Obscura: Privacy-Preserving Protocol for the Algorand Blockchain Using LSAG Ring Signatures

Navid Azimi[1] and Seyedamin Pouriyeh[2]

[1] Emory University, Atlanta, GA 30322, USA
navid.azimi@emory.edu
[2] Kennesaw State University, Kennesaw, GA 30144, USA
spouriye@kennesaw.edu

Abstract. While public blockchains provide transparent and auditable transaction histories, they inherently compromise user privacy. Existing privacy-enhancing protocols, such as those deployed on Ethereum, typically rely on succinct zero-knowledge proofs (zk-SNARKs) to obscure the transaction graph. However, implementing comparable cryptographic guarantees on high-throughput blockchains like Algorand is challenging due to strict per-call execution budgets and the state contention introduced by global Merkle accumulators. This paper presents *Obscura*, a decentralized, non-custodial privacy protocol tailored for constrained smart contract environments. Obscura achieves transaction anonymity using Linkable Spontaneous Anonymous Group (LSAG) signatures over the BN254 elliptic curve, verified entirely on-chain. To overcome limitations of the Algorand Virtual Machine (AVM), we introduce a novel state model that leverages Algorand's Box Storage for $O(1)$ commitment membership checks, eliminating the need for global Merkle accumulators, and a dynamic opcode-budget expansion mechanism via pooled inner application calls. Our implementation demonstrates that signer-ambiguous privacy is practical and efficient on Algorand without relying on trusted setups or succinct proofs. Obscura provides a robust privacy layer for transparent ledgers, bridging the gap between high-throughput blockchain architectures and the dual requirements of cryptographic privacy and selective auditability.

Keywords: Algorand · Linkable Ring Signatures · LSAG · Blockchain Privacy · Anonymous Transactions · On-chain Verification · Smart Contracts

1 Introduction

The transparent nature of public blockchains [48] is fundamental to their security and audibility, yet it inherently undermines user privacy. This friction is especially pronounced on high-throughput blockchains such as Algorand [29], which are increasingly used for enterprise financial services and decentralized finance

(DeFi). In these settings, participants require confidentiality to protect sensitive transaction data, while still maintaining the ability to meet regulatory and audit requirements. Because all transactions are recorded on a public ledger, adversaries can trace the flow of funds, cluster addresses, and deanonymize users through transaction graph analysis [8, 43, 58]. To address this, privacy-preserving protocols and decentralized mixers have been developed [14, 18, 45]. On Ethereum [61], protocols such as Tornado Cash [53] successfully deployed privacy pools that sever the on-chain link between depositors and withdrawers. These systems typically rely on zk-SNARKs to prove membership in a Merkle tree [44] of deposits without revealing which specific fund is being spent. Related smart-contract designs include Möbius [42], which employs linkable ring signatures for trustless tumbling on Ethereum, and Zether [20], which provides confidential account balances.

However, the deployment of SNARK-based privacy pools relies heavily on complex cryptographic circuits and state-heavy global accumulators. Emerging high-throughput blockchains, such as Algorand, optimize for fast finality and predictable fees by imposing strict per-transaction execution budgets (opcode limits) and flat state models. While the Algorand Virtual Machine (AVM) supports native pairing operations, evaluating the multiple pairings required for traditional zk-SNARK verification (e.g., Groth16 [33]) consumes a massive portion of the pooled execution budget. Furthermore, SNARK-based mixers require maintaining a global Merkle tree, which introduces severe state-contention bottlenecks on high-throughput ledgers and relies on trusted setups. This architectural divergence creates a significant gap in the ecosystem: achieving non-custodial transaction anonymity on highly constrained smart contract platforms without relying on trusted setups or global state accumulators.

Proposed Approach. To bridge this gap, we present *Obscura*, a privacy-preserving protocol that achieves transaction anonymity on Algorand without relying on succinct proofs or trusted setups. The core idea of Obscura is that Linkable Spontaneous Anonymous Group (LSAG) signatures [37], combined with scalable key-value storage and dynamic execution budget allocation, can provide a practical privacy layer on smart contract platforms where pairing-based succinct verification is costly and global accumulators are ill-suited to the state model.

In Obscura, a user deposits a fixed denomination by publishing a cryptographic commitment, a public key derived from a secret scalar on the BN254 elliptic curve. To withdraw the funds to a new address, the user generates an LSAG signature over a ring of on-chain commitments (including their own and several decoys). This signature proves knowledge of the secret corresponding to one of the commitments without revealing which one, thereby providing *signer ambiguity*. To prevent double-spending, the signature includes a *key image* [59] (or nullifier), which is deterministically derived from the user's secret. The smart contract verifies the LSAG signature natively using AVM's elliptic-curve opcodes and ensures the key image has not been previously recorded.

Key Challenges and Technical Solutions. Implementing an LSAG-based privacy-preserving protocol on Algorand presents two primary technical challenges: state management and execution constraints.

First, traditional mixers use a global Merkle tree to track deposits, which requires hashing operations for every insertion and Merkle path verifications during withdrawal. Obscura eschews the Merkle accumulator entirely. Instead, it leverages Algorand's *Box Storage* [4], a scalable key-value store, to record individual commitments and nullifiers. Membership of each ring point is verified in $O(1)$ time by asserting the existence of its corresponding box, significantly simplifying the state model and reducing cryptographic overhead.

Second, verifying an LSAG signature requires $O(n)$ elliptic curve scalar multiplications and additions, where n is the ring size. This linear complexity quickly exceeds the AVM's strict per-call opcode budget. Obscura overcomes this limitation through a dynamic budget expansion strategy. By issuing a burst of inner application calls to a dedicated, stateless "dummy" application before the main verification loop, the protocol aggregates the opcode budgets of multiple grouped transactions into a single execution context. This allows the complete verification of the ring signature within a single block.

Novelty and Positioning. To the best of our knowledge, no prior work has investigated a non-custodial privacy pool for Algorand in which linkable ring signatures are verified entirely on-chain by an AVM smart contract. We emphasize that this claim does not concern the underlying cryptographic primitives: LSAG signatures [37] and key-image-based double-spend prevention [50, 59] are well established, Box Storage is a native Algorand facility [4], and opcode budget pooling is a documented feature of the AVM [7]. Rather, the novelty lies in how these components are combined and adapted to the execution and state model of the AVM. Concretely, Obscura differs from existing work in the following respects:

- *Proof system and setup.* SNARK-based designs such as Zerocash [14] and Tornado Cash [53] prove membership in a Merkle tree of deposits with a succinct proof whose verification relies on pairings and a trusted setup. Obscura instead uses an LSAG signature over an explicit, user-chosen ring, requiring neither a trusted setup nor a succinct proof, at the cost of $O(n)$ proof size and a bounded ring size.
- *State model.* Rather than maintaining a global accumulator whose root must be updated on every deposit, Obscura stores each commitment and each key image in its own box and validates every ring member individually by box existence. No Merkle path is computed or verified, and deposits do not contend for a single shared root.
- *Execution environment.* CryptoNote-style systems [50,59] verify linkable ring signatures at the consensus layer of a dedicated blockchain, and Möbius [42] verifies them within Ethereum smart contracts under gas metering. Obscura targets the AVM, whose binding constraints are a fixed per-call opcode budget, a 2048-byte application-argument limit, and per-transaction

box-reference limits. Accordingly, we derive the exact per-member verification cost (7,608 opcodes), the minimum inner-call multiplier that covers it ($M = 12$), and the maximum ring size admitted by the argument limit ($n = 19$).
– *Application setting.* Obscura targets fixed-denomination, signer-ambiguous transfers on an account-based, high-throughput ledger, with user-controlled selective disclosure through the per-deposit secret x. This differs from the goal of Zether [20], which focuses on confidential account balances.

We do not claim that Obscura provides stronger anonymity than accumulator-based mixers; as discussed in Sections 7 and 8, its anonymity set is bounded by the ring size, whereas that of a Merkle-based pool is bounded by the number of deposits.

Contributions. This paper makes the following contributions:

– **Protocol Design:** We formalize Obscura, an LSAG-based privacy protocol tailored for the constraints of the Algorand Virtual Machine, demonstrating that transaction anonymity is achievable without SNARKs or trusted setups.
– **State and Execution Model:** We introduce an $O(1)$ per-member ring validation approach using Box Storage in place of a global Merkle accumulator, and we adapt the AVM's opcode-pooling mechanism to LSAG verification by deriving the exact per-member cost and the minimum deterministic inner-call multiplier, which together circumvent the AVM's state and execution bottlenecks.
– **End-to-End Implementation:** We provide a fully functional, open-source implementation of the protocol, including the PyTeal smart contract, an off-chain Python prover, and a client-side application. We also detail practical privacy-hardening techniques, such as recent-biased decoy selection and cryptographically secure shuffling, to mitigate operational deanonymization risks.

Figure 1 illustrates the end-to-end protocol architecture.

The remainder of this paper is organized as follows: Section 2 introduces the cryptographic and system preliminaries that establish the foundation for the subsequent construction. Building on these preliminaries, Section 3 formalizes the cryptographic construction, while Section 4 describes the resulting protocol and its operation. Section 5 then presents the smart contract execution model and opcode pooling strategy, followed by Section 6, which details the system implementation. With the construction and implementation established, Section 7 analyzes its security and privacy properties under an explicit threat model. Section 8 subsequently defines the evaluation criteria, evaluates the protocol's performance and practical efficiency. Finally, Section 9 concludes the paper and discusses directions for future research.

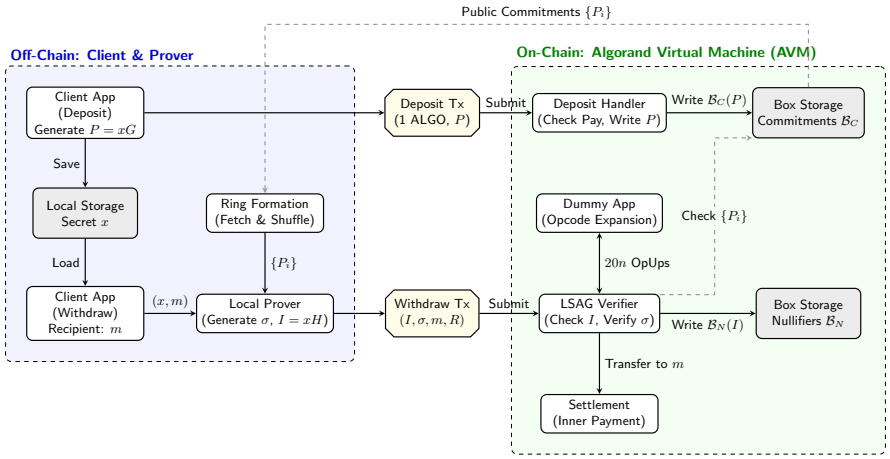


Fig. 1. Obscura protocol end-to-end architecture. Solid arrows denote protocol actions and transaction submissions; dashed arrows denote reads of on-chain public state. The Client App initiates both phases. **Deposit:** the client samples a secret scalar x, computes the public commitment $P = xG$, stores x in local storage, and submits a deposit transaction $(1 \text{ ALGO}, P)$. The on-chain Deposit Handler verifies the payment and writes P into commitment Box Storage $\mathcal{B}_C$. **Withdrawal:** the client loads x and specified recipient m, and fetches public commitments $\{P_i\}$ from $\mathcal{B}_C$ to form a shuffled ring R. THe local prover then produces an LSAG signature σ and key image (nullifier) $I = xH$. The client submits a withdrawal transaction (I, σ, m, R). The LSAG Verifier reads $\{P_i\}$ from $\mathcal{B}_C$, expands its opcode budget via inner calls to a Dummy App ($20n$ OpUps), verifies σ natively with AVM elliptic-curve operations, records I in nullifier Box Storage $\mathcal{B}_N$ if unspent, and settles by an inner payment to m.

2 Preliminaries

This section formalizes the cryptographic primitives and the blockchain execution model that underpin the Obscura protocol.

2.1 Cryptographic Foundations

Elliptic Curve Assumptions. Let $\mathbb{G}$ be a prime-order subgroup of the BN254 elliptic curve [11,19], with order q. We rely on the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP) in $\mathbb{G}$ [35, 46]: given a generator $G \in \mathbb{G}$ and a point $P = xG$, it is computationally infeasible to recover the scalar $x \in \mathbb{Z}_q$. Obscura utilizes two independent, publicly verifiable generators $G, H \in \mathbb{G}$ for which the discrete logarithm relationship is unknown.

Linkable Ring Signatures. A standard ring signature [57] allows a user to sign a message on behalf of an ad hoc set of public keys (the *ring* or *anonymity set*)

without revealing which specific key was used. Unlike group signatures [22], ring signatures are *spontaneous*: no group manager, setup procedure, or coordination among ring members is required. Discrete-logarithm-based instantiations [1] realize the ring structure through proofs of partial knowledge [23]. While this provides signer ambiguity, it lacks a mechanism to prevent a user from signing multiple times using the same secret key.

