StealthHub: UTXO-Based Stealth Address Protocol

Hanze Guo*¶, Yebo Feng†¶, Cong Wu‡¶, Zengpeng Li§, Jiahua Xu*¶
*Centre for Blockchain Technologies, University College London, United Kingdom, {hanze.guo.24, jiahua.xu}@ucl.ac.uk
†Nanyang Technological University, Singapore, yebo.feng@ntu.edu.sg
‡The University of Hong Kong, China, congwu@hku.hk
§Shandong University, China, zengpeng@email.sdu.edu.cn
¶DLT Science Foundation, United Kingdom

***Abstract*—Privacy remains a significant challenge in public blockchain ecosystems. Mainstream add-on privacy solutions, such as Stealth Address Protocols (SAPs) and Zero-Knowledge Proof (ZKP)-based mixers, have recently attracted considerable attention. However, existing SAPs offer only ephemeral anonymity for users' transaction data, and their implementation and evaluation within the highly concurrent Unspent Transaction Output (UTXO) model remain largely unexplored. ZKP-based mixers are limited to native coin transfers with fixed denominations and require additional security assumptions, employing out-of-band encrypted channels to transmit *notes*. To overcome these challenges, we unify the core principles underlying both SAPs and ZKP mixers and formally introduce StealthHub, a UTXO-based SAP. Compared with the widely adopted dual-key-based Umbra protocol prevalent on Ethereum Virtual Machine (EVM)-compatible chains, StealthHub reduces computational overhead for the prepare and scan *announcements* stages by over 71% and 32%, respectively. Furthermore, by leveraging Merkle Mountain Range (MMR) commitments and off-chain batch aggregation, our StealthHub implementation lowers deposit and shielded transfer transaction costs to approximately 76% of those for a standard transfer, substantially improving practical usability.**

***Index Terms*—Blockchain, Smart Contract, Decentralised Application, Privacy, Stealth Address, Zero-Knowledge Proof**

I. Introduction

Permissionless Distributed Ledger Technologies (DLTs) enable decentralised asset transfers but expose transactions to privacy vulnerabilities [1–3], underscoring the need for effective add-on privacy solutions. Zero-Knowledge Proof (ZKP)-based mixers, exemplified by Tornado Cash [4], enhance transaction confidentiality using Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK)-based tumblers (mixing pools [5–8]), yet exhibit substantial limitations. Current mixers require that payers and payees are already acquainted, as payees can only acquire credentials generated by payers through out-of-band channels. This constraint limits their application primarily to self-transfers (referred to as intra-*entity* transfers, where the *entity* controls one or more public addresses), thereby hindering interoperability within the Service-Oriented Architecture (SOA). Moreover, fixed-denomination pools (e.g., 0.1, 1, or 10 ETH) significantly reduce transactional flexibility. The high operational cost also presents a major limitation. For example, depositing 1.7 ETH requires one 1 ETH deposit and seven 0.1 ETH deposits, totalling eight separate transactions. On Tornado Cash [4], this incurs approximately 10 million gas—around 480 times the cost of a standard transfer—posing significant barriers to scalability and adoption.

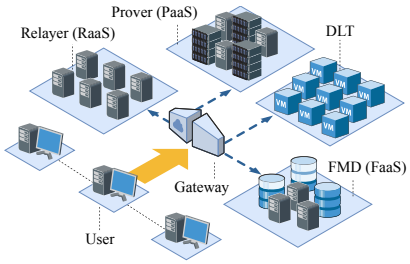


Fig. 1: Service-oriented decomposition of StealthHub.

Alternatively, Stealth Address Protocols (SAPs) protect user privacy by generating unique recipient-controlled one-time addresses per transaction, making linkage analysis significantly more difficult. However, Alex Márk et al. [9] have demonstrated that the anonymity offered by SAPs is fragile—approximately 48.5% of participants remain directly identifiable on Ethereum Mainnet. Moreover, existing SAP implementations, particularly Dual-Key Stealth Address Protocols (DKSAPs), incur significant communication overhead due to complex stealth-address algorithms and present usability challenges when transferring non-native assets because no native cryptocurrency is available to cover fees.

Motivated by these shortcomings, we propose StealthHub, which integrates SAP functionality within a Unspent Transaction Output (UTXO)-based ZKP mixer framework. StealthHub leverages mixer pools built on the UTXO model, which offer both strong privacy guarantees and inherent concurrency benefits, while employing a single key pair per user to minimise communication overhead. It also preserves the strengths of SAP, notably its support for flexible assets—i.e., not limited to fixed-denomination native token transfers—and its robust cross-*entity* transaction capabilities. Consistent with SOA principles, StealthHub is decomposed into four modular, service-oriented components (Fig. 1): Prover-as-a-Service (PaaS), for generating computationally intensive ZKPs; Relayer-as-a-Service (RaaS), for enabling anonymous interactions with StealthHub contracts; Fuzzy Message Detection (FMD)-as-a-Service (FaaS), which asynchronously scans ledger events, decrypts relevant ciphertexts, and notifies users; and a back-end system comprising smart contracts deployed on a DLT network, which implement StealthHub's core business logic.

By modularising these functions into loosely coupled services with defined interfaces and integrating them via a unified gateway as the entry point for user requests, StealthHub supports interoperability and reuse across existing Enterprise Service Bus (ESB) and Business Process Model and Notation (BPMN) engines.

Our contributions include formal specification (Sec. III) and practical Ethereum Virtual Machine (EVM)-based implementation (Sec. IV) of StealthHub. We provide three protocol variants—Incremental Merkle Tree (IMT) [10]-based StealthHub-IMT (SH-I), Merkle Mountain Range (MMR) [11]-based StealthHub-MMR (SH-M), and aggregated batch-processing StealthHub-AGG (SH-A)—and compare their on-chain gas usage and off-chain timings for SAP-specific prepare and scan *announcements* processes, demonstrating efficiency improvements compared to existing solutions. Additionally, we evaluate the complexity of ZK circuits implemented in Circom [12], benchmarked using Groth16 [13] via SnarkJS [14], offering insights into on-chain and off-chain computational trade-offs. Finally, we compare five ZK-friendly hash functions—including our implementations of Poseidon2 [15], Neptune [16], and GMiMC [17]—providing runtime and memory metrics during setup and proving phases as references for ZK developers.

II. Related Work

SAPs were initially proposed in 2011 by a Bitcoin Forum member, using Elliptic Curve Diffie-Hellman (ECDH) key exchanges to generate one-time addresses [18]. Nicolas van Saberhagen later formalised the DKSAP in 2013, introducing separate scanning and spending keys for secure blockchain transaction monitoring [19]. Subsequent research primarily diverged into two directions: performance-focused enhancements using pairing-based cryptography [20–22], and security-focused, post-quantum approaches employing lattice-based cryptography and homomorphic encryption [23–25]. However, research on the privacy limitations of Stealth Address Protocol (SAP) remains scarce [9], and parsing stealth addresses continues to incur substantial computational overhead, particularly due to the additional viewing key introduced in post-2014 DKSAP variants. Umbra [26], the leading SAP implementation on EVM-based blockchains and the baseline for our evaluation, employs such dual-key schemes.

