee that users need to pay.
- The **throughput** measures the number of transactions processed per second. Due to the constrained gas limit per block on blockchains like Ethereum, complex transactions with high gas costs reduce the number of transactions that can fit into each block, thereby decreasing the system throughput.
- The **latency** records the duration when a transaction is successfully processed. It's impacted by both the processing complexity of the transaction implied by its gas cost, and the amount splitting operations in the setting of limited denominations for both Hurricane Cash and Tornado Cash.

Compared Approaches. We compare the performance with the following approaches:

- *Tornado Cash (TC)*, the de facto ZKP-based mixing service widely adopted on Ethereum, serving as the baseline without regulation.
- *Hurricane Cash (HC)*, our proposed regulation-compatible token mixer.
- *Hurricane UTXO (HU)*, our proposed UTXO-based regulation-compatible mixing solution, optimized for transactions with arbitrary transferring amount.

Specifically, for each compared approaches, we break down its workflow into three main components for comparison:

- *Deposit*, the user depositing his/her token in the contract of the mixing service, where both HC and HU requires generating a ZK proof π_d.
- *Transfer*, the sender transferring the mixed tokens to another receiver, requiring the ZK proof π_t besides π_d.
- *Withdraw*, the user withdrawing mixed tokens from the mixer's contract, at the cost of the ZK proof π_w.

Experimental Environment. In all experiments, we use MiMC hash [30] (a ZK-friendly hash function) as the $hash$ and use the EC-ElGamal [31] algorithm as the public key encryption scheme Enc. In addition, we use Groth-16 [15] protocol as our zero-knowledge proof protocol, and all circuits are built in Golang and based on Gnark [32] library. All proof generation experiments were executed on a MacBook Pro with M1 pro chip and 32GB Memory. Moreover, the Hurricane Contract was written in Solidity and tested on the Remix VM[1].

B. Cryptographic Computation

To answer Q1, we report the results in ZKP circuit size, proof generation time and verification time respectively.

The impact of varying Merkle depth dmt. Both circuit size and proof time for transfer and withdraw operations increases as dmt grows, while verify time and all computation costs for deposit operations remain constant as shown in Fig. 4a-4c. During the deposit operation, the cryptographic cost mainly comes from the generation of the ZK proof π_d for the correctness of Eye^f and Leaf L. The dmt dependent operation of inserting the Leaf L into the Merkle Tree is performed in the mixer's smart contract on chain, while the construction of L's correctness proof is unaffected.

In contrary, the cryptographic cost for the withdraw operation increases as dmt grows. This is because when withdrawing tokens, the generated ZK proof π_w requires additionally proving Leaf L's membership in the Merkle tree. Specifically, HC requires more cryptographic computation compared to TC, as it takes extra cost for HC to construct the ZK proof for asymmetric encrypted Eye^b to enable regulation. Furthermore, the withdraw operation in HU requires extra cryptographic cost to generate a new Leaf L'. Regarding the transfer operation, the figures demonstrate that it has performance similar to the withdraw. In fact, transfer has one additional dummy constraint compared to withdraw, which is an extra Leaf L'' in the circuit to prevent double-spending attacks.

Finally, all compared approaches have the same constant verify time. This is because that the Groth-16 algorithm used for the verification only takes 3 times of elliptic curve pairing, regardless of the proof size.

The impact of varying sanction list size ssl. In Fig. 4d-4f, the cryptographic cost under the growing ssl illustrates a similar pattern as dmt varies. Still, the deposit operation is irrelevant to ssl, as its π_d merely addresses the correctness for L and Eye^f, without proving its membership in the sanction list. In contrary, the withdraw cost for HC and HU is linear to ssl, as this step requires proving the sanction list membership when constructing the ZK proof π_w by searching for leaf L in the sanction list with brute force. Notably, TC is irrelevant to ssl as it doesn't support regulation over the sanction list. Thus, HC takes higher cryptographic cost than TC. In addition to HC's extra cost for Eye^b's computation, HC's π_w also requires traversing the sanction list.

