Ghost Litepaper

Private USDC and gasless transfers on Solana

Ghost Labs (ghostlabshq.xyz)

November 2026, v0.1

1. Overview

Ghost is a privacy suite for USDC on Solana. It combines four pieces into one product:

1. **Ghost Shield** is a fixed-denomination privacy pool.
   - A user deposits $D = 1$ USDC together with a commitment C.
   - Later, any fresh address can withdraw $D - f$ with a Groth16 zero-knowledge proof. The proof shows the withdrawer knows the opening of some commitment in the pool, without revealing which one.
2. **Private withdrawals through the Ghost relayer.** The relayer pays the network fee, so the recipient wallet never needs SOL. That removes the funding transaction that would otherwise link the wallets.
3. **Ghost Gas** turns prepaid USDC into gas.
   - Wallet X1 tops up a gas note.
   - Wallet X2, holding **zero SOL**, can then send SPL tokens. Ghost is the fee payer and debits the note.
4. **The Ghost SDK** (`@ghost/sdk`) generates notes, Poseidon commitments, Merkle paths and proofs **on the user's device**. Secrets never leave the client.

Every formula below is implemented in the Ghost codebase. File references are given inline.

2. Notation and parameters

Symbol	Value	Source
p	BN254 scalar field, $p \approx 2^{253.6}$	`sdk/src/recipient.ts`
H_1, H_2	Poseidon with 1 and 2 inputs (circomlib / circomlibjs)	`circuits/merkle_proof.circom`, `sdk/src/poseidon.ts`
d	Merkle depth $= 10$	`TREE_DEPTH`, `GhostCommitment(10)`
N	Capacity $2^d = 1024$ leaves per pool	`MAX_LEAVES`
D	Denomination $= 10^6$ base units (1 USDC)	`DENOMINATION`
β	Shield fee in bps: default 100 (1%), cap 1000	`DEFAULT_FEE_BPS`, `MAX_FEE_BPS`
γ	Gas top-up protocol fee in bps: default 100 (1%)	`GAS_PROTOCOL_FEE_BPS`

3. Commitment scheme

The client samples a 31-byte random secret s and a 31-byte random nullifier n. Both are $< p$, so no reduction is needed. It then computes:

$$C = H_2(s, n), \qquad h = H_1(n).$$

- C is recorded on-chain in two PDAs:
  - `CommitmentRecord` at seeds `["commitment", C]`, which makes each commitment unique;
  - `LeafRecord` at seeds `["leaf", mint, u32_le(i)]`, where i is `pool.next_index`.
- The user keeps a **Ghost Note** `ghost:v1:s:n:amount:USDC:i:C`. It is a bearer secret: whoever holds it can withdraw.
- **Hiding.** C reveals nothing about (s, n), given that Poseidon behaves as a random oracle and s carries 248 bits of entropy.
- **Binding.** Opening C two different ways requires a Poseidon collision.

4. Merkle tree and anonymity set

- Leaves are $C_0, \ldots, C_{k-1}$. Empty leaves are $z_0 = 0$, and $z_j = H_2(z_{j-1}, z_{j-1})$.
- Each internal node is $H_2(\text{left}, \text{right})$, and the root R sits at depth 10 (`sdk/src/merkle.ts`).
- The tree is rebuilt from the on-chain `LeafRecord` PDAs (`relayer/src/sync.ts`). Every historical root is kept (`isKnownRoot`), so a proof generated against an earlier root remains valid.
- **Anonymity set.** A withdrawal made after k deposits is indistinguishable among those k deposits, so $k \leq 1024$ per pool. The fixed denomination removes amount fingerprinting. Waiting longer and using a fresh recipient both increase the effective set. Once a pool fills (`TreeFull`), new deposits go to a new pool.