Linkable Spontaneous Anonymous Group (LSAG) signatures [37] extend this concept by introducing a *key image* (or *linking tag*). For a secret key x and its corresponding public key (commitment) $P = xG$, the key image is deterministically derived as $I = xH$. This key-image mechanism was popularized by the CryptoNote [59] and underpins the MLSAG [50] and CLSAG [32] constructions deployed in Monero. The LSAG construction ensures three critical security properties:

1. **Anonymity (Signer Ambiguity):** Given a signature, a ring of size n, and a key image I, an adversary cannot determine the signer's index with probability significantly greater than $1/n$ [38].
2. **Unforgeability:** It is computationally infeasible to produce a valid signature for a ring without knowing the discrete logarithm of at least one public key in that ring [37].
3. **Linkability:** Any two signatures generated using the same secret key x will necessarily produce the identical key image I. In a decentralized ledger, this property is leveraged to prevent double-spending by ensuring I is recorded and checked against future transactions [50].

The non-interactive nature of the signature is achieved via the Fiat–Shamir heuristic [26], modeling the cryptographic hash function as a random oracle [12].

2.2 Blockchain Execution Model

While public blockchains provide robust consensus and transparency, they expose the entire transaction graph to public scrutiny. Privacy-preserving smart contracts must obscure this graph while operating within the strict resource bounds of the consensus layer. Obscura is designed for the Algorand Virtual Machine (AVM) [2], which imposes specific architectural considerations:

Execution Constraints and Opcode Budget. Smart contracts on the AVM execute within a strict, per-transaction computational limit (opcode budget). While the AVM natively supports elliptic-curve operations on BN254 (e.g., `EcAdd`, `EcScalarMul`), verifying an LSAG signature requires $O(n)$ such operations. To circumvent the per-transaction budget limit, the AVM allows *inner transactions*, application calls made by the contract itself. By issuing a burst of inner calls to a stateless "dummy" application, a contract can dynamically pool the opcode budgets of multiple grouped transactions into a single execution context, enabling the verification of large rings.

State Management via Box Storage. Traditional privacy pools maintain a global Merkle tree of deposits to verify membership succinctly. However, maintaining a global state accumulator introduces storage bottlenecks and requires users to provide Merkle inclusion proofs. Obscura utilizes Algorand's *Box Storage*, an unbounded key-value store scoped to the application. Commitments and key images are stored directly as discrete boxes. This allows the smart contract to verify the membership of any public key in $O(1)$ time by simply asserting the existence of its corresponding box, fundamentally altering the state architecture from a global accumulator to a flat, verifiable key-value mapping.

3 Cryptographic Construction

This section formalizes the Obscura protocol's instantiation of Linkable Spontaneous Anonymous Group (LSAG) signatures over the BN254 elliptic curve. We define the construction in terms of three primary algorithms: KeyGen, Sign, and Verify.

3.1 System Parameters and Hash Instantiation

Let $\mathbb{G}$ be the prime-order subgroup of the BN254 elliptic curve, with prime order $q \approx 2^{254}$. We define two independent, publicly verifiable generators $G, H \in \mathbb{G}$. The discrete logarithm $\log_G H$ is unknown. Points in $\mathbb{G}$ are serialized as 64-byte uncompressed big-endian coordinates to align with the Algorand Virtual Machine (AVM) `EcAdd` and `EcScalarMul` opcodes [2, 3].

We require a cryptographic hash function modeled as a random oracle [12], $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q$. To map the output into a scalar within the AVM's computational constraints, the protocol instantiates $\mathcal{H}$ using SHA-256 [49] followed by a bitwise AND with a fixed 255-bit mask. This clears the most significant bit of the 256-bit digest, constraining the challenge to a 255-bit integer; scalars are subsequently reduced modulo q by the curve arithmetic. Appendix A analyzes the induced challenge distribution and shows that this instantiation preserves the security of the Fiat–Shamir transform.

3.2 Key Generation (KeyGen)

To participate in the protocol (i.e., make a deposit), a user executes the randomized key generation algorithm:

1. Sample a secret scalar $x \xleftarrow{\$} \mathbb{Z}_q$.
2. Compute the public commitment $P = xG \in \mathbb{G}$.
3. Compute the key image (nullifier) $I = xH \in \mathbb{G}$.

The user publishes P to the smart contract during the deposit phase. The secret key x and the key image I are kept private until the withdrawal phase.

3.3 Signature Generation (Sign)

To withdraw funds anonymously, the user acts as a prover to generate an LSAG signature σ. Let $m \in \{0,1\}^{256}$ be the transaction message, defined in Obscura as the 32-byte public key of the withdrawal recipient. This binds the signature to the specific recipient, preventing replay attacks.

The prover selects an anonymity set (ring) of n public commitments $R = \{P_0, \ldots, P_{n-1}\}$ currently stored on-chain, ensuring their own commitment P is included at a secret index $\pi \in [0, n-1]$ such that $P_\pi = xG$. The signature generation proceeds as follows:

1. **Initialization:** Sample a random scalar $\alpha \xleftarrow{\$} \mathbb{Z}_q$. Compute the initial curve points for the true signer index:
$$L_\pi = \alpha G$$
$$R_\pi = \alpha H$$
Compute the subsequent challenge $c_{\pi+1} = \mathcal{H}(m \| L_\pi \| R_\pi)$.
2. **Decoy Simulation:** For each decoy index $i = \pi+1, \ldots, n-1, 0, \ldots, \pi-1$ (evaluating indices modulo n):
   (a) Sample a random response $s_i \xleftarrow{\$} \mathbb{Z}_q$.
   (b) Compute the curve points using the running challenge c_i:
$$L_i = s_i G + c_i P_i$$
$$R_i = s_i H + c_i I$$
   (c) Compute the next challenge $c_{i+1} = \mathcal{H}(m \| L_i \| R_i)$.
3. **Closure:** Close the ring by computing the response for the true signer index π:
$$s_\pi = \alpha - c_\pi x \pmod{q}$$

The resulting signature is the tuple $\sigma = (c_0, s_0, \ldots, s_{n-1})$.

3.4 Signature Verification (Verify)

The smart contract acts as the verifier. It receives the ring R, the key image I, the message m, and the signature σ. Verification requires $O(n)$ elliptic curve operations and proceeds linearly:

1. Initialize the running challenge $c = c_0$.
2. For $i = 0, \ldots, n-1$:
   (a) Compute $L_i = s_i G + c P_i$.
   (b) Compute $R_i = s_i H + c I$.
   (c) Update the challenge $c \leftarrow \mathcal{H}(m \| L_i \| R_i)$.
3. **Accept** if and only if the final challenge c equals the initial c_0.

The algebraic correctness relies on the fact that at the true index π, the verifier computes $L_\pi = s_\pi G + c_\pi P_\pi = (\alpha - c_\pi x)G + c_\pi(xG) = \alpha G$, and similarly $R_\pi = \alpha H$, matching the prover's initialization.

3.5 Proof Serialization

To optimize on-chain parsing, the signature and ring are serialized into a tightly packed byte array (`zk_proof`) structured as:

$$\underbrace{n}_{1\text{ byte}} \;\|\; \underbrace{P_0\|\ldots\|P_{n-1}}_{n\times 64\text{ bytes}} \;\|\; \underbrace{c_0}_{32\text{ bytes}} \;\|\; \underbrace{s_0\|\ldots\|s_{n-1}}_{n\times 32\text{ bytes}} \tag{1}$$

The total payload size is exactly $96n + 33$ bytes. The smart contract extracts c_0 and the responses $\{s_i\}$ directly from this payload, executing the Verify algorithm iteratively while dynamically expanding its opcode budget via inner transactions.

4 Protocol Description

The Obscura protocol operates in two primary phases: deposit (commitment) and withdrawal (anonymous spending). We formalize the transaction semantics, detailing the strict separation between off-chain cryptographic generation and on-chain state transitions.

4.1 Deposit Phase

The deposit phase allows a user to insert funds into the anonymity pool by registering a public commitment. To ensure atomicity between the fund transfer and the state update, the deposit is executed as an *atomic transaction group* on the Algorand blockchain.

Off-Chain Execution. The user samples a secret scalar $x \xleftarrow{\$} \mathbb{Z}_q$ and computes the public commitment $P = xG$. The user securely stores x locally; it is never transmitted to the network.

On-Chain State Transition. The user submits an atomic group consisting of two transactions:

1. A smart contract application call invoking the Deposit method with argument P.
2. A payment transaction transferring exactly 10^6 micro-ALGO (1 ALGO) to the smart contract's escrow address.

Upon receiving the deposit transaction group, the smart contract verifies the payment amount and the receiver. It then checks the global Box Storage for the existence of a commitment box $\mathcal{B}_C$ keyed by a prefix of the commitment, specifically `"c"` $\|\, P[0{:}32]$. If the box already exists, the transaction aborts. Otherwise, the contract allocates the box, writes the full 64-byte P as its value, and increments a global deposit counter. The complete sequence of operations for the deposit phase is illustrated in Figure 2.

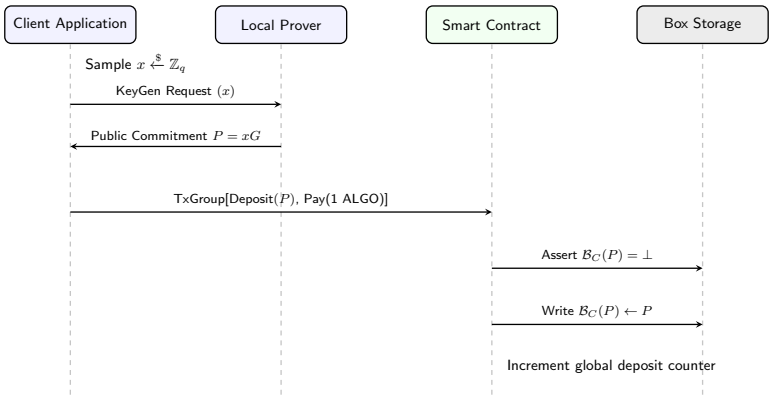


Fig. 2. Deposit phase sequence diagram. The client application generates a secret scalar and delegates the KeyGen elliptic curve operations to the local prover. The client then submits an atomic transaction group to the smart contract, which verifies the payment and allocates a new commitment box in on-chain storage.

4.2 Anonymity Set Construction

Prior to withdrawal, the user must construct an anonymity set (ring) $R = \{P_0, \ldots, P_{n-1}\}$. This process occurs entirely off-chain to prevent the network from learning the true signer's index.

The client queries the blockchain's indexer to retrieve a set of public commitments. To mitigate temporal intersection attacks, the client selects decoys using a recency-biased distribution, favoring recently added commitments; empirical studies of deployed ring-signature systems show that decoy sampling that fails to mimic realistic spending behavior enables statistical deanonymization [36,47]. The client then injects their own commitment P into the set and applies a Fisher–Yates shuffle [27]. This ensures the true commitment is uniformly distributed within the ring, providing optimal $1/n$ signer ambiguity.

4.3 Withdrawal Phase

The withdrawal phase allows a user to spend a previously deposited fund to a new recipient address m without revealing which commitment is being spent.

Off-Chain Execution. The user computes the key image $I = xH$ and executes the Sign algorithm defined in Section 3 over the ring R and message m. The resulting signature σ is packed into a compact byte array.

On-Chain State Transition. The user submits a transaction group (consisting of a main application call and optionally auxiliary calls to supply additional box references if n is large) invoking the Withdraw method, providing the key image

I, the packed signature σ, the recipient address m, and an array of box references corresponding to I and all $P_i \in R$.

Upon receiving the transaction, the smart contract validates the anonymity set against on-chain storage, verifies the LSAG signature natively, and ensures the key image I has not been previously recorded. The precise on-chain execution pipeline and opcode pooling strategy required to perform this validation are detailed in Section 5. Upon successful verification, the contract allocates the nullifier box $\mathcal{B}_N$ to permanently record I as spent, and issues an inner payment transaction transferring the deposit amount (minus protocol and storage fees) to the recipient m.

Figure 3 details the complete sequence of off-chain and on-chain interactions during the withdrawal phase.