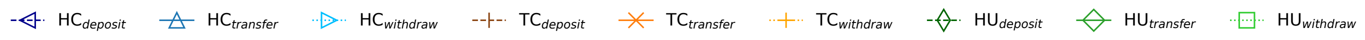
The impact of varying number of UTXO inputs n_{in}. In Fig. 4g-4h, the cost for HU's transfer and withdraw increases as n_{in} grows while its deposit remains fixed. The construction of proof π_d in deposit is irrelevant to n_{in}, since the deposit transaction can take multiple UTXO inputs and yield a single UTXO output with one proof π_d. In contrary, HU's cost for transfer and withdraw grows linearly to n_{in}. Specifically, a proof is required for every single UTXO, proving the correctness for its Nullifier N_i and Leaf L_i and its Merkle membership, and generating an Eye^b.

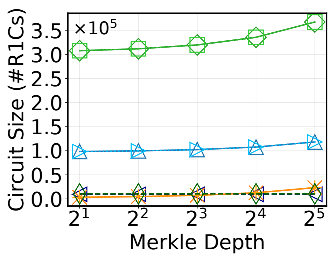
C. On-Chain Gas

To answer Q2, we report the experimental results in gas cost under varying parameters.

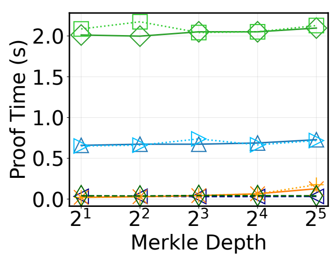
The impact of varying Merkle depth dmt. In Fig. 5a, HC demonstrates an efficient gas consumption comparable to TC with additional support for regulation. For deposit and transfer operations, all methods have a gas cost linear to the growing dmt, as their on-chain functions all require consuming gas to insert Leaf in the Merkle tree. TC has a slightly lower consumption as it doesn't need to store Eyes for regulation. However, for the withdraw operation, HC has the same constant gas cost as TC, as the withdraw transaction doesn't involve Merkle tree operation, but merely verify the ZK proof at a constant cost.

[1]https://remix.ethereum.org/

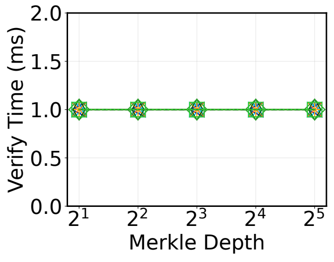




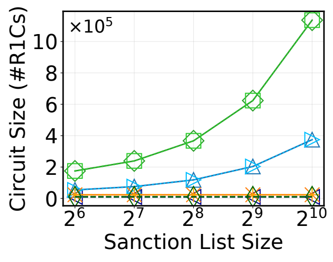
(a) Size of Varying dmt.



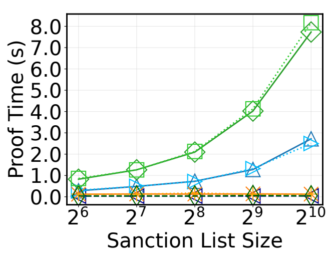
(b) Proof Time of Varying dmt.



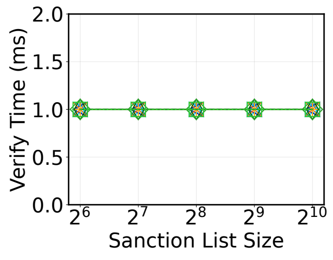
(c) Verify Time of Varying dmt.



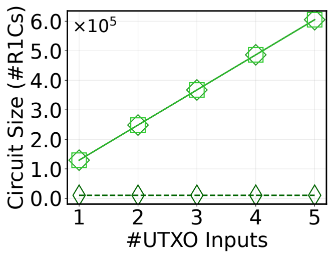
(d) Size of Varying ssl.



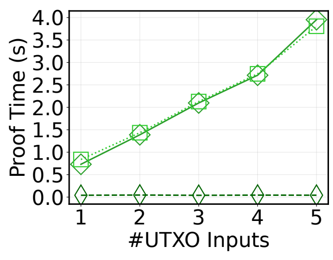
(e) Proof Time of Varying ssl.



(f) Verify Time of Varying ssl.