In parallel, mixers have emerged as prevalent blockchain privacy solutions, initially deployed on Bitcoin [27]. Inspired by ZKPs, as employed in privacy-preserving frameworks such as Zerocash [28], ZKP-based mixers obfuscate transactional linkages. Incentivised mixers like AMR [29] and Tornado Cash [4] have further promoted adoption. While these approaches reduce linkability, they remain limited in supporting cross-*entity* interactions, flexible amounts, and non-native assets—prompting our design of the StealthHub architecture.

Finally, recent ZKP roll-ups [30, 31], rooted in Plasma-style Layer 2 designs [32], influence our SH-I architecture. While Layer 2 solutions improve Layer 1 throughput, they often rely on centralised sequencers. Guo et al. [33] adopt a similar "sequencer" role in Tornado Cash [4] to reduce on-chain costs. Wang et al. [34] shift the sequencer function to users, mitigating central trust but increasing confirmation latency. In contrast, StealthHub supports only user-side batching, where users independently aggregate commitments off-chain based on their UTXO spending needs, then submit a single commitment on-chain (Sec. III-C).

Scheme $\mathcal{S}_{\mathrm{PKE}}$

Preliminaries: A public-key encryption scheme generates a key pair $(\mathrm{pk}, \mathrm{sk})$, where pk encrypts a message m into a ciphertext ct decryptable only with sk, ensuring confidentiality.

Algorithms:

- $\underline{\mathrm{Encrypt}(\mathrm{pk}, m) \to ct}$: Encrypts m using pk to output ct.
- $\underline{\mathrm{Decrypt}(\mathrm{sk}, ct) \to \{m, \perp\}}$: Decrypts ct; outputs $\perp$ if fails.

Fig. 2: Public-key encryption scheme $\mathcal{S}_{\mathrm{PKE}}$.

Scheme $\mathcal{S}_{\mathrm{AE}}$

Preliminaries: Let $v \in \mathcal{D}$ be asset metadata of arbitrary length, and $c, \mu \in \{0,1\}^{\lambda}$ a commitment and its encoding under security parameter $\lambda \in \mathbb{N}$. Let $\sigma \in \{0,1\}^{\nu}$, with $\nu < \lambda$, denote the asset's fixed-length encoded suffix. Define:

$$f : v \to \sigma \text{ (injective encoding)}$$

$$f^{-1} : \sigma \to v \cup \{\perp\} \text{ (decoding with error symbol)}$$

Algorithms:

- $\underline{\mathrm{Encode}(v, c) \to \mu}$:
  1) Computes the suffix of the encoding commitment: $\sigma \leftarrow f(v)$.
  2) Truncates the first $\lambda - \nu$ bits of c: $c' \leftarrow c[:\lambda - \nu]$.
  3) Generates the encoded commitment: $\mu \leftarrow c' \parallel \sigma$.
- $\underline{\mathrm{Decode}(\mu) \to v}$:
  1) Generates σ by parsing $\mu \leftarrow c' \parallel \sigma$ where $|c'| = \lambda - \nu$.
  2) Decodes $v \leftarrow f^{-1}(\sigma)$; outputs $\perp$ if fails.

Fig. 3: Asset encoding scheme $\mathcal{S}_{\mathrm{AE}}$.

III. StealthHub: Protocol Description

StealthHub involves three key roles: *payer*, *payee*, and *tumbler*. *Payer* set ($addr_{\mathrm{payer}} \subseteq addr_{\mathrm{user}}$) includes *users* anonymously transferring cryptocurrency via StealthHub. *Payee* set ($addr_{\mathrm{payee}} \subseteq addr_{\mathrm{user}}$) comprises users anonymously receiving assets through key pairs $(\mathrm{pk}, \mathrm{sk})$. *Tumbler* ($addr_{\mathrm{tumbler}}$), implemented as smart contracts deployed on a DLT network depicted in Fig. 1, acts as a privacy-preserving intermediary whose security depends on the underlying DLT consensus.

A. Design Goals

StealthHub aims to achieve three primary design goals: cross-entity capability, flexibility, and high concurrency.

a) Cross-Entity Capability: Typical UTXO-based privacy schemes require out-of-band communication of a secret *note* s. *Payer* submits a commitment $c = H(s)$ to *tumbler* as proof of deposit, and subsequently, *payee* must prove knowledge of s to *tumbler* to withdraw the corresponding funds, where $H(\cdot)$ denotes a cryptographic hash function. StealthHub addresses this issue via a simplified public-key encryption scheme $\mathcal{S}_{\mathrm{PKE}}$ (Fig. 2). *Payer* encrypts a generated secret *note* $s = r \parallel N$ under *payee*'s public key pk, where r is a random nonce, and N is a nullifier to prevent double-spending. The ciphertext is generated as $ct \leftarrow \mathcal{S}_{\mathrm{PKE}}.\mathrm{Encrypt}(\mathrm{pk}, s)$. Both

Protocol $\Pi_{\text{StealthHub}}$

Preliminaries: UTXO $u = (c, v)$ with global set $\mathbb{U}$, and commitment $c = H(s)$ is computed by hashing a secret preimage s, and $v \in \mathbb{R}^+$ denotes the token amount. $addr_{\text{payer}}, addr_{\text{payee}}, addr_{\text{relayer}}$ denote address sets of users, with $addr_{\text{relayer}} \cap addr_{\text{payer}} = \emptyset$ and $addr_{\text{relayer}} \cap addr_{\text{payee}} = \emptyset$. Let $\mathbb{U}_a, \mathbb{U}_b, \mathbb{U}_c, \mathbb{U}_d$ denote UTXO sets corresponding to deposit outputs, shielded inputs, shielded outputs, and withdrawals, respectively, with no assumption on their sizes. Let f_s and f_w denote the total fees for shielded transfers and withdrawals, respectively, including gas costs, and let π_s and π_w denote the corresponding ZKPs. Let v_w denote the total amount of tokens withdrawn.

Deposit (tx_{dep}) with Π_{Deposit} (Fig. 5):
- From $addr_{\text{payer}}$:
  - Generates $\mathbb{U}_a = \{u_{a\,i}\}_i$, where $u_{a\,i} = (H(s_{a\,i}), v_{a\,i})$.
  - Submits $tx_{\text{dep}} \ni \mathbb{U}_a$.
- Updates $\mathbb{U} \leftarrow \mathbb{U} \cup \mathbb{U}_a$.