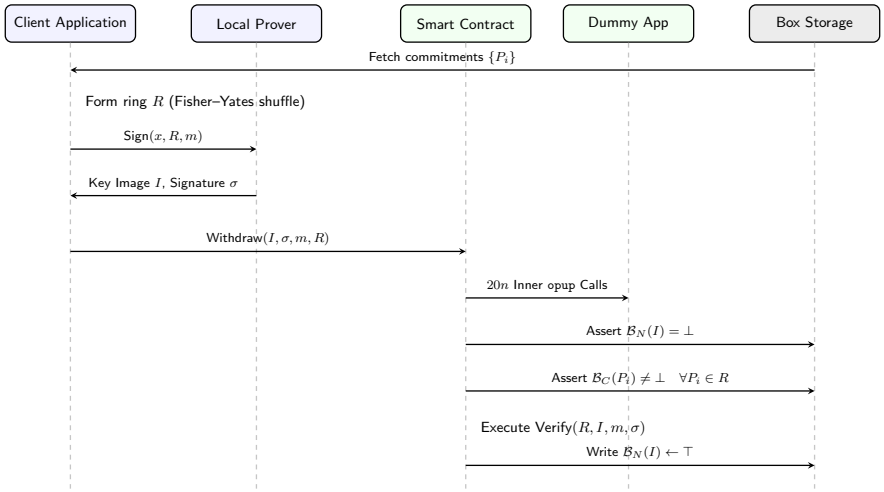


Fig. 3. Withdrawal phase sequence diagram. The client application constructs an anonymity set from on-chain public commitments and delegates signature generation to the local prover. The smart contract dynamically expands its opcode budget via inner calls to a dummy application, validates the ring members and key image against Box Storage, verifies the LSAG signature, and settles the payment.

4.4 Double-Spend Prevention and Linkability

The protocol's security relies on the strict decoupling of the deposit and withdrawal phases. The only cryptographic link between a deposit P and a withdrawal is the shared secret x, knowledge of which is proven in zero-knowledge [30] via the LSAG signature.

Double-spending is deterministically prevented by the algebraic properties of the key image $I = xH$. Because H is a fixed generator, any attempt to spend P a second time will inevitably yield the identical key image I. The smart contract's enforcement of the $\mathcal{B}_N$ box uniqueness guarantees that I acts as a globally unique, unforgeable serial number for the underlying deposit, preserving ledger integrity without compromising signer ambiguity.

5 Smart Contract Execution

Executing complex cryptographic protocols on high-throughput blockchains requires reconciling the computational intensity of the verifier with the strict execution limits of the underlying virtual machine. This section details how Obscura implements the Verify algorithm within the Algorand Virtual Machine (AVM), leveraging specific opcode strategies to ensure privacy-preserving transaction validation.

5.1 Dynamic Opcode Pooling and Deterministic Budgeting

To maintain high network throughput, the AVM imposes a strict computational limit, referred to as the *opcode budget*, on every smart contract invocation. A standard application call is allocated a base budget of 700 units [6]. However, verifying a single member of an LSAG ring over the BN254 curve requires four scalar multiplications (`EcScalarMul`) and two point additions (`EcAdd`). In the AVM, opcode costs are strictly constant and deterministic; they do not vary based on network congestion, input values, or blockchain state. Table 1 provides the exact mathematical breakdown of the AVM opcode costs required to verify a single ring member, totaling 7,608 opcodes.

To overcome this limitation without requiring interactive verification or off-chain SNARK proofs, Obscura employs a dynamic budget expansion strategy known as *opcode pooling*. The AVM allows the execution budget of multiple transactions within an atomic group (including inner transactions) to be pooled into a shared execution context [7].

Before executing the cryptographic verification loop, the Obscura smart contract issues $12n$ inner application calls to a stateless, foreign "dummy" application. This dummy application exposes a single `opup` method that performs no logic and simply approves the call. Because each inner transaction contributes an additional 700 units to the shared pool, $12n$ calls dynamically provision sufficient computational headroom to verify a ring of size n entirely on-chain. This trades a marginally higher transaction fee for the ability to execute non-interactive, single-epoch verification.

To increase the available budget, the contract executes a loop that constructs and submits inner application calls. As detailed in Table 2, building and submitting an inner transaction introduces an overhead of 23 opcodes in the main transaction. Since each inner transaction adds exactly 700 opcodes of pooled budget, the net budget gain per inner application call is 677 opcodes.

Table 1. Deterministic AVM Opcode Costs for LSAG Verification (Per Ring Member)

Operation	**Opcode Cost**
4× `EcScalarMul` (BN254g1) (4×1810)	7,240
2× `EcAdd` (BN254g1) (2×125)	250
1× `Sha256` (1×35)	35
1× `BytesAnd` (1×6)	6
1× `App.box_length` (1×1)	1
Loop Overhead and Control Flow	∼76
Total Exact Cost per Member (C)	**7,608**

Table 2. Inner Transaction Budget Gain and Multiplier Calculation

Budget Parameter	**Value (Opcodes)**
Budget Added per Inner Call	+700
Main Loop Inner Txn Submit Overhead	-23
Net Budget Gain per Call (G)	**677**

Let n be the ring size and M be the inner transaction multiplier (the number of inner calls per ring member). The total opcode cost is $7608n + 23Mn + 115$ (where 115 accounts for non-loop initialization overhead), and the total opcode budget provided is $700 + 700Mn$. For successful execution, the budget must strictly exceed the cost:

$$\begin{aligned} 700 + 700Mn &> 7608n + 23Mn + 115 \\ 677Mn &> 7608n - 585 \\ 677M &> 7608 - \frac{585}{n} \end{aligned}$$

For the maximum practical ring size $n = 19$, this simplifies to $677M > 7577.2$, requiring $M > 11.19$. Because the multiplier must be an integer, the absolute theoretical minimum multiplier is $M = 12$. At this setting, each member is allocated a budget of 8,400 opcodes (net gain of 8,124 opcodes), providing a precise 516-opcode (7.2%) safety margin over the required 7,608 opcodes. By utilizing this exact deterministic minimum, the protocol guarantees successful execution while achieving the absolute lowest possible user transaction fees.

5.2 On-Chain Verification Pipeline

The on-chain verification pipeline strictly orders state validation and cryptographic execution to ensure soundness and prevent resource exhaustion attacks. The execution path for the withdrawal proceeds as follows:

1. **Budget Provisioning:** The contract executes the $12n$ inner `opup` calls to expand the available opcode budget.

2. **Double-Spend Assertion:** The contract evaluates the key image I provided in the transaction arguments. It executes a state read against the Box Storage to assert that the nullifier box $\mathcal{B}_N$ ($\texttt{"n"} \parallel I[0{:}32]$) has a length of zero (i.e., it does not exist). If the box exists, the execution halts immediately, preventing double-spending.
3. **Anonymity Set Validation:** The contract parses the packed proof payload to extract the ring size n and the array of public commitments $\{P_0, \ldots, P_{n-1}\}$. For each P_i, the contract asserts the existence of its corresponding commitment box $\mathcal{B}_C$ ($\texttt{"c"} \parallel P_i[0{:}32]$). This enforces that the anonymity set is composed exclusively of valid deposits, preventing an adversary from injecting fabricated public commitments to trivially forge a signature.
4. **Cryptographic Loop:** The contract initializes the running challenge $c = c_0$. For each index $i \in [0, n-1]$, it executes the core LSAG verification logic using native AVM opcodes:

$$\begin{aligned} L_i &= \texttt{EcAdd}(\texttt{EcScalarMul}(G, s_i), \texttt{EcScalarMul}(P_i, c)) \\ R_i &= \texttt{EcAdd}(\texttt{EcScalarMul}(H, s_i), \texttt{EcScalarMul}(I, c)) \\ h_i &= \text{SHA-256}(m \| L_i \| R_i) \\ c' &= \texttt{BytesAnd}(h_i, \text{mask}) \end{aligned}$$

   The running challenge is updated ($c \leftarrow c'$). The `BytesAnd` opcode applies a fixed 255-bit mask to the hash digest, ensuring the resulting scalar is strictly within the field $\mathbb{Z}_q$.
5. **Closure and Settlement:** After the loop terminates, the contract asserts that the final challenge c strictly equals the initial c_0 provided in σ. Upon successful closure, the contract allocates $\mathcal{B}_N$ to record the key image I and issues the inner payment transaction to the recipient m.

5.3 Execution-Layer Atomicity

The AVM's execution model guarantees that state transitions are strictly atomic. If any assertion fails, whether due to an invalid signature closure, a non-existent commitment box, or an already-spent key image, the entire transaction group reverts. This ensures that the nullifier is never recorded unless the signature is cryptographically valid, and funds are never disbursed unless the nullifier is successfully recorded. Furthermore, because the recipient address m is explicitly bound into the hash chain h_i, any attempt by an adversary or a malicious node to intercept the transaction and alter the payout destination will deterministically break the signature closure, causing the execution to revert.

6 System Implementation

The Obscura protocol is realized as a full-stack decentralized application, bridging off-chain cryptographic generation with on-chain verification. The implementation is partitioned into three distinct layers: the on-chain smart contract, the

off-chain cryptographic prover, and the client-side application. This architecture strictly isolates sensitive key material from public network transmission while optimizing for the execution constraints of the Algorand Virtual Machine (AVM). Figure 4 illustrates the system's architecture and trust boundaries.

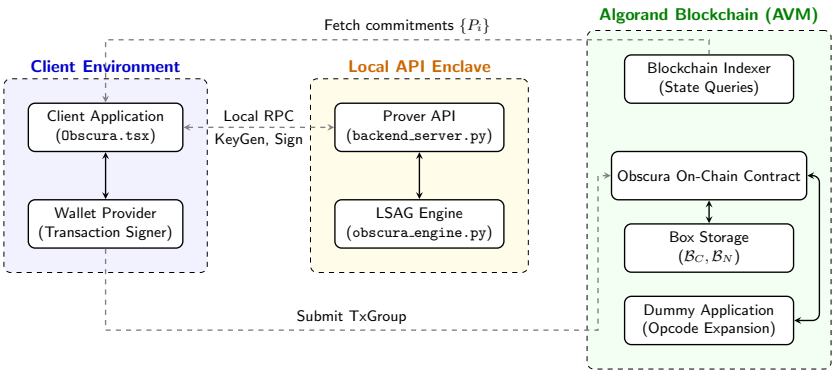


Fig. 4. Obscura implementation architecture. The system is partitioned into three distinct trust domains. The client environment handles decoy selection and transaction construction. The local API enclave executes heavy elliptic curve operations without exposing the secret scalar x to the network. The Algorand blockchain enforces state transitions, utilizing the indexer for state queries, Box Storage for membership checks, and a dummy application for opcode pooling.

6.1 Smart Contract Layer

The verification logic and state management are implemented in PyTeal [5] and compiled to AVM bytecode. The contract acts as the ultimate arbiter of the protocol, enforcing the atomic state transitions defined in Section 4.

To circumvent the storage limitations of global and local state schemas, the contract utilizes Algorand Box Storage. Commitment boxes and nullifier boxes are instantiated using 33-byte keys, prefixed with `"c"` and `"n"` respectively, concatenated with the first 32 bytes of the curve point, mapping to their full 64-byte values. This flat, key-value architecture enables $O(1)$ membership and double-spend checks, entirely avoiding the computational overhead of maintaining and verifying Merkle tree accumulators on-chain.

6.2 Off-Chain Prover Enclave

Heavy elliptic curve operations required for signature generation are delegated to a dedicated Python-based prover engine. This engine implements the KeyGen and Sign algorithms and exposes them via a stateless HTTP prover API.

When a user initiates a withdrawal, the client application transmits the secret scalar x, the selected anonymity set R, and the recipient address m to the prover. The prover computes the key image I and the LSAG signature σ, returning them to the client. In a production environment, this prover must be operated as a trusted local enclave (e.g., compiled to WebAssembly within the browser) to ensure the secret scalar x is never exposed to an untrusted network.

6.3 Client Application and Decoy Selection

The frontend is a React/TypeScript application that manages user interaction, wallet integration (via the Pera Wallet connector [52]), and transaction construction.

A critical security responsibility of the client application is the construction of the anonymity set. To mitigate temporal intersection attacks, the client queries the blockchain indexer for recent deposits, maintaining a recency-biased pool of commitments. It selects $n-1$ decoys and injects the user's true commitment P. To prevent positional bias that could deanonymize the user, the client applies a Fisher–Yates shuffle to the ring. Crucially, this shuffle is seeded using the cryptographically secure `crypto.getRandomValues()` API, thereby avoiding the use of weak pseudorandom number generators [41].

6.4 Engineering Constraints and Parameterization

To minimize transaction size and parsing overhead on-chain, the signature σ and the ring R are serialized into a tightly packed byte array before transmission. As defined in Equation 1, the payload strictly follows the format: a 1-byte ring size n, followed by n 64-byte commitments, a 32-byte initial challenge c_0, and n 32-byte scalar responses. This results in a deterministic payload size of $96n + 33$ bytes.

The ring size n is a configurable protocol parameter, taking values from $n = 1$ to $n = 19$. This upper limit is dictated by the AVM consensus parameter `MaxAppTotalArgLen`, which restricts the total size of application arguments to 2048 bytes. Because the proof payload requires exactly $96n + 33$ bytes, $n = 19$ provides the maximum anonymity set that fits within this limit (1857 bytes for the proof, plus additional routing and state arguments). While smaller than the anonymity sets found in SNARK-based protocols, this parameterized design allows users to select an optimal balance between signer ambiguity and the linear execution costs of native elliptic-curve operations on Algorand.

7 Security and Privacy Analysis

We evaluate the security and privacy guarantees of the Obscura protocol under a formal adversarial model, separating cryptographic properties from operational and system-level limitations. Formal definitions, theorems, and complete proofs are deferred to Appendix A.