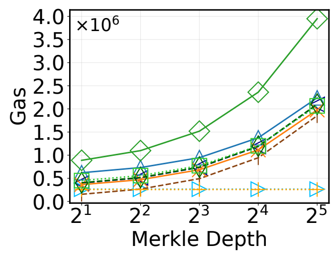


(g) Size of Varying n_{in}.

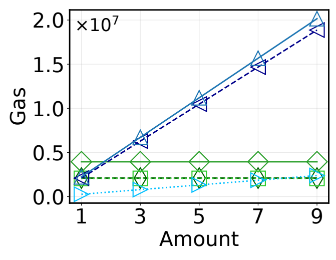


(h) Proof Time of Varying n_{in}.

Fig. 4: Experimental Results of Cryptographic Computation.



(a) Gas Cost of Varying dmt.



(b) Gas Cost of Varying amt.

Fig. 5: Experimental Results of On-Chain Gas.

In contrary, HU's withdraw is dependent on dmt. This is because that HU needs to additionally insert a new Leaf for the sender's charge in the UTXO output. Moreover, HU also has a higher gas consumption for the transfer operation. Specifically, HU's transfer generates 2 Leaves to insert in the Merkle tree: one for the receiver's received money, and one for the sender's charge. Therefore, such gap grows as the dmt increases.

The impact of varying transaction amount amt. As illustrated in Fig. 5b, the gas costs for all operations in HU remain stable, while the gas cost in HC, especially the deposit and transfer operations, grows linearly with the varying amt. For the small amount exactly equal to the unit denomination ($amt = 1$), both the withdraw and transfer operations in HU incur higher gas costs compared to those in HC. This discrepancy arises from an additional Merkle tree insertion required for both operations in HU. However, when the amount of tokens to be transferred or withdrawn exceeds the denomination size ($amt > 1$), HU can become more gas-efficient than HC. For instance, when $amt = 5$, the deposit and transfer operations in HC consumed merely 20% and 35% gas respectively compared to HU. This implies that if a user intends to transfer multiple denomination units, HU offers lower gas costs than HC. In contrast, for the withdraw operation, gas cost parity between HC and HU is achieved only when withdrawing $amt > 7$ times the denomination. Therefore, from a gas efficiency perspective, HU is preferable for operations involving either large token amounts or very small amounts (i.e., less than the smallest denomination in HC), provided that withdrawals are infrequent. Otherwise, HC may be the more cost-effective choice for withdrawals or transactions with unit denomination amounts.

D. System Performance

To answer Q3, we report the results in system throughput and transaction latency.

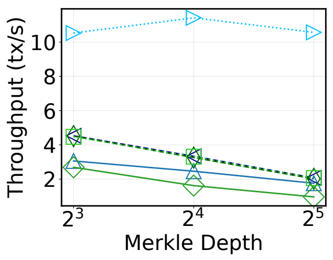
Throughput under varying Merkle depth dmt. In Fig. 6a, HC's system throughput outperforms HU. This is because of the constrained gas limit in each block, where each block will contain less transactions if they consume more gas. Given the higher gas consumption for HU due to extra Leaf insertion in Fig. 5a, it ends up with a lower throughput compared to HC.

Latency under varying transaction amount amt. In Fig. 6b, HC's latency grows linearly along with amt, while HU has a stable latency regardless of amt. This is because that HC needs to split a single transaction into multiple transactions with amounts matching the fixed denomination. HU is designed to handle multiple UTXOs at once. Thus HC's transaction latency grows linearly along with the growing amt, i.e., the number of split transactions, while HU's latency stays consistent.

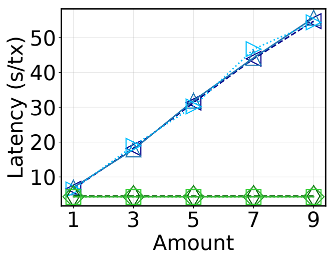
E. Experiment Summary

In closing, we summarize our findings as follows:

- **The answer to Q1** is that the proof construction of both HC and HU are affected by dmt and ssl, and HU is also impacted by n_{in}. As these parameters increase, it takes more time to generate larger zero-knowledge proofs. However, the verify time for the on-chain contract can stay constant regardless of these parameters.