Shielded Transfer (tx_{xfer}) with $\Pi^{(\text{multi})}_{\text{ShieldedXfer}}$ (Fig. 9):
- Processes $\mathbb{U}_b = \{u_{b\,i}\}_i \subseteq \mathbb{U}$ to generate $\mathbb{U}_c = \{u_{c\,i}\}_i$:
  - Value conservation: $\sum v_{b\,i} == \sum v_{c\,i} + f_s$.
  - Generates π_s such that

$$\pi_s \vdash \left(\bigwedge u_{b\,i} \in \mathbb{U}\right) \wedge \left(\sum v_{b\,i} == \sum v_{c\,i} + f_s\right).$$

- Via $addr_{\text{relayer}}$:
  - Submits $tx_{\text{xfer}} \supset \{\mathbb{U}_b, \mathbb{U}_c, \pi_s, f_s\}$.
- Updates: $\mathbb{U} \leftarrow \mathbb{U} \setminus \mathbb{U}_b$, $\mathbb{U} \leftarrow \mathbb{U} \cup \mathbb{U}_c$.

Withdrawal (tx_{wd}) with Π_{Withdraw} (Fig. 8):
- Redeems $\mathbb{U}_d = \{u_{d\,i}\}_i \subseteq \mathbb{U}$:
  - Value conservation: $\sum v_{d\,i} == v_w + f_w$.
  - Specifies $addr_{\text{payee}}$ as a public input to π_w.
  - Generates π_w such that

$$\pi_w \vdash \left(\bigwedge (u_{d\,i} \in \mathbb{U}) \wedge (c_{d\,i} == H(s_{d\,i}))\right) \wedge \left(\sum v_{d\,i} == v_w + f_w\right).$$

- Via $addr_{\text{relayer}}$:
  - Submits $tx_{\text{wd}} \supset \{\mathbb{U}_d, \pi_w, f_w, addr_{\text{payee}}\}$.
- Updates: $\mathbb{U} \leftarrow \mathbb{U} \setminus \mathbb{U}_d$.

Fig. 4: StealthHub protocol $\Pi_{\text{StealthHub}}$.

Protocol Π_{Deposit}

The *payer* ($addr_{\text{payer}}$) creates a UTXO $u = (c, v)$ (see Fig. 4) and initiates the deposit transaction tx_{dep} to the *tumbler* ($addr_{\text{tumbler}}$). The *payer* uses the *payee*'s public key (pk_{payee}) to encrypt the secret s into the ciphertext ct, which is then transmitted to the *payee* via smart contract events.

Payer Deposit (External Transfer):
- Generates secret $s = r \| N$ with a random nonce r and nullifier N.
- Computes commitment $c = H(s)$.
- Creates deposit transaction tx_{dep} containing:
  - $ct \leftarrow \mathcal{S}_{\text{PKE}}.\text{Encrypt}(\text{pk}_{\text{payee}}, s)$, where s is the generated secret.
  - $u = (c, v)$, where c is a commitment and v a token amount.
- Sends $tx_{\text{dep}} \supset \{ct, u\}$ from $addr_{\text{payer}}$ to $addr_{\text{tumbler}}$.

Tumbler Processes Deposit:
- On receiving tx_{dep}:
  - Extracts asset metadata v and commitment c.
  - Computes commitment-ready value $\mu \leftarrow \mathcal{S}_{\text{AE}}.\text{Encode}(v, c)$.
- Updates cryptographic accumulator:
  - Executes $\text{acc}_S \leftarrow \mathcal{S}_{\text{Acc}}.\text{Accumulate}(\mu)$.
- Publishes log event: $\text{DepositEvent}(ct, \mu, \text{acc}_S)$.

Fig. 5: Deposit protocol Π_{Deposit}.

Scheme $\mathcal{S}_{\text{Acc}}$

Preliminaries: A cryptographic accumulator enables a party to compile a set $S = \{e_1, \ldots, e_n\}$ into a succinct digest acc_S, with which one can efficiently prove membership or non-membership of an element e in S by generating a compact witness ω without revealing the entire set S.

Algorithms:
- $\underline{\text{Accumulate}(S) \to \text{acc}_S}$: Computes acc_S for S.
- $\underline{\text{Prove}(\text{acc}_S, e) \to \omega}$: Generates witness $\omega \vdash e \in S$.
- $\underline{\text{Verify}(\text{acc}_S, e, \omega) \to \{\text{true}, \text{false}\}}$: Checks ω and outputs true if valid; otherwise false.

Fig. 6: Accumulator scheme $\mathcal{S}_{\text{Acc}}$.

the commitment $c = H(s)$ and ciphertext ct are published on-chain. *Payee* decrypts ct with their private key sk, retrieves $s = r \| N$, and verifies $c = H(s)$, thus confirming ownership of the asset.

b) Flexibility: Traditional UTXO ZKP mixers [4] impose fixed denominations and single-asset constraints, restricting use-cases such as Non-Fungible Token (NFT). StealthHub allows cross-entity transfers and arbitrary asset types and amounts using asset encoding scheme $\mathcal{S}_{\text{AE}}$ (Fig. 3). During deposits, smart contracts compute an encoded commitment $\mu \leftarrow \mathcal{S}_{\text{AE}}.\text{Encode}(v, c)$, where μ represents the commitment containing asset information encoded by *tumbler* and recorded on-chain. During withdrawals or shielded transfers, smart contracts decode $v \leftarrow \mathcal{S}_{\text{AE}}.\text{Decode}(\mu)$, facilitating arbitrary asset transfers without compromising anonymity.

c) High Concurrency: By leveraging the UTXO model, StealthHub[1] (Fig. 4) supports concurrent deposits, withdrawals, and shielded transfers. Unlike account-based SAPs requiring embedded native token fees per transaction, StealthHub simplifies transaction fee management: users only maintain sufficient concealed native tokens within *tumbler* to cover intermediary relayer fees, ensuring efficient concurrent transaction execution involving native or non-native tokens (ERC-20, NFTs).

B. Transaction Types

a) Deposit: Initiated by *payer*, who deposits via their own public key or any published recipient key, assets move from *payer*'s blockchain address into *tumbler*'s privacy pool. The deposit transaction tx_{dep} (Fig. 5) utilises cryptographic accumulators and zk-SNARK, as shown in Fig. 6 and Fig. 7, respectively[2]. Instantiations using IMT [10], MMR [11], and Groth16 [13] are later demonstrated in Fig. 14 and Tab. III in Sec. IV. Resulting ciphertext ct, emitted via contract events, enables users to scan and decrypt privately.

b) Withdraw: Initiated by *payees* holding concealed assets and executed via relayers. According to Fig. 8, after

[1]In Fig. 4 and the subsequent Fig. 5, Fig. 8, and Fig. 9, transaction descriptions focus solely on token amount computations, omitting components such as token ID for simplicity. Commitment generation and verification in Fig. 4 omit token amount encoding / decoding, which are detailed in the subsequent protocols. Deposit fees are excluded, as deposits are directly initiated by payers rather than relayers ($addr_{\text{relayer}}$). Figures for withdrawal and shielded transfer protocols also omit explicit relayer details for brevity.