7.1 Threat Model and Assumptions

We assume a public, immutable ledger where all state transitions, smart contract payloads, and Box Storage contents are perfectly visible to a Globally Passive Adversary (GPA), following standard anonymity terminology [54]. The adversary may also act actively by injecting malicious transactions, attempting to double-spend, or providing malformed cryptographic payloads. We assume the underlying Algorand consensus mechanism [29] and the AVM execution environment are correct and secure.

Crucially, we assume the off-chain Sign algorithm is executed within a trusted local enclave controlled by the user. Delegating proof generation to an untrusted remote server trivially violates privacy, as the server must observe the secret scalar x to compute the signature.

7.2 Cryptographic Guarantees

Unforgeability and Soundness. The protocol's soundness relies on the existential unforgeability of the LSAG construction under chosen-message attacks (EUF-CMA) [31]. Assuming the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP) on the BN254 curve, and modeling the hash function $\mathcal{H}$ as a random oracle, it is computationally infeasible for an adversary to produce a valid signature σ without knowledge of the secret scalar x corresponding to at least one public commitment P_i in the ring R [37].

Non-Malleability and Replay Resistance. To prevent transaction malleability and replay attacks, the recipient's public key m is explicitly bound into the Fiat–Shamir challenge chain: $h_i = \text{SHA-256}(m \| L_i \| R_i)$. If an active network adversary intercepts a valid withdrawal transaction and attempts to redirect the funds by altering m, the hash outputs will diverge entirely. This breaks the algebraic closure of the ring (i.e., the final challenge will not equal c_0), causing the smart contract to deterministically reject the transaction.

Linkability and Double-Spend Resistance. Double-spending is prevented via the cryptographic property of linkability. The protocol guarantees that any two signatures generated from the same secret scalar x will yield the identical key image $I = xH$. Because the AVM smart contract atomically asserts the non-existence of the nullifier box $\mathcal{B}_N$ prior to executing the payout, the system achieves deterministic double-spend resistance. An adversary cannot reuse a spent commitment, even if they select a completely different set of decoys for the second ring.

7.3 Privacy Analysis and Practical Limitations

Signer Ambiguity. The primary privacy guarantee of the protocol is signer ambiguity. Given a valid signature σ over a ring R of size n, a computationally bounded adversary cannot identify the true signer's index π with a probability significantly greater than $1/n$. The true commitment P and the key image I are

generated using independent base points (G and H), rendering them computationally unlinkable without knowledge of x under the Decisional Diffie–Hellman assumption in $\mathbb{G}$ (Theorem 3, Appendix A).

Auditability and Selective Disclosure. While Obscura provides default anonymity to the public, it inherently supports user-controlled selective transparency. Because the off-chain client retains the secret scalar x, a user can cryptographically prove the provenance and destination of their funds to an authorized third party (e.g., an auditor or regulatory body). By disclosing x, the auditor can deterministically verify both the deposit commitment $P = xG$ and the withdrawal key image $I = xH$. This mechanism ensures compatibility with compliance and reporting obligations without compromising the privacy of the broader anonymity set, in the spirit of accountable-privacy designs for decentralized anonymous payments [28].

Decoy Selection and Intersection Attacks. Theoretical anonymity assumes a uniform distribution of fetched commitments. In practice, adversaries can employ temporal intersection attacks or statistical correlation if users predictably select decoys or withdraw immediately after depositing; analogous heuristics have been used to trace a substantial fraction of transactions in early Monero [36,47] and to deanonymize careless users of Ethereum mixers [15]. The Obscura client mitigates this by enforcing a recency-biased decoy pool and applying a cryptographically secure shuffle. This ensures the true commitment's position in the on-chain array reveals no information. However, users must still exercise user-side operational discipline (e.g., introducing time delays between deposits and withdrawals) to maximize the effective anonymity set.

Anonymity Set Size Constraints. Due to the 2048-byte application argument limit of the AVM, the current testnet implementation caps the ring size at $n = 19$. While this provides functional privacy against casual blockchain surveillance and is a significant improvement over smaller rings, the combinatorial hiding remains weaker than SNARK-based accumulators (e.g., Tornado Cash), where the effective anonymity set is bounded only by the number of active deposits in a pool. Obscura trades the theoretical unbounded size of the anonymity set for the ability to execute natively and transparently on a high-throughput, non-pairing-friendly virtual machine.

Post-Quantum Vulnerability. The protocol's security relies fundamentally on the hardness of ECDLP. A quantum adversary executing Shor's algorithm [60] could efficiently recover the secret scalar x from the public commitment $P = xG$. With x exposed, the adversary could compute $I = xH$ to deanonymize that specific transaction (if already spent) or forge a withdrawal for the corresponding commitment (if unspent). While the per-deposit sampling of x isolates the compromise to individual deposits rather than a global identity, this remains a fundamental limitation of all classical elliptic-curve privacy protocols; transitioning to post-quantum security would require replacing the LSAG construction with lattice-based or hash-based alternatives [24,39].

8 Performance Evaluation

We evaluate the practical resource footprint of the Obscura protocol, focusing on computational overhead, storage requirements, and the inherent scalability limitations of executing ring signatures natively on the AVM. We first describe the experimental setup and define the evaluation criteria, then position the evaluation with respect to prior work, and finally report and interpret the measurements.

8.1 Experimental Setup

All measurements were obtained with the reference implementation of Section 6 deployed on the Algorand Testnet: the PyTeal contract and the dummy `opup` application on-chain, the Python prover exposed through its local HTTP API, and the React client with Pera Wallet. We sweep the ring size over its entire admissible range, $n \in \{1, \ldots, 19\}$, and execute complete deposit and withdrawal flows at each ring size. The client instruments each flow into the operational stages defined in Section 8.5. Each flow is executed five times per ring size, and latencies are reported as the mean over these five runs, with error bars denoting one standard deviation. Proof size, inner-transaction count, and transaction fees are deterministic functions of n and are therefore reported without error bars.

8.2 Evaluation Criteria

Because LSAG proof size and verification cost both scale linearly with n, the main question for Obscura is not whether a single configuration is fast, but how each cost grows with the anonymity set and where it meets the hard limits of the Algorand blockchain. We therefore report every criterion as a function of n. Table 3 summarizes the criteria, which we define as follows.

- **Anonymity set size (n).** The number of commitments in the ring, which bounds the probability of identifying the true signer by $1/n$ plus a negligible term (Theorem 3). It serves as the independent variable of the evaluation, against which all costs are traded off. We do not empirically measure resistance to statistical deanonymization; as discussed in Section 7, operational behavior can only degrade this cryptographic bound.
- **Proof size (bytes).** The length of the packed `zk_proof` payload of Equation 1. It measures the communication cost of a withdrawal and determines feasibility, since the payload must fit within the 2048-byte `MaxAppTotalArgLen` limit together with the remaining arguments. It is therefore the criterion that fixes the maximum ring size.
- **On-chain execution cost (opcodes and inner transactions).** The opcode cost of verification and the number of inner transactions the contract must issue to provision it. Because AVM opcode costs are deterministic, this criterion tests whether the analytical cost model of Section 5 is sufficient for verification to complete in a single call at every admissible n. The inner-transaction count is also the quantity that drives the fee criterion below.

- **Transaction fees (ALGO).** The network fees incurred by the deposit and withdrawal transaction groups. This is the direct monetary cost borne by users, and it captures the economic price of opcode pooling, since each inner transaction incurs the minimum network fee.
- **Proof generation time (ms).** The time for the local prover to compute the key image and execute Sign. It measures the client-side computational burden, which is relevant because the security model requires signing to occur in a trusted local environment rather than on a remote server (Section 7).
- **End-to-end latency and stage breakdown (ms).** The wall-clock time from the user initiating a flow to on-chain confirmation, decomposed into stages. It reflects the user-perceived responsiveness of the protocol, and the breakdown separates costs that Obscura controls (data preparation, proof generation, transaction construction) from network-dependent costs (submission and block confirmation).
- **On-chain state footprint and payout overhead.** The storage allocated per deposit and per withdrawal, the associated minimum balance requirement, and the resulting reduction of the withdrawal payout. These measure the long-term state cost of replacing an accumulator with per-item boxes and the effective value received by the user.

Table 3. Evaluation Criteria and Their Role in the Evaluation

Criterion	What It Assesses	Reported In
Ring size n	Signer ambiguity bound ($1/n$)	Independent variable
Proof size	Communication cost; feasibility under 2048-byte limit	Fig. 9
Execution cost	Sufficiency of opcode pooling; inner calls required	Tables 1, 2; Fig. 5
Transaction fees	Monetary cost to the user	Fig. 6
Proof generation	Client-side computational burden	Fig. 8
Latency	User-perceived responsiveness, by stage	Figs. 7–11
State and payout	Storage growth; value received by user	Section 8.4

8.3 Positioning of the Evaluation

To the best of our knowledge, no prior work has reported an on-chain implementation and measurement of linkable ring signature verification on the AVM. Consequently, there is no like-for-like Algorand baseline against which Obscura can be compared. We also deliberately refrain from numerical comparisons with Ethereum-based designs such as Tornado Cash [53] or Möbius [42], as such comparisons would conflate differences in fee markets, execution metering (gas versus a per-call opcode budget), and block times, and we have not measured those systems under a common setup. Instead, we evaluate Obscura against two references

that are well defined for our setting: the analytical cost model derived in Section 5, and the hard limits imposed by the AVM. This choice also reflects what distinguishes the design. The quantities measured here, namely the per-member opcode cost covered by inner `opup` calls, the box-based validation of every ring member, and the argument-size bound on n, arise from the design choices described in Section 5. They differ structurally from the costs of accumulator-based designs, in which membership is established by verifying a succinct proof against a Merkle root.

8.4 Computational and Storage Overhead

Proof Size and Communication Complexity. Unlike succinct zero-knowledge proofs (e.g., zk-SNARKs [33]) which offer $O(1)$ proof sizes, LSAG signatures scale linearly with the anonymity set size n. As defined in Equation 1, the serialized `zk_proof` payload requires exactly $96n + 33$ bytes. For the maximum ring size of $n = 19$, the payload is 1857 bytes, fitting just under the 2048-byte protocol limit. While this is efficient compared to non-elliptic-curve ring signatures, the $O(n)$ scaling imposes a strict upper bound on feasible ring size due to transaction size constraints and increasing network propagation overhead.

On-Chain State Footprint. Obscura eschews global state accumulators in favor of Algorand Box Storage. Each deposit allocates a 64-byte commitment box, and each withdrawal allocates a 64-byte nullifier box. This results in an $O(N)$ storage footprint, where N is the total number of transactions in the system's history. While this prevents the computational bottleneck of Merkle tree updates, it shifts the burden to state bloat. To mitigate this, the protocol enforces a strict minimum balance requirement (MBR) for box allocation, which is deducted from the withdrawal payout.

Execution Cost and Opcode Scaling. The computational bottleneck of the protocol is the on-chain Verify algorithm. As mathematically proven in Section 5, each ring member strictly requires 7,608 opcode units for elliptic curve scalar multiplications and additions. To satisfy this, the client must provision $12n$ inner `opup` calls. This linear scaling of execution cost directly translates to higher transaction fees for the user, as each inner call incurs the network's minimum base fee. Figure 5 illustrates the scaling of the required inner transactions, and Figure 6 demonstrates the resulting linear impact on the total withdrawal transaction fee, compared to the constant fee of the deposit phase.

8.5 Scalability Trade-offs

Scalability vs. Anonymity. The fundamental limitation of Obscura is the trade-off between execution feasibility and the size of the anonymity set. Because both proof size and verification cost scale as $O(n)$, the system cannot practically support the massive anonymity sets achieved by Merkle-based SNARK mixers. The current testnet implementation supports ring sizes up to $n = 19$. While this

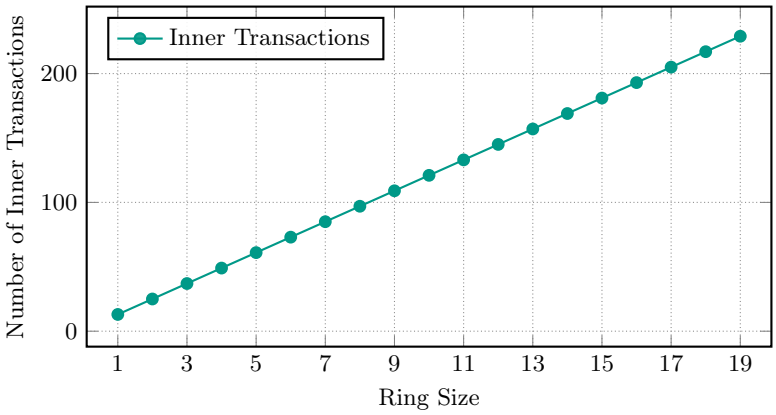


Fig. 5. Number of inner transactions issued during the withdrawal phase versus ring size. To overcome the strict per-transaction AVM opcode budget, the smart contract dynamically groups inner `opup` calls. Each ring member strictly requires 12 inner transactions to safely cover the 7,608 opcode verification cost, leading to an $O(n)$ scaling in the total number of inner transactions.