5. Nullifiers and double-spend protection

- The public input $h = H_1(n)$ is deterministic per note.
- `withdraw` creates a `NullifierRecord` PDA at seeds `["nullifier", h]` with Anchor `init`. A second withdrawal of the same note fails because the account already exists, and the program enforces this on-chain.
- h cannot be linked to C without inverting Poseidon.
- Public signals must be canonical field elements ($< p$). snarkjs `groth16Verify` enforces this (`publicInputsAreValid`), so a nullifier has exactly one on-chain encoding.

6. The zero-knowledge statement

`circuits/merkle_proof.circom`, template `GhostCommitment(10)`:

- Public inputs: (R, h, r).
- Private inputs: $(s, n, \text{path}_{0..9}, b_{0..9})$.

The circuit enforces:

1. $h = H_1(n)$
2. $\ell = H_2(s, n)$
3. $b_j(1 - b_j) = 0$ for every j, so each path index is a bit
4. $x_0 = \ell$, and $x_{j+1} = H_2(\text{mux}_{b_j}(x_j, \text{path}_j))$, where the mux chooses left/right order
5. $R = x_{10}$
6. $r \cdot r$ is constrained, so the recipient r is bound into the R1CS

Soundness. Groth16 is knowledge-sound in the algebraic group model. A valid proof therefore implies the prover knows (s, n) such that $H_2(s, n)$ is a leaf under R and $H_1(n) = h$.

Recipient binding. The recipient is a public input, $r = \text{int}_{BE}(\text{pubkey}) \bmod p$. Changing the recipient invalidates the proof, so a withdrawal cannot be redirected or front-run. The relayer additionally checks that `publicSignals[2]` matches the exact requested recipient.

Setup. Proving keys come from a Powers of Tau (2^{15}) plus a circuit-specific phase-2 contribution (`circuits/scripts/ptau.mjs`, `build.mjs`). The verification key is public.

6.1 Verification pipeline

Property	Enforced by
Exactly D in, unique non-zero C, tree capacity	`ghost_pool::deposit` (on-chain)
Each nullifier used once	`NullifierRecord` PDA `init` (on-chain)
Fee split $D - f$ / f and fee cap	`ghost_pool::withdraw`, `set_fee_bps` (on-chain)
Withdrawals submitted only by the designated relayer	`pool.relayer` signer check (on-chain)
Groth16 proof, root $\in$ history, recipient binding	Ghost relayer (snarkjs), then on-chain `alt_bn128` verification on the roadmap

7. Fee model and treasury

7.1 Shield withdrawal fee (on-chain)

$$f = \left\lfloor \frac{D \cdot \beta}{10^4} \right\rfloor, \qquad \text{payout} = D - f, \qquad 0 \le \beta \le 1000.$$

The program computes this in `u128` checked arithmetic. With $\beta = 100$: - $f = 10{,}000$ base units (0.01 USDC) goes to the treasury in the same transaction; - the payout is $990{,}000$ (0.99 USDC).

The pool authority can tune β with `set_fee_bps`. It is hard-capped at 10%, and every change emits a `FeeUpdated` event.

Withdrawals	Volume	Treasury (1%)
1,000	1,000 USDC	10 USDC
100,000	100,000 USDC	1,000 USDC

Illustrative arithmetic, not a forecast.

7.2 Gas fees (relayer ledger)

For a confirmed top-up of amount a:

$$v = \begin{cases} a & \text{USDC} \\ \lfloor q(a) \cdot (10^4 - \eta)/10^4 \rfloor & \text{other allowlisted token} \end{cases}$$

$$\phi = \lfloor v\gamma/10^4 \rfloor, \qquad \text{credit}_0 = v - \phi.$$

- q is a Jupiter price quote, and $\eta = 200$ bps is the pricing haircut.
- ϕ is the 1% protocol fee, kept by the treasury.

- Each sponsored transfer debits $c = 10{,}000$ (0.01 USDC), so a note supports $\lfloor \text{credit}_0 / c \rfloor$ transfers.
- **Example:** a 5 USDC top-up gives $\phi = 0.05$ and a credit of 4.95 USDC, which covers 495 sponsored transfers.
- Conservation holds exactly: $v = \phi +$ usage $+$ outstanding (`gasRevenueSummary`).