[2]In this work, we use $\dashv$ / $\vdash$ to denote provability: if Γ is a set of axioms and φ a formula, $\Gamma \vdash \varphi$ means there is a formal proof of φ from Γ. 1^λ denotes the unary encoding of the security parameter λ. $\cdot$ denotes omitted public parameters in the sub-protocols.

Protocol $\mathcal{P}_{\text{zkSNARK}}$

Preliminaries: zk-SNARKs allow a prover to convince a verifier that a statement x and a witness w satisfy an NP relation $\mathcal{R}$ (i.e., $(x, w) \in \mathcal{R}$) without revealing w. The proof π consists of a single message, which the verifier checks independently. A setup algorithm generates a trusted common reference string crs to pre-process $\mathcal{R}$, reducing the verifier's workload.

Algorithms:

- $\text{Setup}(1^\lambda, \mathcal{R}) \to \text{crs}$: Outputs common reference string crs for security parameter $\lambda \in \mathbb{N}$ and NP relation $\mathcal{R}$.
- $\text{Prove}(\text{crs}, x, w) \to \pi$: Generates proof $\pi \vdash (x, w) \in \mathcal{R}$.
- $\text{Verify}(\text{crs}, x, \pi) \to \{\text{true}, \text{false}\}$: Checks π under crs and outputs true if valid; otherwise false.

Fig. 7: zk-SNARK protocol $\mathcal{P}_{\text{zkSNARK}}$.

Protocol Π_{Withdraw}

The payee ($addr_{\text{payee}}$), holding the private key (sk_{payee}) and monitoring deposit events, proves the ability to spend the UTXO $u = (c, v) \in \mathbb{U}$ by constructing a ZKP of ownership (see Fig. 4).

Payee Prepares Withdrawal:

- For $\text{DepositEvent}(ct, \mu, \text{acc}_S)$:
  - Recovers asset metadata $v \leftarrow \mathcal{S}_{\text{AE}}.\text{Decode}(\mu)$.
  - Decrypts $s \leftarrow \mathcal{S}_{\text{PKE}}.\text{Decrypt}(\text{sk}_{\text{payee}}, ct)$.
  - Verifies $\mu[:\lambda-\nu] == H(s)[:\lambda-\nu]$.
- Generates membership witness ω for $\mu \in \text{acc}_S$:
  - Computes $\omega \leftarrow \mathcal{S}_{\text{Acc}}.\text{Prove}(\text{acc}_S, \mu)$.
- Constructs ZKP $\pi_w \leftarrow \mathcal{P}_{\text{zkSNARK}}.\text{Prove}(\cdot)$ attesting:
  - Knowledge of s.
  - Checks $\text{true} \leftarrow \mathcal{S}_{\text{Acc}}.\text{Verify}(\text{acc}_S, \mu, \omega)$.
  - Checks metadata decoding $v == \mathcal{S}_{\text{AE}}.\text{Decode}(\mu)$.
- Creates withdrawal transaction tx_{wd} containing:
  - π_w, v, recipient address $addr_{\text{payee}}$.
- Sends $tx_{\text{wd}} \supset \{\pi_w, v, addr_{\text{payee}}\}$ from $addr_{\text{relayer}}$ to $addr_{\text{tumbler}}$.

Tumbler Processes Withdrawal:

- On receiving tx_{wd}:
  - Checks $\text{true} \leftarrow \mathcal{P}_{\text{zkSNARK}}.\text{Verify}(\pi_w, \cdot)$.
- If verification succeeds:
  - Transfers asset specified by v to $addr_{\text{payee}}$.

Fig. 8: Withdrawal protocol $\Pi_{\text{Withdrawal}}$.

tumbler verification, relayers release assets from the privacy pool to *payee*'s blockchain address.

c) Shielded Transfer: Initiated by *payers* holding shielded assets, executed via relayers, shielded transfers resemble deposits and withdrawals. However, rather than exiting the pool, assets are reassigned internally to the same or another user. Fig. 9 describes executing multiple internal UTXO transfers within a single blockchain transaction.

C. Batch-Efficient Instantiation

In UTXO-based add-on schemes, individual on-chain state updates are costly (see Tab. III in Sec. IV). StealthHub mitigates this by employing off-chain batching and ZK-verified aggregated state updates—a design approach previously explored in efforts to reduce Tornado Cash [4]'s on-chain costs [33, 34]. We employ two Merkle trees: an on-chain *era*-tree of height h_{era}, which tracks state transitions, and off-chain *slot*-trees of height h_{slot}, which aggregate user transactions. Each leaf of the *era*-tree serves as the root of a corresponding *slot*-tree, forming a composite structure with

Protocol $\Pi^{(\text{multi})}_{\text{ShieldedXfer}}$

The payer $(addr_{\text{payer}}, \text{pk}_{\text{payer}}, \text{sk}_{\text{payer}})$ proves ownership of a set of UTXOs $\mathbb{U}_b = \{u_{b_i}\}_i \subseteq \mathbb{U}$ (see Fig. 4), and constructs a new set of UTXOs $\mathbb{U}_c = \{u_{c_i}\}_i$ for recipients $\{\text{pk}_{\text{payee}_i}\}_i$, such that each u_{c_i} corresponds in order and cardinality to $\text{pk}_{\text{payee}_i}$, with the associated ciphertexts ct_{c_i}.

Multi-Asset Redemption & Partitioning:

- For $\{\text{DepositEvent}(ct_{b_i}, \mu_{b_i}, \text{acc}_S)\}_i$:
  - Recovers $s_{b_i} \leftarrow \mathcal{S}_{\text{PKE}}.\text{Decrypt}(\text{sk}_{\text{payer}}, ct_{b_i})$.
  - Verifies $\mu_{b_i}[:\lambda-\nu] \in \text{acc}_S == H(s_{b_i})[:\lambda-\nu]$.
  - Generates membership witness ω_{b_i} for $\mu_{b_i} \in \text{acc}_S$:
    - Computes $\omega_{b_i} \leftarrow \mathcal{S}_{\text{Acc}}.\text{Prove}(\text{acc}_S, \mu_{b_i})$.
- Atomic consumption ($\mathbb{U}_b$) and redistribution ($\mathbb{U}_c$):
  - Computes $v_{b_i} \leftarrow \mathcal{S}_{\text{AE}}.\text{Decode}(\mu_{b_i})$.
  - Allocates new values v_{c_i} such that $\sum v_{c_i} < \sum v_{b_i}$.
  - Computes $c_{c_i} \leftarrow H(s_{c_i})$.
  - Encrypts $ct_{c_i} \leftarrow \mathcal{S}_{\text{PKE}}.\text{Encrypt}(\text{pk}_{\text{payee}_i}, s_{c_i})$.
- Constructs ZKP $\pi_s \leftarrow \mathcal{P}_{\text{zkSNARK}}.\text{Prove}$ attesting:
  - Knowledge of s_{c_i}.
  - Checks $\text{true} \leftarrow \bigwedge \mathcal{S}_{\text{Acc}}.\text{Verify}(\text{acc}_S, \mu_{b_i}, \omega_{b_i})$.
  - Checks metadata decoding $v_{c_i} == \mathcal{S}_{\text{AE}}.\text{Decode}(\mu_{c_i})$.
- Creates atomic transfer tx_{xfer} containing:
  - ct_{c_i}, c_{c_i}, v_{c_i}, and π_s.
- Sends $tx_{\text{xfer}} \supset \{ct_{c_i}, c_{c_i}, v_{c_i}, \pi_s\}$ from $addr_{\text{relayer}}$ to $addr_{\text{tumbler}}$.