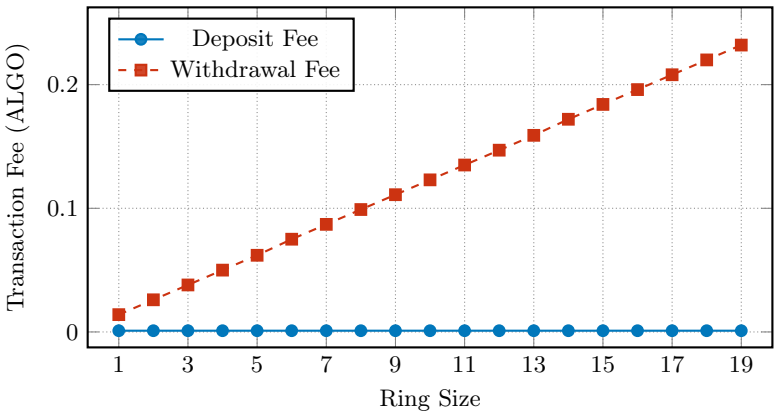


Fig. 6. Deposit and withdrawal transaction fees as a function of ring size. The deposit fee remains constant at 0.001 ALGO because it requires a simple state box allocation. Conversely, the withdrawal fee scales linearly due to the dynamic opcode pooling mechanism, which issues $12n$ inner `opup` calls per ring member to provision sufficient AVM execution budget.

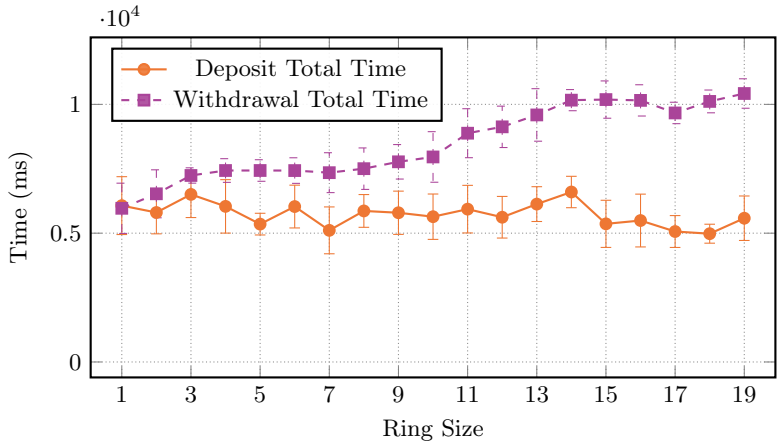


Fig. 7. End-to-end deposit and withdrawal latency across varying ring sizes. The deposit process exhibits a relatively stable latency profile regardless of the anonymity set size, as its operations are constant. In contrast, withdrawal latency increases linearly with the ring size, driven primarily by the $O(n)$ computational overhead of generating the zero-knowledge LSAG signature off-chain and managing larger transaction groups.

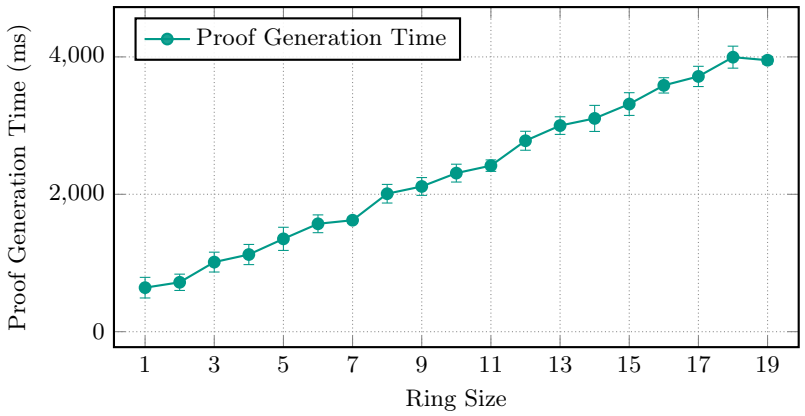


Fig. 8. Zero-knowledge ring signature proof generation time versus ring size. The off-chain local prover executes the Sign algorithm using BN254 elliptic curve operations. Since the prover must compute decoy responses for every member in the anonymity set to construct the LSAG signature, the generation time scales linearly $O(n)$.

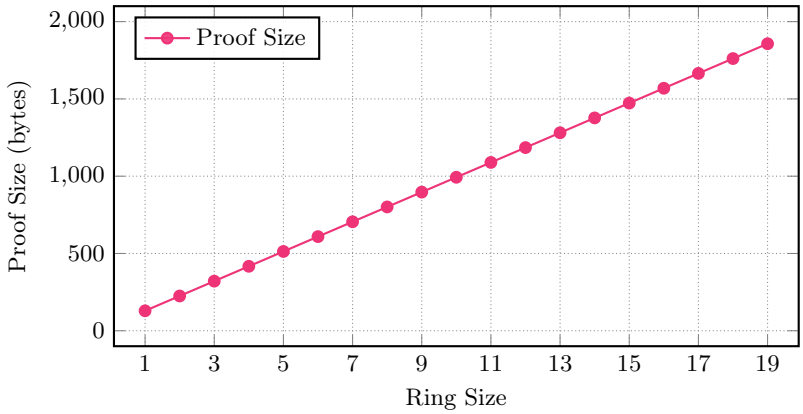


Fig. 9. Proof size growth as a function of ring size. The packed `zk_proof` payload includes the ring size, n public commitments, an initial challenge, and n scalar responses, resulting in a deterministic size of $96n + 33$ bytes. The strict 2048-byte `MaxAppTotalArgLen` limit of the AVM dictates the practical upper bound of $n = 19$ for a single verification call.

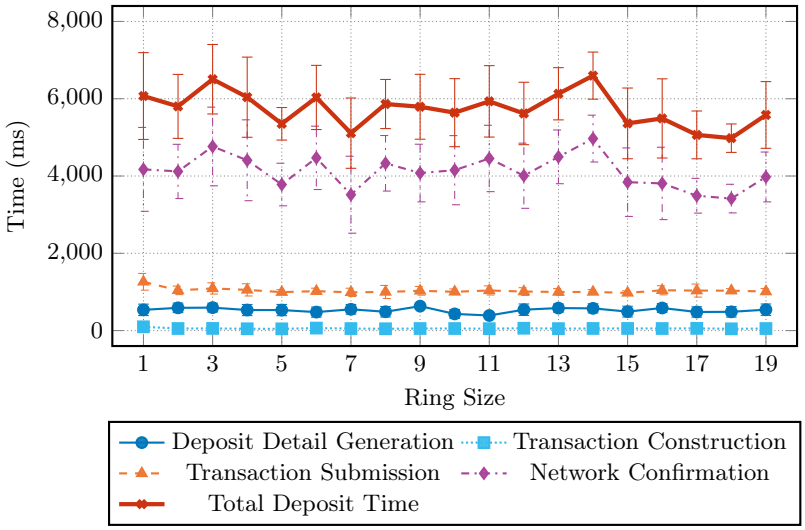


Fig. 10. Breakdown of deposit latency versus ring size. The total deposit time is decomposed into four discrete operational stages: deposit detail generation, transaction construction, transaction submission, and network confirmation. Because the deposit phase involves a constant-time commitment generation and a standard atomic transaction group, all latency components remain independent of the ring size parameter n.

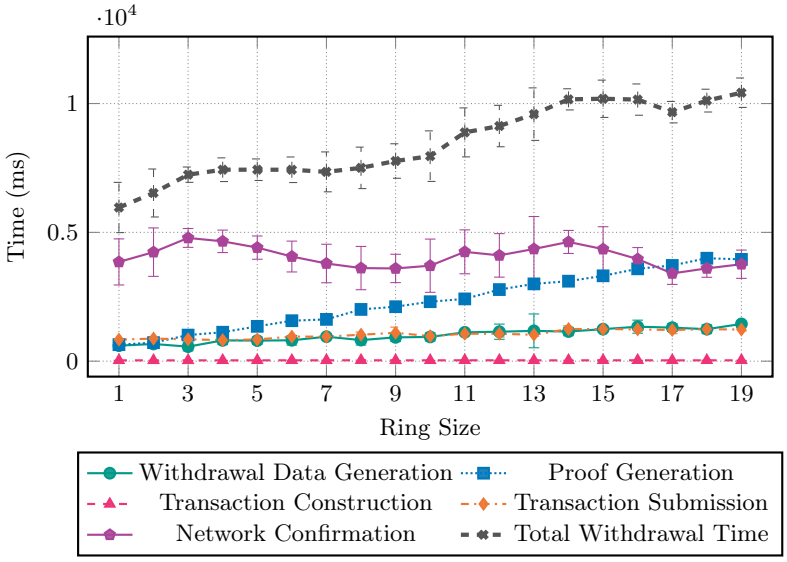


Fig. 11. Breakdown of withdrawal latency versus ring size. The end-to-end withdrawal time is dominated by proof generation, which scales linearly $O(n)$ due to the cryptographic complexity of the LSAG signature. Transaction construction and submission times also exhibit slight growth for larger rings ($n > 6$) as the client must construct and sign additional grouped auxiliary transactions to bypass the AVM's Box Reference limits.

provides significantly stronger combinatorial hiding than smaller rings, it remains vulnerable to sophisticated statistical deanonymization if the overall pool of users is small or highly correlated. Figure 9 and Figure 8 illustrate the linear scaling of proof payload size and generation time, respectively. Figure 7 shows the end-to-end deposit and withdrawal latency. Figure 10 and Figure 11 provide a detailed breakdown of the latency for each phase of the protocol. To fully contextualize these measurements, the end-to-end execution is instrumented into discrete operational stages.

End-to-End Latency Breakdown. For the **deposit flow** (Figure 10), the latency is decomposed into:

- *Deposit detail generation:* The duration required to generate the cryptographically secure random secret locally and invoke the backend service to compute the corresponding BN254 commitment.
- *Transaction construction:* The time taken to fetch suggested transaction parameters from the Algorand node, construct the application call (carrying the commitment box reference) alongside the payment transaction, and group them atomically.
- *Transaction submission:* The latency of signing the transaction group via the wallet provider and submitting the raw payload to the network.

– *Network confirmation:* The network-level latency incurred waiting for the block to be proposed and the transaction to be confirmed on-chain.

For the **withdrawal flow** (Figure 11), the latency incorporates the computationally intensive cryptographic generation:

– *Withdrawal data generation:* The time taken to retrieve contract state (reading commitment boxes), query the Indexer for recent deposits, select decoys, and construct the cryptographically shuffled anonymity set (the ring).
– *Proof generation:* The exact duration the local backend enclave requires to compute the key image and generate the zero-knowledge LSAG ring signature proof using native BN254 curve operations. This phase exhibits the most significant $O(n)$ computational scaling.
– *Transaction construction:* The time to decode the recipient address, fetch parameters, and build the withdrawal payload. For larger ring sizes, this includes the overhead of splitting box references across multiple grouped auxiliary transactions to bypass the AVM's consensus limits.
– *Transaction submission:* The time to sign the withdrawal transaction (or transaction group) and submit it to the network.
– *Network confirmation:* The latency from transaction broadcast to final on-chain confirmation.

Fixed Denominations and Amount Constraints. In the current testnet implementation, the transaction value is strictly fixed to 1 ALGO to ensure that all commitments within the anonymity set remain indistinguishable. While the system architecture could be extended in the future to support multiple parallel pools with different fixed denominations (e.g., separate 10 ALGO or 100 ALGO pools), arbitrary value selection within a single pool is not supported. Enabling flexible, hidden transaction amounts would require transitioning to a full Confidential Transaction (CT) scheme [50]. Such an extension would significantly increase the cryptographic overhead and verification complexity on the AVM. Consequently, users wishing to mix large volumes of funds must execute multiple discrete deposits and withdrawals, increasing their operational overhead and fee burden.

Payout Reductions. The withdrawal payout is not a 1:1 return of the deposited 1 ALGO. The smart contract deducts the network fees required for the inner `opup` burst, the inner payment transaction, and the Box Storage MBR. In the current reference implementation, the recipient receives $1{,}000{,}000 - 1{,}000 - 100{,}000 = 899{,}000$ micro-ALGO. Users must account for this $\sim$10% protocol overhead when utilizing the protocol.

8.6 Summary of Results Against the Criteria

Table 3 defines the criteria; the measurements against them are summarized below.