(a) Throughput of Varying dmt.



(b) Latency of Varying amt.

Fig. 6: Experimental Results of System Performance.

- **The answer to Q2** is that all methods take higher gas costs with a growing dmt. Intuitively, inserting Leaves into a Merkle tree with large size (high dmt) consumes more gas. Moreover, HU illustrates a stable gas consumption under the varying amt, making HU more suitable for deposits and transfers with amounts above the denomination unit and HC preferable for withdrawals with amounts close to the unit.
- **The answer to Q3** is that HC has higher system throughput, while HU gains lower latency for real-world transactions with arbitrary amount. HU's system throughput is constrained by its higher gas consumption under the blockchain's gas limit. However, in real-world scenario where the transaction amount doesn't exactly match the denomination, HU yields a stable transaction latency much lower than HC.

IX. Conclusion

Hurricane Mixer reconciles privacy and oversight by embedding on-chain sanction-list checks and bidirectional "Eyes" into zero-knowledge mixers, delivering Cash (fixed-denomination) and UTXO (arbitrary-amount) variants that achieve regulation-compatible security without exposing honest users. Experiments show Cash slashes transfer gas when payments match pool sizes, while UTXO is cheaper for large or fractional amounts, and both maintain competitive efficiency. Overall, our evaluation confirms that practical, cost-effective, regulation-ready privacy mixers are feasible without sacrificing user confidentiality or system performance.

References

[1] M. Youn, K. Chin, and K. Omote, "Empirical analysis of cryptocurrency mixer: Tornado cash," in 2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE). IEEE, 2023, pp. 2324–2331.

[2] A. Pertsev, R. Semenov, and R. Storm, "Tornado cash privacy solution version 1.4," Tornado cash privacy solution version, vol. 1, p. 7, 2019.

[3] D. A. Zetzsche, F. Annunziata, D. W. Arner, and R. P. Buckley, "The markets in crypto-assets regulation (mica) and the eu digital finance strategy," Capital Markets Law Journal, vol. 16, no. 2, pp. 203–225, 2021.

[4] P. D. Biase, "Eu mica final approval - europe adopts comprehensive crypto legal framework," 2023, accessed: 2024-08-22. [Online]. Available: https://blockchain.bakermckenzie.com/2023/05/30/eu-mica-final-approval-europe-adopts-comprehensive-crypto-legal-framework/

[5] A. Brownworth, J. Durfee, M. Lee, and A. Martin, "Regulating decentralized systems: evidence from sanctions on tornado cash," FRB of New York Staff Report, no. 1112, 2024.

[6] J. Burleson, M. Korver, and D. Boneh, "Privacy-protecting regulatory solutions using zero-knowledge proofs," 2022.

[7] V. Buterin et al., "Ethereum white paper," GitHub repository, vol. 1, pp. 22–23, 2013.

[8] W. Ni, Z. Yiwei, W. Sun, L. Chen, P. Cheng, C. J. Zhang, and X. Lin, "Money never sleeps: Maximizing liquidity mining yields in decentralized finance," in Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2024, pp. 2248–2259.

[9] Z. Wang, J. Gao, and X. Wei, "Do nfts' owners really possess their assets? a first look at the nft-to-asset connection fragility," in Proceedings of the ACM Web Conference 2023, 2023, pp. 2099–2109.

[10] J. Huang, L. Kong, G. Cheng, Q. Xiang, G. Chen, G. Huang, and X. Liu, "Advancing web 3.0: Making smart contracts smarter on blockchain," in Proceedings of the ACM on Web Conference 2024, 2024, pp. 1549–1560.

[11] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali, 2019, pp. 203–225.

[12] 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.

[13] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," Cryptology ePrint Archive, 2018.

[14] 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.

[15] J. Groth, "On the size of pairing-based non-interactive arguments," in Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. Springer, 2016, pp. 305–326.

[16] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in Annual International Cryptology Conference. Springer, 2022, pp. 359–388.

[17] E. C. Company, "Zcash: Privacy-protecting digital currency," https://z.cash/, 2024, accessed: 2024-10-08.

[18] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkbridge: Trustless cross-chain bridges made practical," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022, pp. 3003–3017.