Tumbler Processes Atomic Shielded Transfer:

- On receiving tx_{wd}:
  - Checks $\text{true} \leftarrow \mathcal{P}_{\text{zkSNARK}}.\text{Verify}(\pi_s, \cdot)$.
- If verification succeeds:
  - Encodes $\mu_{c_i} \leftarrow \mathcal{S}_{\text{AE}}.\text{Encode}(v_{c_i}, c_{b_i})$.
  - Updates $\text{acc}_S \leftarrow \mathcal{S}_{\text{Acc}}.\text{Accumulate}(\mu_{c_i})$.
- Emits $\text{TransferEvent}^{(\text{multi})}(ct_{c_i}, \mu_{c_i}, \text{acc}_S)$.

Fig. 9: Shielded transfer (multiple) protocol $\Pi^{(\text{multi})}_{\text{ShieldedXfer}}$.

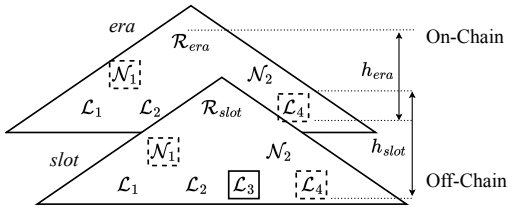


Fig. 10: An example with $h_{\text{era}} = h_{\text{slot}} = 2$, resulting in a combined tree height of $h_{\text{era-slot}} = 3$. $\mathcal{R}$, $\mathcal{N}$, and $\mathcal{L}$ denote the root node, intermediate nodes, and leaf nodes, respectively. The nodes enclosed in dashed boxes represent the Merkle hash path required to verify the membership of leaf node $\mathcal{L}_3$.

height $h_{\text{era-slot}} = h_{\text{era}} + h_{\text{slot}} - 1$. The level containing both the *slot*-tree roots and the *era*-tree leaves is referred to as the overlap layer (Fig. 10). Further implementation details are provided in Sec. IV.

IV. Experiments

We deployed implementations of SH-I (IMT as accumulator per Fig. 6), SH-M (MMR as accumulator), and SH-A (MMR as accumulator, with pre-aggregated commitments per Sec. III-C) on four EVM-compatible DLTs networks, comprising Solidity-based *tumbler* contracts and a JavaScript client: Ethereum [35], Hedera [36], XRP Ledger (XRPL) EVM Sidechain [37], and BNB Chain [38]. These initial deployments illustrate the broad applicability of StealthHub

TABLE I: Umbra vs. StealthHub per-step and timings (ms). Steps marked with "-" indicate no computational complexity difference between the two and are thus not listed with improvement percentages, yet are included in the total.

Prepare Announcement			
Step	Umbra	StealthHub	Improvement (%)
KeyPair creation (dual. vs. single.)	0.1200	0.0126	89.5
Rand. number generation	0.2040	0.0189	-
Rand. number encryption	4.9745	2.4765	50.2
Compute (*pk* & *addr.* vs. *SHA256*)	3.4541	0.0258	99.3
Total	8.7526	2.5338	71.1
Scan Announcement			
Step	Umbra	StealthHub	Improvement (%)
Batch retrieval	5.4799	5.3849	-
Get uncompressed key	0.1268	0.1146	-
Decrypt payload	2.5631	2.4868	-
Compute (*addr.* vs. *SHA256*)	2.1966	0.0149	99.3
Comparison	0.0079	0.0001	98.7
Total	10.3743	7.0013	32.5

across existing EVM-centric DLTs [39]. Our code is open-sourced at https://github.com/hanzeG/StealthHub.

a) SAP Announcement Detecting: We first evaluated StealthHub's scanning efficiency compared to dual-key-based SAP approaches. As summarised in Tab. I, the UTXO structure eliminates computational overhead from one-time stealth address generation (*addr.*) and public key compression (*pk*). Additionally, employing the efficient xxHash library [40] (*SHA256*) allowed StealthHub to reduce computational time for prepare and scan *announcement* phases by over 71% and 32%, respectively.

b) ZK Circuits: We implemented the corresponding Rank-1 Constraint System (R1CS) circuits in Circom [12], using Groth16 [13] as an instance of the zk-SNARK described in Fig. 7, and SnarkJS [14] for proof generation. We then analysed circuit complexity by measuring non-linear constraints for five ZK-friendly hash functions with input length two. Neptune required 228 constraints; Poseidon and Poseidon2 each had 240; GMiMC needed 678; MiMC required 1,320. For an *era-slot* tree of height $h_{\mathrm{era-slot}}$ and batch size b, the total number of constraints is given by $C = num \cdot h_{\mathrm{era-slot}} \cdot b$, where num denotes the number of constraints required for a single invocation of the hash function. At $b = 2^{10}$ and $h_{\mathrm{era-slot}} = 40$, Neptune, Poseidon, and Poseidon2 yielded between 2^{22} and 2^{23} constraints, while MiMC and GMiMC required significantly more—between 2^{24} and 2^{25} (Fig. 12).

Benchmarking prover max RSS on full binary-tree circuits of heights 1–8 (1–255 hash computations) over five runs (on M1 Pro CPU, 16 GB RAM, 10 cores), we normalised Poseidon2's single-execution metric to M_0. At the largest circuit size, setup runtime was approximately $40\,M_0$, representing 97%, 20%, and 73% runtime reductions compared to GMiMC, MiMC, and Neptune, respectively. Proof-generation runtime at the same scale was about $10\,M_0$, yielding savings of 91%, 62%, and 16% compared to GMiMC, MiMC, and Neptune. Regarding peak memory usage, Poseidon2 consumed roughly $4.3\,M_0$ (setup phase in Fig. 7) and $2.9\,M_0$ (prove phase in Fig. 7), compared to Neptune's $9.9\,M_0$ and $3.7\,M_0$. Consequently, Poseidon2 was selected for the MMR and overall StealthHub implementation.

TABLE II: Gas costs of deployment transactions.