- **Proof size.** The measured payload grows from 129 bytes at $n = 1$ to 1857 bytes at $n = 19$, matching $96n + 33$ exactly (Figure 9). The argument-size limit, rather than the opcode budget, is therefore the binding constraint on n.
- **Execution cost.** The withdrawal issues $12n + 1$ inner transactions, namely $12n$ `opup` calls and the inner payout, ranging from 13 to 229 (Figure 5). Withdrawals completed and were confirmed at every ring size from 1 to 19 (Figure 11), which is consistent with the analytical result that $M = 12$ suffices across the admissible range.
- **Transaction fees.** The deposit fee is constant at 0.001 ALGO, whereas the withdrawal fee grows linearly from 0.014 ALGO at $n = 1$ to 0.232 ALGO at $n = 19$ (Figure 6), reflecting the inner-transaction count.
- **Proof generation.** Local signing time grows approximately linearly from about 0.64 s at $n = 1$ to about 4.0 s at $n = 19$ (Figure 8).
- **Latency.** End-to-end deposit latency remains between approximately 5.0 and 6.6 s with no dependence on n. End-to-end withdrawal latency increases from about 6.0 s at $n = 1$ to about 10.4 s at $n = 19$ (Figure 7); roughly 3.3 s of this 4.5 s increase is attributable to proof generation. Network confirmation contributes roughly 3.4–5.0 s in both phases independently of n (Figures 10 and 11).
- **State and payout.** Each deposit and each withdrawal allocates one 64-byte box, so storage grows linearly with protocol usage rather than through accumulator updates, and the recipient of a 1 ALGO note receives 899,000 micro-ALGO.

9 Conclusion and Future Work

We introduced Obscura, a decentralized privacy protocol that establishes transaction anonymity on highly constrained smart contract platforms. By instantiating Linkable Spontaneous Anonymous Group (LSAG) signatures over the BN254 curve, we demonstrated that signer ambiguity and double-spend resistance can be enforced entirely on-chain without relying on trusted setups or native pairing operations. Our approach circumvents traditional virtual machine bottlenecks by replacing global Merkle accumulators with an $O(1)$ Box Storage state model, and it overcomes strict execution limits via dynamic opcode pooling. Obscura bridges the gap between high-throughput ledger architectures and the cryptographic requirements of non-custodial privacy pools, demonstrating that default transaction anonymity and user-controlled regulatory compliance can coexist.

While the protocol provides a functional and implementation-ready privacy layer, its reliance on $O(n)$ elliptic-curve verification inherently bounds the practical anonymity set size. We identify several critical directions for future research:

- **Efficient Succinct Verification Without Pairings:** To scale anonymity sets, future work must explore zero-knowledge proof systems that offer sublinear verification. While the AVM provides support for native elliptic-curve

pairings, their high computational cost in terms of opcode consumption, combined with state access contention arising from Merkle accumulator updates, makes the efficient deployment of traditional pairing-based zk-SNARK mixers challenging to scale efficiently. Adapting inner-product arguments (e.g., Bulletproofs [21]) or hash-based STARKs [13] for the specific opcode constraints of the AVM would significantly enhance combinatorial hiding without relying on trusted setups.

- **Variable Denominations:** The current protocol enforces a fixed denomination to maintain commitment indistinguishability. Extending the system to support arbitrary transaction amounts requires integrating Confidential Transactions [40, 50]. Research is needed to efficiently implement Pedersen commitments [51] and zero-knowledge range proofs [21] within strict smart contract execution budgets.
- **Post-Quantum Resilience:** Because Obscura's soundness and privacy rely fundamentally on the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP), it is vulnerable to future quantum adversaries. Transitioning the on-chain verification logic to support post-quantum ring signatures, such as lattice-based constructions [24, 39], remains a significant open challenge for smart contract privacy.
- **Formal Verification:** Finally, while the cryptographic primitives are well-studied, their instantiation in PyTeal and subsequent compilation to AVM bytecode may introduce potential implementation vulnerabilities. Formally verifying the correctness of the compiled bytecode and the underlying cryptographic algebra is essential to provide high-assurance security guarantees for the on-chain verifier.

Code Availability

To facilitate reproducibility and public auditing, the complete source code supporting this work is available at https://github.com/n-azimi/Obscura.

The repository provides the full end-to-end implementation of the Obscura protocol, including the PyTeal smart contract and associated verification script for on-chain execution, the Python-based LSAG cryptographic engine and local prover API, and the React client application responsible for decoy selection and transaction construction. Furthermore, the repository includes contract deployment scripts and analytical tooling (Obscura Inspector and Obscura Lens) for evaluating transaction topology and ledger state.

A Formal Security Model and Proofs

This appendix provides a formal cryptographic treatment of the Obscura protocol, adapting the security model of Linkable Spontaneous Anonymous Group (LSAG) signatures [37] to our construction over the BN254 elliptic curve. We fix notation and computational assumptions, formalize the scheme syntax and its correctness, analyze the concrete challenge-hash instantiation, and reduce each

claimed security property to a well-studied hard problem in the random oracle model (ROM) [12].

A.1 Notation

Table 4 summarizes the notation used throughout this appendix. We note one convention inherited from the main text: the unsubscripted symbol R always denotes the ring of public commitments, whereas the subscripted symbols $L_i, R_i \in \mathbb{G}$ denote the verification points of the i-th ring member.

Table 4. Summary of Notation

Symbol	Description
λ	Security parameter
$\mathbb{G}$, q	Prime-order subgroup $\mathbb{G}_1$ of BN254 and its order, $q \approx 2^{254}$
G, H	Independent generators of $\mathbb{G}$ with $\log_G H$ unknown
x, $P = xG$	Secret scalar and public commitment
$I = xH$	Key image (nullifier)
R, n, π	Ring of public commitments, ring size, secret signer index
m	Signed message (32-byte recipient public key)
$\sigma = (c_0, s_0, \ldots, s_{n-1})$	LSAG signature
$\mathcal{H}$	Challenge hash $\{0,1\}^* \to \mathcal{D}$, modeled as a random oracle
$\mathcal{D}$	Challenge domain $[0, 2^{255})$ induced by the masked digest
$\mathcal{B}_C$, $\mathcal{B}_N$	Commitment and nullifier boxes in on-chain storage
q_H, q_S	Number of adversarial random-oracle and signing-oracle queries

A.2 Computational Assumptions

Definition 1 (Elliptic Curve Discrete Logarithm Problem (ECDLP)). *Given a generator $G \in \mathbb{G}$ and a point $P \in \mathbb{G}$, it is computationally infeasible for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ to output $x \in \mathbb{Z}_q$ such that $P = xG$; the success probability of any PPT $\mathcal{A}$ is negligible in λ.*

Definition 2 (Decisional Diffie–Hellman (DDH) Assumption [16]). *Given a tuple $(G, aG, bG, Z) \in \mathbb{G}^4$ for uniformly sampled $a, b \xleftarrow{\$} \mathbb{Z}_q$, it is computationally infeasible for any PPT adversary to distinguish whether $Z = abG$ or $Z = cG$ for a uniformly sampled $c \xleftarrow{\$} \mathbb{Z}_q$.*

Because BN254 is a pairing-friendly curve, the hardness of DDH in $\mathbb{G} = \mathbb{G}_1$ is precisely the *external Diffie–Hellman* (XDH) assumption [9, 17]: DDH is conjectured hard in $\mathbb{G}_1$ of a Type-3 pairing configuration because no efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ is known. The self-pairing test that renders DDH easy in symmetric pairing groups is unavailable here, since the pairing requires one argument from each of $\mathbb{G}_1$ and $\mathbb{G}_2$, while all Obscura protocol elements lie in $\mathbb{G}_1$.

Assumption 1 (Generator Independence). *The secondary generator H is derived by applying a hash-to-curve map [25] to a fixed, public domain-separation string, and this map is modeled as a random oracle into $\mathbb{G}$. Consequently, (i) no party knows $\log_G H$, and (ii) security reductions in the ROM may program the map, i.e., substitute a challenge point for H.*

Assumption 1 is necessary, not merely convenient: if some party knew $k = \log_G H$, it could compute $I = kP$ for every on-chain commitment P without knowledge of any secret scalar, linking all deposits to their nullifiers and collapsing the anonymity of the entire system. The nothing-up-my-sleeve derivation renders the assumption publicly verifiable.

Security Parameter Selection. The BN254 group order satisfies $q \approx 2^{254}$, so generic discrete-logarithm algorithms (e.g., Pollard's rho [56]) require approximately $\sqrt{q} \approx 2^{127}$ group operations, providing a 127-bit security level for the ECDLP and XDH instances used above. We note that tower-number-field-sieve advances [10, 34] degrade the security of the *pairing* groups of BN254 (i.e., the DLP in the embedding field $\mathbb{F}_{p^{12}}$) to roughly 100 bits; Obscura, however, performs no pairing computations and relies solely on the elliptic-curve group law in $\mathbb{G}_1$, whose 127-bit security level is unaffected.

A.3 Scheme Syntax and Correctness

Definition 3 (Linkable Ring Signature Scheme). *An LSAG signature scheme is a triple of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*:*

- $(x, P) \leftarrow \mathsf{KeyGen}(1^\lambda)$*: outputs a secret scalar* $x \xleftarrow{\$} \mathbb{Z}_q$ *and the public commitment* $P = xG$*;*
- $(\sigma, I) \leftarrow \mathsf{Sign}(x, R, m)$*: on input a secret key* x*, a ring* R *containing* $P = xG$ *at a secret index* π*, and a message* m*, outputs a signature* $\sigma = (c_0, s_0, \ldots, s_{n-1})$ *and the key image* $I = xH$*;*
- $\{0, 1\} \leftarrow \mathsf{Verify}(R, m, I, \sigma)$*: outputs* 1 *(accept) or* 0 *(reject).*

The scheme is correct *if, for every message* m*, every honestly generated key pair* (x, P)*, and every ring* R *containing* P*,* $\mathsf{Verify}(R, m, I, \sigma) = 1$ *whenever* $(\sigma, I) \leftarrow \mathsf{Sign}(x, R, m)$*.*

Theorem 1 (Correctness). *The Obscura instantiation of* Sign *and* Verify *(Section 3) is correct: an honestly generated signature is always accepted.*

Proof. Let π be the true signer's index. During signing, the prover sets $L_\pi = \alpha G$, $R_\pi = \alpha H$, and $c_{\pi+1} = \mathcal{H}(m \| L_\pi \| R_\pi)$; for each decoy index $i \neq \pi$ it samples s_i and computes $L_i = s_i G + c_i P_i$, $R_i = s_i H + c_i I$, and $c_{i+1} = \mathcal{H}(m \| L_i \| R_i)$; finally it closes the ring with $s_\pi = \alpha - c_\pi x \pmod q$.

During verification, at every decoy index the verifier recomputes exactly the prover's points, since it uses the same s_i, c_i, P_i, and I. At the true index π it

computes

$$L_\pi = s_\pi G + c_\pi P_\pi = (\alpha - c_\pi x)G + c_\pi(xG) = \alpha G,$$
$$R_\pi = s_\pi H + c_\pi I = (\alpha - c_\pi x)H + c_\pi(xH) = \alpha H,$$

which coincide with the prover's initialization. Hence every challenge in the verifier's chain equals the corresponding challenge in the prover's chain, and after the final iteration the running challenge equals c_0, so Verify accepts. □

A.4 Distribution of the Fiat–Shamir Challenges

The protocol instantiates $\mathcal{H}(z) = \text{SHA-256}(z) \wedge \text{mask}$, where the mask clears the most significant bit of the digest. In the ROM, the output is therefore uniform over the domain $\mathcal{D} = [0, 2^{255})$. Since the BN254 group order satisfies $q < 2^{255} < 3q$, challenges are *not* uniform modulo q; scalars $c \geq q$ are reduced implicitly by the `EcScalarMul` opcode. Correctness is unaffected, because the prover and the verifier compute identical masked digests and the final ring-closure check compares the 32-byte strings directly. The following lemma establishes that this instantiation also preserves every property required by the security proofs.

Lemma 1 (Challenge Distribution). *Model $\mathcal{H}$ as a random oracle with outputs uniform on $\mathcal{D} = [0, 2^{255})$. Then: (i) any challenge not yet queried is unpredictable except with probability 2^{-255}; (ii) for any fixed $c \in \mathcal{D}$ and a fresh uniform $c' \in \mathcal{D}$, $\Pr[c' \equiv c \pmod q \wedge c' \neq c] \leq 3 \cdot 2^{-255}$, so the difference $c' - c$ is invertible modulo q with overwhelming probability; and (iii) the joint distribution of the challenges is determined by the random oracle alone and is independent of the signer index.*

Proof. (i) follows directly from the uniformity of the oracle output on $\mathcal{D}$. For (ii), the number of elements of $\mathcal{D}$ congruent to c modulo q is at most $\lceil |\mathcal{D}|/q \rceil = 3$, so a fresh uniform c' collides with the residue class of c with probability at most $3/2^{255}$; whenever $c' \not\equiv c \pmod q$, the difference is a nonzero element of $\mathbb{Z}_q$ and hence invertible, as q is prime. For (iii), every challenge is the image of a distinct oracle query, and the oracle's answers are sampled independently of the signer index. □

Property (i) guarantees that a forger cannot anticipate the ring-closure condition; property (ii) guarantees that the extraction steps of Theorems 2 and 4, which divide by challenge differences modulo q, succeed with overwhelming probability; property (iii) underpins the signer-ambiguity argument of Theorem 3.