[19] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "Deco: Liberating web data using decentralized oracles for tls," in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 1919–1938.

[20] S. Zheng, Z. Li, J. Luo, Z. Xin, and X. Liu, "Idea-dac: Integrity-driven editing for accountable decentralized anonymous credentials via zk-json," in Proceedings of the ACM on Web Conference 2024, 2024, pp. 1868–1879.

[21] Z. Li, S. Zheng, J. Luo, Z. Xin, D. Yuan, S. Gao, S. Yang, B. Xiao, and X. Liu, "Poudr: Proof of unified data retrieval in decentralized storage networks," Cryptology ePrint Archive, 2024.

[22] M. N. S. Perera, T. Nakamura, M. Hashimoto, H. Yokoyama, C.-M. Cheng, and K. Sakurai, "A survey on group signatures and ring signatures: Traceability vs. anonymity," Cryptography, vol. 6, no. 1, p. 3, 2022.

[23] N. T. Courtois and R. 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, 2017, pp. 559–566.

[24] B. W. contributors, "Coinjoin," 2013. [Online]. Available: https://en.bitcoin.it/wiki/CoinJoin

[25] V. Buterin, "Some ways to use zk-snarks for privacy," 2022. [Online]. Available: https://vitalik.eth.limo/general/2022/06/15/using_snarks.html

[26] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, Handbook of applied cryptography. CRC press, 2018.

[27] C. Delerablée and D. Pointcheval, "Dynamic threshold public-key encryption," in Annual International Cryptology Conference. Springer, 2008, pp. 317–334.

[28] R. Song, S. Gao, Y. Song, and B. Xiao, ": A traceable and privacy-preserving data exchange scheme based on non-fungible token and zero-knowledge," in 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS). IEEE, 2022, pp. 224–234.

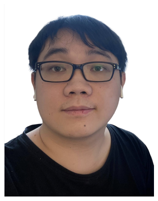
[29] C. Pérez-Solà, S. Delgado-Segura, G. Navarro-Arribas, and J. Herrera-Joancomartí, "Another coin bites the dust: an analysis of dust in utxo-based cryptocurrencies," Royal Society open science, vol. 6, no. 1, p. 180817, 2019.

[30] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in International Conference on the Theory and Application of Cryptology and Information Security. Springer, 2016, pp. 191–219.
[31] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE transactions on information theory, vol. 31, no. 4, pp. 469–472, 1985.
[32] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, "Consensys/gnark: v0.11.0," Sep. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.5819104

Zonglun Li received his B.Sc. degree in Computer Science from the University of Alberta, Edmonton, Canada. He is currently working toward Ph.D. degree with the School of Computer Science, McGill University, Canada. His research interests include Blockchain Systems, Data Privacy, and Cryptography Applications.

Wangze Ni received the BS degree in information science and electronic engineering from Zhejiang University, China, in 2017, the MPhil degree in technology, leadership, and entrepreneurship in 2019 and the PhD degree in computer science and engineering in 2024, both from the Hong Kong University of Science and Technology. He is currently a ZJU100 Young professor, with both the College of Computer Science and Technology and the Institute of Cyberspace Research (ICSR), Zhejiang University, China. His research interests include AI safety, AI efficiency, blockchain, and database.

Shuhao Zheng (Graduate Student Member, IEEE) received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 2021, and is currently working toward the Ph.D. degree in computer science at McGill University, Montréal, QC, Canada. His research interests include applied zero-knowledge proofs for blockchain, incentive-compatible mechanism design in decentralized finance, and large-scale distributed systems.



Junliang Luo received his B.Sc. degree in Computer Science from Northeastern University, and the M.Sc. degree in Computer Science from Dalhousie University. He is now pursuing the Ph.D. degree in Computer Science at McGill University. His research interests span blockchain analytics, transaction graph modeling, and characterization of decentralized finance (DeFi).

Weijie Sun received the B.Sc. degree in Computer Science from Zhejiang University. He is now pursuing the Ph.D. degree in Computer Science at the Hong Kong University of Science and Technology. His research interests include blockchain consensus, authenticated query, and database optimization.