Scheme	Height	Cost	MMR Improvement (%)
MMR	-	1,104,295	-
IMT	4	1,416,057	22.0
	8	1,506,107	26.7
	16	1,687,311	34.5
	31	2,031,036	46.2

TABLE III: On-chain cost under different instantiations.

Scheme	p.t.	SH-I (h = 31)		SH-M		SH-A ($b = 2^5$)	wit.
		dep.	s.t.	dep.$^{\Psi}$	s.t.$^{\Psi}$	dep./s.t.†	
Gas (10^3)	21	1,042	2,077	311	744	16 = 509/*b*	255
Percentage† (%)	100	4,962	9,890	1,481	3,543	76	1,214
USD ($)	0.68	33.88	67.52	10.11	24.19	0.52	8.29

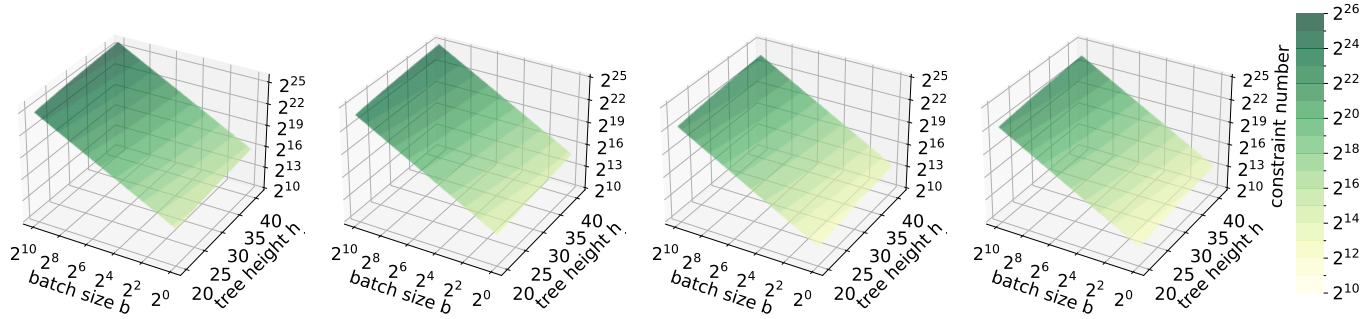
† Gas cost percentage of the plain transfer (p.t.).
$^{\Psi}$ Evaluated the first 2^{16} deposit (dep., Fig. 5) and shielded transfer (s.t., Fig. 9) transactions due to resource limits; averaged values used as reference. Withdrawal (wit., Fig. 8) cost remains constant.
‡ The amortized gas cost per transaction in the batch (b = 32), based on the first 2^{16} transactions, with the average value calculated accordingly.

c) On-chain Gas Costs: We benchmarked on-chain costs by comparing MMR against an IMT baseline. Contract deployment gas (Tab. II) indicated MMR reduced deployment costs by approximately 46.2% relative to IMT height 31 and by over 22% compared to height 4. For insertions, Fig. 13 illustrates average gas consumption for the first 2^{12} and 2^{16} leaves. MMR required 2.43×10^5 and 3.04×10^5 gas respectively, whereas IMTs of heights 12, 16, and 31 consumed significantly more—3.83×10^5, 4.99×10^5, and 9.60×10^5. Thus, MMR achieved gas reductions of over 36%, 39%, and 68% compared to IMTs of heights 12, 16, and 31 for the first 2^{16} insertions.

Building on Sec. III, we then implemented the *tumbler* prototype, excluding the asset encoding scheme due to unresolved security. This is left to future work, potentially informed by systems like Railgun [41]. Furthermore, as the EVM-compatible networks, i.e., XRPL EVM Sidechain, BNB Chain, and Hedera, share the same gas accounting model when calling smart contract functions—due to their adherence to EVM semantics—differences in actual on-chain costs stem solely from platform-specific pricing[3]. We therefore focus our analysis on Ethereum as a representative case. We measured gas costs for SH-I (an IMT of height $h = 31$), SH-M (using an MMR), and SH-A (an era-slot tree with MMR eras and IMT slots of height $h = 5$), all using Poseidon2. As shown in Tab. III[4], the first 2^{16} deposit and shielded transfer transactions in SH-M cost approximately 311K and 744K gas, respectively. In SH-A, which aggregates UTXO insertions and verifies ZK proofs on-chain, each such transaction consumed roughly

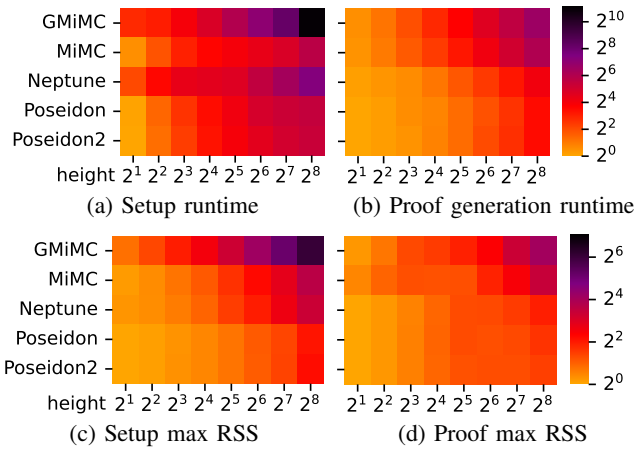
[3]Hedera differs from Ethereum in gas accounting for contract deployment by incorporating storage duration. For opcodes like *LOG* and *SSTORE*, it applies a rent-like model, unlike Ethereum's fixed-cost approach.

[4]At the time of writing: 1 ETH = 3,251 USD, mainnet gas price = 10 Gwei.



(a) MiMC (b) GMiMC (c) Poseidon & Poseidon2 (d) Neptune

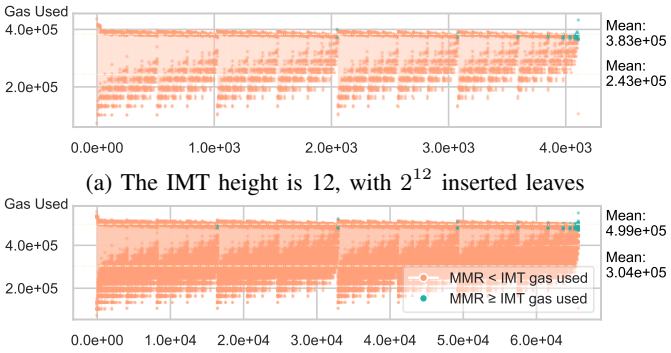
Fig. 11: Constraints number in Circom circuits (from SnarkJS) vs. Merkle tree height h and batch size b.



(a) Setup runtime (b) Proof generation runtime

(c) Setup max RSS (d) Proof max RSS

Fig. 12: Metrics of hash functions in SnarkJS Groth16.