A.5 Formal Security Model

The security of the Obscura protocol relies on three properties: existential unforgeability, signer ambiguity, and linkability. In all experiments the adversary $\mathcal{A}$ is a PPT algorithm with access to the random oracle $\mathcal{H}$.

Definition 4 (Existential Unforgeability against Adaptive Chosen-Message and Chosen-Public-Key Attacks). *Let $\mathcal{SO}(R, m)$ be a signing oracle that accepts any public key list R and any message m, and produces a valid signature σ. An LSAG signature scheme is existentially unforgeable if, for any PPT adversary $\mathcal{A}$ with access to $\mathcal{SO}$ and the random oracle $\mathcal{H}$, the probability that $\mathcal{A}$ outputs a valid forgery (m^*, R^*, σ^*) is negligible. The forgery must satisfy $\mathsf{Verify}(R^*, m^*, I^*, \sigma^*) = 1$, and (m^*, R^*) must not correspond to any query–response pair of $\mathcal{SO}$. The interaction between the challenger and the adversary for this security experiment is illustrated in Figure 12.*

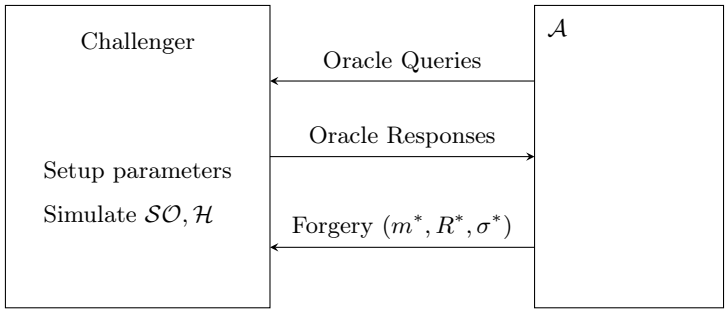


Fig. 12. Existential Unforgeability (EUF-CMA) Security Experiment.

Definition 5 (Signer Ambiguity). *Consider the following experiment, illustrated in Figure 13. The adversary $\mathcal{A}$ selects a message m, a ring R of n honestly generated public keys, and a set D_t of t corrupted secret keys ($0 \leq t < n - 1$) whose corresponding public keys lie in R. The challenger samples the signer index π uniformly from the non-corrupted indices, computes $(\sigma, I) \leftarrow \mathsf{Sign}(x_\pi, R, m)$, and returns (σ, I) to $\mathcal{A}$, who outputs a guess $\hat{\pi}$. An LSAG signature scheme is* signer ambiguous *if for every PPT adversary $\mathcal{A}$,*

$$\Pr[\hat{\pi} = \pi] \leq \frac{1}{n - t} + \epsilon(\lambda)$$

for some negligible function ϵ.

Definition 6 (Linkability). *An LSAG signature scheme is* linkable *if there exists a PPT algorithm Link that, given two valid signature tuples (R, m_1, I_1, σ_1) and (R, m_2, I_2, σ_2), outputs 1 if and only if the two signatures were generated using the same secret key, except with negligible probability. In Obscura, Link outputs 1 if and only if $I_1 = I_2$. The property requires that no PPT adversary in possession of at most one secret key corresponding to the ring R can output two valid signatures with distinct key images $I_1 \neq I_2$, nor a valid signature whose key image differs from xH for the secret key x it used.*

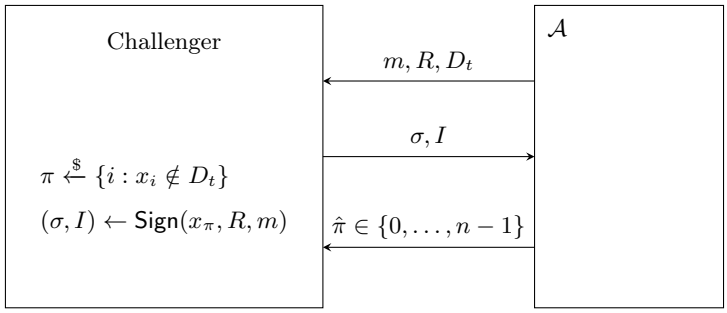


Fig. 13. Signer Ambiguity Security Experiment.

A.6 Proof of Existential Unforgeability

Theorem 2. *The Obscura LSAG signature scheme is existentially unforgeable against adaptive chosen-message and adaptive chosen-public-key attacks in the random oracle model, provided the ECDLP is hard in* $\mathbb{G}$.

Proof. We construct a PPT reduction $\mathcal{M}$ that uses a successful forging adversary $\mathcal{A}$ to solve the ECDLP. Let P^* be the ECDLP challenge point; $\mathcal{M}$ aims to find x^* such that $P^* = x^*G$. $\mathcal{M}$ generates N key pairs honestly, replaces the public key at a uniformly random position with P^*, publishes the resulting key list (all keys are identically distributed, so the embedding position is information-theoretically hidden from $\mathcal{A}$), and answers oracle queries as follows.

Simulation of $\mathcal{H}$: $\mathcal{M}$ answers by lazy sampling: for any new query it returns a uniform value from $\mathcal{D}$ and records the pair to maintain consistency across duplicated queries.

Simulation of $\mathcal{SO}$: When $\mathcal{A}$ requests a signature on message m for a ring $R = \{P_0, \ldots, P_{n-1}\}$, $\mathcal{M}$ simulates the signature without knowing any secret key by back-patching the random oracle. $\mathcal{M}$ selects an index $\pi \xleftarrow{\$} \{0, \ldots, n-1\}$ and samples $c_0, s_0, \ldots, s_{n-1}$ uniformly. To keep key images consistent with a real signer, $\mathcal{M}$ maintains a table assigning to each public key P_i a fixed scalar $r_i \xleftarrow{\$} \mathbb{Z}_q$ and reuses the key image $I = r_\pi H$ across all queries simulated for P_π. For each $i \neq \pi$, $\mathcal{M}$ computes $L_i = s_iG + c_iP_i$ and $R_i = s_iH + c_iI$, obtaining $c_{i+1} = \mathcal{H}(m\|L_i\|R_i)$ from the simulated oracle. For the index π, $\mathcal{M}$ computes $L_\pi = s_\pi G + c_\pi P_\pi$ and $R_\pi = s_\pi H + c_\pi I$, and *programs* the oracle so that $\mathcal{H}(m\|L_\pi\|R_\pi) = c_{\pi+1}$ (interpreting the index modulo n). Programming fails only if this input was previously queried; since s_π is uniform, L_π is uniform in $\mathbb{G}$, so a collision with any of the at most $q_H + nq_S$ recorded queries occurs with probability at most $(q_H + nq_S)/q$, which is negligible. The resulting transcript is distributed identically to a real signature.

Extraction via Rewinding: Suppose $\mathcal{A}$ outputs a valid forgery (m, R, σ) with $\sigma = (c_0, s_0, \ldots, s_{n-1})$. Except with negligible probability, all n oracle queries used in verification appear among the recorded queries (otherwise $\mathcal{A}$ must have guessed an unqueried oracle output, which by Lemma 1(i) succeeds with

probability at most 2^{-255}). Let the *gap index* π denote the ring position whose challenge corresponds to the earliest of these queries. By the Forking Lemma [55], $\mathcal{M}$ rewinds $\mathcal{A}$ to that critical query and supplies fresh oracle responses; with non-negligible probability $\mathcal{A}$ outputs a second valid forgery $\sigma' = (c'_0, s'_0, \ldots, s'_{n-1})$ for the same m and R, agreeing with the first transcript up to the fork, with $c'_\pi \neq c_\pi$.

Both transcripts satisfy the verification equations at the gap index with the same points L_π, R_π (they are determined before the fork):

$$L_\pi = s_\pi G + c_\pi P_\pi = s'_\pi G + c'_\pi P_\pi,$$
$$R_\pi = s_\pi H + c_\pi I = s'_\pi H + c'_\pi I.$$

By Lemma 1(ii), $(c'_\pi - c_\pi)$ is invertible modulo q with overwhelming probability. Setting $x_\pi = (s_\pi - s'_\pi)(c'_\pi - c_\pi)^{-1} \bmod q$, the first equation yields $P_\pi = x_\pi G$, and the second yields $I = x_\pi H$: the extracted witness simultaneously opens the ring member *and* the key image (this second identity is the basis of Theorem 4).

Since the forgery involves no corrupted keys and the embedding position of P^* is uniform and independent of $\mathcal{A}$'s view, $P_\pi = P^*$ holds with probability at least $1/N$, in which case $\mathcal{M}$ outputs $x^* = x_\pi$ and solves the ECDLP. As the ECDLP is hard in $\mathbb{G}$, the forgery probability of $\mathcal{A}$ must be negligible. □

Corollary 1 (Concrete Security). *Suppose $\mathcal{A}$ makes at most q_H random-oracle queries and q_S signing-oracle queries, and outputs a valid forgery with probability at least $1/Q(\lambda)$ for some polynomial Q. Then there exists an algorithm that solves the ECDLP in $\mathbb{G}$ with probability at least $\left(\frac{1}{n(q_H+nq_S)\,Q(\lambda)}\right)^2$ and expected running time at most twice that of $\mathcal{A}$, following the rewind-on-success analysis of [37].*

A.7 Proof of Signer Ambiguity

Theorem 3. *Under Assumption 1, the Obscura LSAG signature scheme is signer ambiguous in the random oracle model, provided the DDH (XDH) problem is hard in $\mathbb{G}$.*

Proof. We prove the case $t = 0$; the cases $0 < t < n - 1$ follow by the same argument with the corrupted keys handed to the adversary. Suppose a PPT adversary $\mathcal{A}$ wins the experiment of Definition 5 with probability at least $\frac{1}{n} + \epsilon$ for non-negligible ϵ. We construct a PPT distinguisher $\mathcal{M}$ for the DDH problem.

$\mathcal{M}$ is given a DDH challenge tuple $(G, \alpha G, \beta G, Z) \in \mathbb{G}^4$, where Z is either $\alpha\beta G$ or a uniformly random point γG, and must decide which. $\mathcal{M}$ programs the hash-to-curve oracle of Assumption 1 so that the secondary generator becomes $H = \beta G$. This step is where the reduction crucially relies on H being a programmable random-oracle output rather than an arbitrary fixed constant; since βG is uniformly distributed in $\mathbb{G}$, the programmed oracle is perfectly indistinguishable from an unprogrammed one.

$\mathcal{M}$ selects a target index $\pi \xleftarrow{\$} \{0, \ldots, n-1\}$, sets $P_\pi = \alpha G$, generates all other ring members honestly as $P_i = x_i G$ with known x_i, and sets the key image

$I = Z$. $\mathcal{M}$ then simulates a signature σ for the ring R and key image I using the back-patching technique of Theorem 2, which requires no secret key, and runs $\mathcal{A}$ on (m, R, σ, I).

$\mathcal{A}$ outputs a guess $\hat{\pi}$, and we distinguish the two cases of the challenge:

- If $Z = \alpha\beta G$, then $I = \alpha(\beta G) = \alpha H$ is the well-formed key image for $P_\pi = \alpha G$, and the simulated transcript is distributed identically to a real signature by the member at index π: all responses s_i are uniform, and by Lemma 1(iii) the challenges are random-oracle outputs whose distribution is independent of the signer index. By assumption, $\Pr[\hat{\pi} = \pi] \geq \frac{1}{n} + \epsilon$.
- If $Z = \gamma G$ for uniform γ, then I is a uniformly random point independent of every ring member. Every component of $\mathcal{A}$'s view (the ring, the uniform responses, the oracle-derived challenges, and the independent key image) is distributed independently of π, so $\Pr[\hat{\pi} = \pi] = \frac{1}{n}$ exactly.

$\mathcal{M}$ outputs 1 if and only if $\hat{\pi} = \pi$. Its distinguishing advantage is therefore

$$\Pr[\mathcal{M} = 1 \mid Z = \alpha\beta G] - \Pr[\mathcal{M} = 1 \mid Z = \gamma G] \geq \left(\tfrac{1}{n} + \epsilon\right) - \tfrac{1}{n} = \epsilon,$$

contradicting the hardness of DDH in $\mathbb{G}$. Hence ϵ must be negligible. □

Remark 1 (Culpability and Selective Disclosure). When $t = n - 1$, or whenever the true signer's secret key is disclosed, any party can test $x_i H \stackrel{?}{=} I$ and identify the signer with certainty; this is why Definition 5 requires $t < n - 1$. Such *culpability* is inherent to linkable ring signatures and is precisely the mechanism enabling the user-controlled auditability of Section 7: by revealing x, a user proves both the deposit commitment $P = xG$ and the withdrawal key image $I = xH$ to an auditor, without affecting the ambiguity of any other participant's transactions.