Lei Chen is a Chair Professor in Data Science and Analytics at HKUST (GZ), a Fellow of ACM and IEEE. Currently, he serves as the Dean of the Information Hub and the Director of the Big Data Institute at HKUST (GZ). Prof. Chen's research spans several areas, including Data-driven AI, Big Data Analytics, the Metaverse, knowledge graphs, blockchain technology, data privacy, crowdsourcing, and spatial and temporal databases, as well as probabilistic databases. He earned his Ph.D. in Computer Science from the University of Waterloo, Canada. Prof. Chen has received several prestigious awards, including the SIGMOD Test-of-Time Award in 2015 and the Best Research Paper Award at VLDB 2022. His team's system also won the Excellent Demonstration Award at VLDB 2014. He served as the Program Committee Co-chair for VLDB 2019 and currently holds the position of Editor-in-Chief for IEEE Transactions on Data and Knowledge Engineering. In addition, he was the General Co-Chair of VLDB 2024 and will serve as the General Co-Chair of IJCAI China 2025.

Dr. Xue (Steve) Liu (S'02, M'07, SM'19, F'20) received the B.S. degree in Mathematics and the M.S. degree in Automatic Control from Tsinghua University in 1996 and 1999, respectively, and the Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 2006. He is the Associate VP of Research, and Professor of Machine Learning, and Professor of Computer Science at MBZUAI (Mohamed bin Zayed University of Artificial Intelligence). He is also a Professor in Computer Science and Professor of Mathematics and Statistics (Courtesy Appointment) at McGill University. He was VP R&D, Chief Scientist, and Co-Director of the Samsung AI Center Montreal, where he led the R&D of AI innovations in multiple areas including telecommunications, mobile computing, IoT, and robotics. He was also the Chief Scientist at Tinder Inc., leading the research and innovation for the world's largest dating and social discovery app . He worked briefly as the Samuel R. Thompson Chair Associate Professor in the Department of Computer Science and Engineering at The University of Nebraska-Lincoln, at Hewlett-Packard Labs in Palo Alto, California, and IBM T. J. Watson Research Center in New York.

Dr. Liu is an IEEE Fellow, and a Fellow of the Canadian Academy of Engineering. He is an associate member at the Quebec AI Institute (Mila). His research interests focus on AI/Machine Learning, Intelligent Computing and Communications Systems, Sustainable Computing, IoT and CPS. He has published 5 books and over 400 research papers in major peer-reviewed international journals and conference proceedings, and received 10 best paper awards from IEEE or ACM. He has served as associate editor or advisor of several international academic journals and was the chair of ACM SIGBED from 2021 to 2026. Dr. Liu is an advisor of several high-tech startups.

Tianhang Zheng is an assistant professor at Zhejiang University. He received his B.S. degree from Peking University, China, in 2016, M.S. degree from University at Buffalo, Buffalo, NY, USA, in 2019, and Ph.D. degree from University of Toronto, Toronto, ON, Canada, in 2023. His research interests focus on AI security and Privacy.

Zhan Qin is currently a ZJU100 Young Professor, with both the College of Computer Science and Technology and the Institute of Cyberspace Research (ICSR) at Zhejiang University, China. He was an assistant professor at the Department of Electrical and Computer Engineering in the University of Texas at San Antonio after receiving the Ph.D. degree from the Computer Science and Engineering department at State University of New York at Buffalo in 2017. His current research interests include data security and privacy, secure computation outsourcing, artificial intelligence security, and cyber-physical security in the context of the Internet of Things. His works explore and develop novel security sensitive algorithms and protocols for computation and communication on the general context of Cloud and Internet devices.

Kui Ren received degrees from three different majors, i.e., his Ph.D. in Electrical and Computer Engineering from Worcester Polytechnic Institute, USA, in 2007, M.Eng in Materials Engineering in 2001, and B.Eng in Chemical Engineering in 1998, both from Zhejiang University, China. Professor Kui Ren, AAAS, ACM, CCF, and IEEE Fellow, is currently the dean of the College of Computer Science and Technology at Zhejiang University. He is mainly engaged in research in data security and privacy protection, AI security, and security in intelligent devices and vehicular networks.