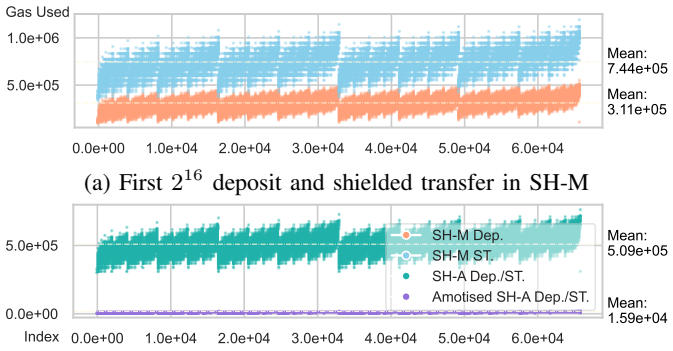


(a) The IMT height is 12, with 2^{12} inserted leaves

(b) The IMT height is 16, with 2^{16} inserted leaves

Fig. 13: Gas costs for leaf node insertions in MMR and IMT.

509K gas. With a batch size $b = 2^5$ (i.e., the number of leaf nodes in a *slot* tree of height $h_{\text{slot}} = 5$), SH-A's amortized per-transaction cost dropped to about 16K gas, below the 21K gas of a standard transfer. Withdrawals incurred a fixed cost of 255K gas due to a constant proof size and the absence of tree updates (Fig. 14). Thus, SH-A achieves deposit and shielded transfer gas costs around \$0.52 per transaction, while withdrawal costs remain approximately 12 times that of a standard transfer—a reasonable trade-off, as a single ZK proof can withdraw multiple UTXOs (Tab. III).



(a) First 2^{16} deposit and shielded transfer in SH-M

(b) First 2^{16} deposit and shielded transfer and the amortised in SH-A

Fig. 14: Gas consumption on Ethereum.

V. Conclusion and Future Work

We introduced StealthHub, a UTXO-based SAP, instantiated in three variants: SH-I, SH-M, and SH-A. Our evaluation demonstrates consistently low computational overhead, both on-chain and off-chain, across a variety of representative workloads. Future work includes providing a formal security proof, integrating StealthHub into more advanced Decentralised Finance (DeFi) workflows [29] (e.g., token swaps [1, 42], lending [43], and yield farming [44, 45]), and assessing the robustness and scalability of its batch-processing design under realistic usage conditions. A major focus is to quantitatively evaluate the privacy properties of existing blockchain privacy protocols, drawing inspiration from prior studies on Tornado Cash [4, 5] and Umbra [9, 26]. We aim to conduct a comprehensive privacy comparison using graph-theoretic [46, 47] and information-theoretic [9, 48, 49] methods, in order to rigorously evaluate and further enhance StealthHub's potential to provide strong privacy guarantees. These efforts are intended to contribute to a unified and generalizable theoretical framework for analyzing add-on privacy mechanisms in DLTs.

Acknowledgements

This work was supported in part by the Ethereum Foundation through the ZK Grant Round under Grant ID: FY24-1503, the National Natural Science Foundation of China under Grant No. 62472255, the Natural Science Foundation of Qingdao, China under Grant No. 23-2-1-152-zyyd-jch, and Ripple's University Blockchain Research Initiative (UBRI) [50].

References

[1] J. Xu, K. Paruch, S. Cousaert, and Y. Feng, "SoK: Decentralized Exchanges (DEX) with Automated Market Maker (AMM) Protocols," *ACM Computing Surveys*, vol. 55, no. 11, 2 2023. [Online]. Available: /doi/pdf/10.1145/3570639?download=true

[2] G. Goodell, D. R. Toliver, and H. D. Nakib, "A Scalable Architecture for Electronic Payments," in *International Conference on Financial Cryptography and Data Security*, vol. 13412 LNCS. Springer, 2023, pp. 645–678. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-32415-4_38

[3] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais, "SoK: Decentralized Finance (DeFi) Attacks," in *Symposium on Security and Privacy (SP)*, vol. 2023-May. IEEE, 2023, pp. 2444–2461.

[4] "tornadocash/tornado-core: Tornado cash. Non-custodial private transactions on Ethereum." [Online]. Available: https://github.com/tornadocash/tornado-core

[5] Z. Wang, S. Chaliasos, K. Qin, L. Zhou, L. Gao, P. Berrang, B. Livshits, and A. Gervais, "On How Zero-Knowledge Proof Blockchain Mixers Improve, and Worsen User Privacy," in *Web Conference (WWW)*. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3543507.3583217

[6] N. Bena, M. Pedrinazzi, M. Anisetti, O. Hasan, and L. Brunie, "A Transparent Certification Scheme Based on Blockchain for Service-Based Systems," in *International Conference on Web Services (ICWS)*. IEEE, 2024, pp. 501–511.

[7] J. Liu, C. A. Sun, T. Wu, and M. Aiello, "Blockchain-based Privacy-preserving Data Service Provisioning for Internet of Things," in *International Conference on Web Services (ICWS)*. IEEE, 2024, pp. 524–534.

[8] X. Du, X. Chen, Z. Lu, Q. Duan, Y. Wang, and J. Wu, "BIECS: A Blockchain-based Intelligent Edge Cooperation System for Latency-Sensitive Services," in *International Conference on Web Services (ICWS)*. IEEE, 2022, pp. 367–372.

[9] A. M. Kovács and I. A. Seres, "Anonymity Analysis of the Umbra Stealth Address Scheme on Ethereum," in *Web Conference (WWW)*. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3589335.3651963

[10] M. Szydlo, "Merkle Tree Traversal in Log Space and Time," in *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, vol. 3027. Springer, 2004, pp. 541–554. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-24676-3_32

[11] Peter Todd, "Merkle Mountain Ranges - Grin Documentation," 2017. [Online]. Available: https://docs.grin.mw/wiki/chain-state/merkle-mountain-range/

[12] "iden3/circom: zkSnark circuit compiler." [Online]. Available: https://github.com/iden3/circom

[13] J. Groth, "On the size of pairing-based non-interactive arguments," in *35th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, vol. 9666. Springer, 2016, pp. 305–326. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-662-49896-5_11

[14] "iden3/snarkjs: zkSNARK implementation in JavaScript & WASM." [Online]. Available: https://github.com/iden3/snarkjs

[15] L. Grassi, D. Khovratovich, and M. Schofnegger, "Poseidon2: A Faster Version of the Poseidon Hash Function," in *14th International Conference on Cryptology in Africa (AFRICACRYPT)*, vol. 14064 LNCS. Springer, 2023, pp. 177–203. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-37679-5_8

[16] L. Grassi, S. Onofri, M. Pedicini, and L. Sozzi, "Invertible Quadratic Non-Linear Layers for MPC-/FHE-/ZK-Friendly Schemes over Fnp: Application to Poseidon," *IACR Transactions on Symmetric Cryptology*, vol. 2022, no. 3, pp. 20–72, 9 2022. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/9849