A.8 Proof of Linkability

Theorem 4. *The Obscura LSAG signature scheme is linkable in the random oracle model, provided the ECDLP is hard in* $\mathbb{G}$.

Proof. The proof proceeds in two steps.

Step 1: every valid signature carries a well-formed key image. The rewinding argument of Theorem 2 applies to any algorithm producing a valid signature. Forking at the gap index π yields two accepting transcripts sharing the points L_π, R_π, and hence the pair of equations

$$(s_\pi - s'_\pi)\, G = (c'_\pi - c_\pi)\, P_\pi, \qquad (s_\pi - s'_\pi)\, H = (c'_\pi - c_\pi)\, I,$$

where $(c'_\pi - c_\pi)$ is invertible modulo q by Lemma 1(ii). Writing $x_\pi = (s_\pi - s'_\pi)(c'_\pi - c_\pi)^{-1} \bmod q$, the first equation gives $P_\pi = x_\pi G$ and the second gives $I = x_\pi H$ *with the same witness* x_π. Hence, except with negligible probability, any signature accepted by Verify has key image $I = x_\pi H$, where x_π is the unique discrete logarithm of some ring member P_π.

Step 2: identical keys force identical key images. Since $\mathbb{G}$ has prime order, the discrete logarithm of each ring member with respect to G is unique in $\mathbb{Z}_q$. Suppose an adversary in possession of a single secret key x outputs two valid signatures σ_1, σ_2 for rings containing $P = xG$. By Step 1, their key images satisfy $I_1 = x_{\pi_1} H$ and $I_2 = x_{\pi_2} H$ for extracted witnesses x_{π_1}, x_{π_2}. If both witnesses equal x, then $I_1 = xH = I_2$, and Link outputs 1. Producing $I_1 \neq I_2$ therefore requires the two extracted witnesses to differ, i.e., the adversary must know the discrete logarithm of a *second* ring member, which contradicts the ECDLP assumption for an adversary holding only one secret key. □

Corollary 2 (Double-Spend Resistance). *In the Obscura protocol, each deposit can be withdrawn at most once. By Theorem 4, every valid withdrawal spending the commitment $P = xG$ must present the key image $I = xH$. The smart contract asserts the non-existence of the nullifier box $\mathcal{B}_N$ before verification and records I in $\mathcal{B}_N$ atomically with the payout; any second withdrawal attempt for the same deposit necessarily presents the identical I and deterministically reverts, regardless of the decoy set chosen for the second ring.*

A.9 Protocol-Level Security

We now lift the signature-level guarantees to the on-chain protocol, considering malicious users, malicious relayers and block proposers, and passive blockchain observers.

Lemma 2 (Non-Frameability). *No PPT adversary can produce a valid signature whose key image equals $I = xH$ for an honest user's secret key x, except with negligible probability.*

Proof. By Step 1 of Theorem 4, any valid signature with key image I yields, under rewinding, a witness x_π satisfying both $I = x_\pi H$ and $P_\pi = x_\pi G$ for some ring member P_π. Since discrete logarithms in a prime-order group are unique, $I = xH$ forces $x_\pi = x$, so a framing adversary can be used to extract the honest user's secret key, contradicting the ECDLP assumption. (Note also that computing I directly from the public P without x is an instance of the Computational Diffie–Hellman problem, given Assumption 1.) In particular, an adversary can neither preemptively "spend" an honest user's deposit nor cause an honest withdrawal to be falsely rejected as a double-spend. □

Lemma 3 (Replay Resistance and Destination Binding). *A valid withdrawal transaction cannot be replayed, and its recipient address cannot be altered, except with negligible probability.*

Proof. The recipient address m is bound into every challenge of the Fiat–Shamir chain, $c_{i+1} = \mathcal{H}(m \| L_i \| R_i)$. Substituting a different recipient $m' \neq m$ changes every random-oracle input; by Lemma 1(i), the probability that the mutated chain still closes (i.e., the recomputed final challenge equals c_0) is at most 2^{-255} per attempt. An active network adversary (e.g., a malicious relayer or block

proposer) who intercepts a pending withdrawal therefore cannot redirect the payout without producing a fresh forgery, contradicting Theorem 2. Replaying the unmodified transaction is prevented unconditionally by Corollary 2, as the key image I is recorded in $\mathcal{B}_N$ upon first execution. □

Blockchain Observers and Transaction Integrity. A globally passive adversary observes the ring R, the key image I, the recipient m, and the signature σ of every withdrawal. By Theorem 3, it identifies the true signer with probability at most $1/n$ plus a negligible term; the same reduction, together with Assumption 1, shows that it cannot link any deposit commitment P to its nullifier I. The AVM's atomic group execution further guarantees transaction integrity: the nullifier is recorded if and only if the signature verifies, and funds are disbursed if and only if the nullifier is recorded, leaving no reachable partial state for any interleaving of adversarial transactions.

Trust Assumptions and Limitations. The guarantees above are conditional on the trust model of Section 7: the Sign algorithm must execute in a trusted local environment, since any party observing the secret scalar x trivially breaks both anonymity and non-frameability. The anonymity bound $1/n$ is a cryptographic ceiling; operational behavior (biased decoy selection, temporal correlation between deposits and withdrawals, small effective pools) can only degrade it, as discussed in Section 7. Finally, all reductions in this appendix are classical; quantum adversaries, which invalidate the ECDLP and XDH assumptions, are outside the model (cf. the post-quantum discussion in Section 7).

Complexity of Cryptographic Operations. The on-chain verification requires exactly n iterations. Each iteration performs four scalar multiplications and two point additions on $\mathbb{G}$ (two of each for L_i and for R_i), plus one hash evaluation, matching the per-member cost accounting of Table 1. The computational complexity is therefore $O(n)$. The AVM's deterministic opcode pricing ensures that this complexity translates to a fixed execution cost of $7608n + O(1)$ opcodes. The dynamic opcode pooling mechanism via inner transactions scales linearly to meet this requirement, guaranteeing that the protocol remains executable up to the AVM's maximum argument size limit ($n = 19$), without introducing unbounded loops or resource exhaustion vectors.

References

1. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: International conference on the theory and application of cryptology and information security. pp. 415–432. Springer (2002)
2. Algorand Foundation: Algorand virtual machine (avm) specification, https://developer.algorand.org/docs/get-details/dapps/avm/, last accessed: 2026-04-25
3. Algorand Foundation: Avm opcodes, https://dev.algorand.co/reference/algorand-teal/opcodes/, last accessed: 2026-07-05

4. Algorand Foundation: Box storage, https://dev.algorand.co/concepts/smart-contracts/storage/box/, last accessed: 2026-05-03
5. Algorand Foundation: Pyteal: Algorand smart contracts in python. https://github.com/algorand/pyteal, last accessed: 2026-05-03
6. Algorand Foundation: Smart contract costs & constraints, https://dev.algorand.co/concepts/smart-contracts/costs-constraints/, last accessed: 2026-05-03
7. Algorand Foundation: Transaction fees, https://dev.algorand.co/concepts/transactions/fees/#pooled-transaction-fees, last accessed: 2026-05-03
8. Androulaki, E., Karame, G.O., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating user privacy in bitcoin. In: International conference on financial cryptography and data security. pp. 34–51. Springer (2013)
9. Ballard, L., Green, M., De Medeiros, B., Monrose, F.: Correlation-resistant storage via keyword-searchable encryption. IACR Cryptol. ePrint Arch. **2005**, 417 (2005)
10. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. Journal of cryptology **32**(4), 1298–1336 (2019)
11. Barreto, P.S., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: International workshop on selected areas in cryptography. pp. 319–331. Springer (2005)
12. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on computer and communications security. pp. 62–73. ACM (1993)
13. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018)
14. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE symposium on security and privacy (SP). pp. 459–474. IEEE (2014)
15. Béres, F., Seres, I.A., Benczúr, A.A., Quintyne-Collins, M.: Blockchain is watching you: Profiling and deanonymizing ethereum users. In: 2021 IEEE international conference on decentralized applications and infrastructures (DAPPS). pp. 69–78. IEEE (2021)
16. Boneh, D.: The decision diffie-hellman problem. In: International algorithmic number theory symposium. pp. 48–63. Springer (1998)
17. Boneh, D., Boyen, X.: Short signatures without random oracles. In: International conference on the theory and applications of cryptographic techniques. pp. 56–73. Springer (2004)
18. Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J.A., Felten, E.W.: Mixcoin: Anonymity for bitcoin with accountable mixes. In: International conference on financial cryptography and data security. pp. 486–504. Springer (2014)
19. Bos, J.W., Costello, C., Miele, A.: Elliptic and hyperelliptic curves: A practical security analysis. In: International workshop on public key cryptography. pp. 203–220. Springer (2014)
20. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International conference on financial cryptography and data security. pp. 423–443. Springer (2020)
21. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE symposium on security and privacy (SP). pp. 315–334. IEEE (2018)
22. Chaum, D., van Heyst, E.: Group signatures. In: Workshop on the theory and application of cryptographic techniques. pp. 257–265. Springer (1991)

23. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Annual international cryptology conference. pp. 174–187. Springer (1994)
24. Esgin, M.F., Zhao, R.K., Steinfeld, R., Liu, J.K., Liu, D.: Matrict: Efficient, scalable and post-quantum blockchain confidential transactions protocol. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 567–584. ACM (2019)
25. Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R., Wood, C.: Rfc 9380: Hashing to elliptic curves (2023)
26. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
27. Fisher, R.A., Yates, F.: Statistical tables for biological, agricultural and medical research. Oliver and Boyd (1953)
28. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: International conference on financial cryptography and data security. pp. 81–98. Springer (2016)
29. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th symposium on operating systems principles. pp. 51–68. ACM (2017)
30. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on computing **18**(1), 186–208 (1989)
31. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on computing **17**(2), 281–308 (1988)
32. Goodell, B., Noether, S., RandomRun: Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Report 2019/654 (2019)
33. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016)
34. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: Annual international cryptology conference. pp. 543–571. Springer (2016)
35. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of computation **48**(177), 203–209 (1987)
36. Kumar, A., Fischer, C., Tople, S., Saxena, P.: A traceability analysis of monero's blockchain. In: European symposium on research in computer security. pp. 153–173. Springer (2017)
37. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups. In: Australasian conference on information security and privacy. pp. 325–335. Springer (2004)
38. Liu, J.K., Wong, D.S.: Linkable ring signatures: Security models and new schemes. In: International Conference on Computational Science and Its Applications. pp. 614–623. Springer (2005)
39. Lu, X., Au, M.H., Zhang, Z.: Raptor: A practical lattice-based (linkable) ring signature. In: International conference on applied cryptography and network security. pp. 110–130. Springer (2019)
40. Maxwell, G.: Confidential transactions. https://elementsproject.org/features/confidential-transactions (2015)
41. MDN Web Docs: Crypto.getrandomvalues() - web apis, https://developer.mozilla.org/docs/Web/API/Crypto/getRandomValues, last accessed: 2026-05-03

42. Meiklejohn, S., Mercer, R.: Möbius: Trustless tumbling for transaction privacy. Proceedings on Privacy Enhancing Technologies **2018**(2), 105–121 (2018)
43. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: characterizing payments among men with no names. In: Proceedings of the 2013 internet measurement conference. pp. 127–140. ACM (2013)
44. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Conference on the theory and application of cryptographic techniques. pp. 369–378. Springer (1987)
45. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: 2013 IEEE symposium on security and privacy (SP). pp. 397–411. IEEE (2013)
46. Miller, V.S.: Use of elliptic curves in cryptography. In: Conference on the theory and application of cryptographic techniques. pp. 417–426. Springer (1985)
47. Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., Hogan, K., Hennessey, J., Miller, A., Narayanan, A., Christin, N.: An empirical analysis of traceability in the monero blockchain. Proceedings on Privacy Enhancing Technologies **2018**(3), 143–163 (2018)
48. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
49. National Institute of Standards and Technology: Secure hash standard (shs). FIPS PUB 180-4 (2015)
50. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
51. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)
52. Pera Wallet: Pera wallet connect: Javascript sdk for web dapp integration. https://github.com/perawallet/connect, last accessed: 2026-05-03
53. Pertsev, A., Semenov, R., Storm, R.: Tornado cash privacy solution version 1.4. Tornado cash privacy solution version **1**(6) (2019)
54. Pfitzmann, A., Hansen, M.: A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. v0.34, TU Dresden (2010)
55. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of cryptology **13**(3), 361–396 (2000)
56. Pollard, J.M.: Monte carlo methods for index computation (mod p). Mathematics of computation **32**(143), 918–924 (1978)
57. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: International conference on the theory and application of cryptology and information security. pp. 552–565. Springer (2001)
58. Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: International conference on financial cryptography and data security. pp. 6–24. Springer (2013)
59. van Saberhagen, N.: Cryptonote v 2.0. https://www.getmonero.org/ru/resources/research-lab/pubs/whitepaper_annotated.pdf (2013)
60. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. pp. 124–134. Ieee (1994)
61. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014)