7.3 Treasury

- Shield fees flow to `pool.treasury`.
- Gas protocol and usage fees accrue to the Gas treasury account (`GAS_TREASURY`).
- Both treasuries are designed to be held by a multisig.

8. Gas credit accounting

- **Gas note.** `ghost-gas:v1:σ:κ:amount`, with $\kappa = \text{SHA256}(\sigma)$. The relayer stores only κ. The holder proves ownership by presenting σ.
- **Top-up claim** (`registerGasCreditDeposit`). A claim is credited only if all of these hold:
  - the transfer succeeded and is at most 15 minutes old;
  - it carries an SPL Memo equal to κ;
  - the token delta into the Ghost treasury account equals the claimed amount exactly;
  - its signature has never been claimed before, enforced by a persistent index plus an in-flight reservation.
- **Spend** (`/v1/gas/prepare`, `/v1/gas/submit`):
  - The relayer builds a transaction with itself as fee payer. It contains only compute-budget instructions, an optional idempotent token-account creation, and a `transferChecked` of an allowlisted mint.
  - The user signs it. The relayer re-validates the instruction set (`assertSponsoredTransferTx`), debits c under a lock, co-signs and broadcasts.
  - If broadcast fails, the debit is refunded.
  - Per-sender and per-IP rate limits apply (`GAS_MAX_PER_HOUR`).

9. Relayer economics

Per sponsored transfer, the relayer pays: - the signature fee, 5000 lamports; - a priority fee of $200{,}000$ CU $\times$ 1000 μlamports $= 200$ lamports.

At a SOL price of P, that cost is $5200 \cdot 10^{-9} P$ USDC, about \$0.001 at $P = 200$. The income is $c = 0.01$ USDC per transfer plus the 1% top-up fee ϕ, which gives a positive margin on every standard transfer.

For Shield withdrawals, the 1% fee f funds the relayer's transaction costs and the treasury.

10. Security design

Adversary	How Ghost handles it
Chain observer	Fixed denomination, a shared Merkle anonymity set, proofs generated client-side, and gasless withdrawals to fresh addresses (no linking SOL top-up)
Front-runner	The recipient is a public input of the proof, so redirecting a withdrawal invalidates it

Adversary	How Ghost handles it
Double spender	On-chain nullifier PDAs; canonical public signals
Gas abuser	Memo-bound, exact-amount, time-boxed top-ups; one claim per signature; instruction allowlist; rate limits
Server-side snooping	Ghost Notes and secrets never leave the device. The relayer sees only proofs and public signals
Operator key risk	Authority-only `set_relayer` / `set_fee_bps`, fee cap enforced on-chain, multisig treasury design

11. Architecture

- **Client.** A Next.js app with Reown AppKit / WalletConnect, plus `@ghost/sdk` for notes, Poseidon, Merkle paths and the snarkjs prover.
- **Relayer.** A Fastify service with:
  - Gas endpoints: `/v1/gas/*`;
  - Shield endpoints: `/v1/merkle-path`, `/v1/withdraw`;
  - Groth16 verification, root history, and the Gas credit ledger;
  - Jupiter quotes for multi-asset top-ups.
- **On-chain.** The `ghost_pool` Anchor program holds the `PoolConfig`, the vault, the treasury, and the commitment, leaf and nullifier PDAs, and uses SPL Token.

12. Roadmap

1. On-chain Groth16 verification via Solana's `alt_bn128` syscalls, and an on-chain root history.
2. A multi-party trusted-setup ceremony, then an independent security audit.
3. Multiple denominations and larger trees; a relayer network.
4. Gas-as-a-service SDK for wallets and dApps; rent-aware pricing for new token accounts.
5. Optional compliance: association-set proofs.
6. Token launch and token flywheel (details to follow).

Roadmap items are planned features.