[17] M. R. Albrecht, L. Grassi, L. Perrin, S. Ramacher, C. Rechberger, D. Rotaru, A. Roy, and M. Schofnegger, "Feistel Structures for MPC, and More," in *24th European Symposium on Research in Computer Security (ESORICS)*, vol. 11736 LNCS. Springer, 2019, pp. 151–171. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-29962-0_8

[18] ByteCoin, "Untraceable transactions which can contain a secure message are inevitable." 2011. [Online]. Available: https://bitcointalk.org/index.php?topic=5965.0

[19] N. Van Saberhagen, "CryptoNote v 2.0," 2013. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-46766-1_27

[20] J. Fan, J. Bai, Z. Wang, Y. Li, Y. Luo, and Y. Hao, "A new stealth address scheme for blockchain," in *Turing Celebration Conference-China*. ACM, 5 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3321408.3321573

[21] M. M. Miki´c and M. Srbakoski, "Elliptic Curve Pairing Stealth Address Protocols," 12 2023. [Online]. Available: https://arxiv.org/abs/2312.12131v4

[22] C. Feng, L. Tan, H. Xiao, X. Qi, Z. Wen, and Y. Liu, "EDKSAP : Efficient Double-Key Stealth Address Protocol in Blockchain," in *20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2021, pp. 1196–1201.

[23] M. M. Miki´c, M. Srbakoski, and S. Praška, "Post-Quantum Stealth Address Protocols," 1 2025. [Online]. Available: https://arxiv.org/abs/2501.13733v1

[24] Y. Yan, G. Shao, D. Song, M. Song, and Y. Jin, "HE-DKSAP: Privacy-Preserving Stealth Address Protocol via Additively Homomorphic Encryption," 12 2023. [Online]. Available: https://arxiv.org/abs/2312.10698v1

[25] Z. Liu, K. Nguyen, G. Yang, H. Wang, and D. S. Wong,

"A Lattice-Based Linkable Ring Signature Supporting Stealth Addresses," in *24th European Symposium on Research in Computer Security (ESORICS)*, vol. 11735 LNCS. Springer, 2019, pp. 726–746. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-29959-0_35

[26] "Umbra v2: Flexible, Interoperable, and More than Just Payments," 2024. [Online]. Available: https://scopelift.co/blog/introducing-umbra-v2-architecture

[27] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin," in *19th European Symposium on Research in Computer Security (ESORICS)*, vol. 8713 LNCS, no. PART 2. Springer, 2014, pp. 345–364. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-11212-1_20

[28] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Symposium on Security and Privacy (SP)*. IEEE, 11 2014, pp. 459–474.

[29] D. V. Le and A. Gervais, "AMR: Autonomous coin mixer with privacy preserving reward distribution," in *3rd Conference on Advances in Financial Technologies (AFT)*. ACM, 9 2021, pp. 142–155. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3479722.3480800

[30] "ZKsync Docs." [Online]. Available: https://docs.zksync.io/

[31] "Starknet documentation." [Online]. Available: https://docs.starknet.io/

[32] J. Poon and V. Buterin, "Plasma: Scalable Autonomous Smart Contracts," 2017. [Online]. Available: https://plasma.io/

[33] H. Guo, Y. Feng, C. Wu, Z. Li, and J. Xu, "Benchmarking ZK-Friendly Hash Functions and SNARK Proving Systems for EVM-compatible Blockchains," 9 2024. [Online]. Available: https://arxiv.org/pdf/2409.01976

[34] Z. Wang, M. Cirkovic, D. V. Le, W. Knottenbelt, and C. Cachin, "Pay Less for Your Privacy: Towards Cost-Effective On-Chain Mixers," 2023. [Online]. Available: https://eprint.iacr.org/2023/1222[39]

[35] "Ethereum (ETH) Blockchain Explorer." [Online]. Available: https://etherscan.io/

[36] "Hedera Dashboard." [Online]. Available: https://hashscan.io/mainnet/dashboard

[37] "XRPL Explorer | Ledgers." [Online]. Available: https://livenet.xrpl.org/

[38] "BNB Smart Chain (BNB) Blockchain Explorer." [Online]. Available: https://bscscan.com/

[39] Y. Luo, Y. Feng, J. Xu, and P. Tasca, "Piercing the Veil of TVL: DeFi Reappraised," in *Financial Cryptography and Data Security (FC)*. Springer, 4 2025. [Online]. Available: https://arxiv.org/pdf/2404.11745v4

[40] "Cyan4973/xxHash: Extremely fast non-cryptographic hash algorithm." [Online]. Available: https://github.com/Cyan4973/xxHash

[41] "RAILGUN Privacy System | Wiki." [Online]. Available: https://docs.railgun.org/wiki/learn/privacy-system

[42] J. Xu, D. Ackerer, and A. Dubovitskaya, "A game-theoretic analysis of cross-chain atomic swaps with HTLCs," in *41st International Conference on Distributed Computing Systems (ICDCS)*, vol. 2021-July. IEEE, 7 2021, pp. 584–594.

[43] J. Xu, Y. Feng, D. Perez, and B. Livshits, "Auto.gov: Learning-based Governance for Decentralized Finance (DeFi)," *IEEE Transactions on Services Computing*, pp. 1–14, 3 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10937069/

[44] J. Xu and Y. Feng, "Reap the Harvest on Blockchain: A Survey of Yield Farming Protocols," in *Transactions on Network and Service Management (TNSM)*, vol. 20, no. 1. IEEE, 3 2023, pp. 858–869.

[45] S. Cousaert, J. Xu, and T. Matsui, "SoK: Yield Aggregators in DeFi," in *International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2022.

[46] H. Fu, Y. Feng, C. Wu, and J. Xu, "Perseus: Tracing the Masterminds Behind Cryptocurrency Pump-and-Dump Schemes," 3 2025. [Online]. Available: https://arxiv.org/pdf/2503.01686

[47] H. Du, Z. Che, M. Shen, L. Zhu, and J. Hu, "Breaking the Anonymity of Ethereum Mixing Services Using Graph Feature Learning," in *Transactions on Information Forensics and Security (TIFS)*, vol. 19. IEEE, 2024, pp. 616–631.

[48] H. Lin, M. Ren, P. Barucca, and T. Aste, "Granger Causality Detection with Kolmogorov-Arnold Networks," 12 2024. [Online]. Available: https://arxiv.org/pdf/2412.15373

[49] D. M. Kelen and I. A. Seres, "Towards Measuring the Traceability of Cryptocurrencies," 11 2022. [Online]. Available: https://arxiv.org/pdf/2211.04259

[50] Y. Feng, J. Xu, and L. Weymouth, "University Blockchain Research Initiative (UBRI): Boosting blockchain education and research," *IEEE Potentials*, vol. 41, no. 6, pp. 19–25, 2022.