SolSeal Protocol

Encrypted Balance Infrastructure for Solana

Part I: Core Privacy Infrastructure

Part II: Privacy Yield Extension

SolSeal Protocol Team
solseal.xyz | github.com/solseal

January 27, 2026

Abstract

We present SolSeal, a two-part privacy infrastructure protocol for Solana. **Part I** introduces encrypted balance vaults enabling confidential deposits, withdrawals, and transfers using ElGamal homomorphic encryption over the BabyJubJub curve with Groth16 zero-knowledge proofs. Unlike traditional mixers that only obscure transaction linkages while keeping amounts visible, SolSeal maintains permanently encrypted balances with a novel O(1) hint system using ECDH and Poseidon hashing for instant balance discovery. A secret key ownership model decouples vault control from wallet addresses, preventing ownership attribution. Part I is production-deployed on Solana mainnet (Program ID: EbhjvV5oXvrkN5zVjd6GE1NXe8aNaWfgRLmGgaL2r5K3). **Part II** extends this foundation to privacy-preserving DeFi yields, enabling institutions to earn returns from established Solana protocols (Kamino, Marinade, Jupiter) while maintaining encrypted balances. Through a delegation token model and privacy pools, users access yields without exposing position sizes or strategies. This two-part architecture provides both immediate privacy infrastructure (Part I) and institutional-grade privacy yield capabilities (Part II roadmap).

PART I

CORE PRIVACY INFRASTRUCTURE

Encrypted Balance Vaults for Confidential Transactions

Status: Production Deployed on Solana Mainnet

1. INTRODUCTION: THE TRANSPARENCY PROBLEM

Public blockchains expose all account balances and transaction amounts to maintain trustless verification. On Solana, any observer can view wallet holdings, track transfer amounts, and analyze financial activity patterns. While this transparency enables decentralization, it creates fundamental privacy challenges for users managing significant holdings, executing trading strategies, or conducting business operations.

1.1 Why Existing Privacy Solutions Fall Short

Traditional mixers (tumblers) obscure the connection between deposits and withdrawals but suffer critical limitations:

• **Visible amounts:** Deposit 5 SOL → observers see 5 SOL entered. Withdraw 5 SOL → observers see 5 SOL exited. The linkage may be obscured, but amounts are public.

• **No persistent privacy:** Once funds exit the mixer, subsequent transactions occur transparently unless repeatedly mixed.

• **Fixed denominations:** Many mixers require standard amounts (0.1, 1, 10 SOL), leaking information through denomination patterns.

• **Wallet-locked ownership:** Funds are tied to specific wallet addresses, enabling probabilistic attribution analysis.

1.2 SolSeal's Encrypted Balance Approach

SolSeal provides **encrypted balance vaults** rather than transaction mixing. Your balance exists as an ElGamal ciphertext on-chain—a cryptographic scrambling indistinguishable from random data to external observers. Key properties:

• **Hidden deposits:** Deposit amounts immediately encrypted, never visible in plaintext

• **Hidden balances:** Account balances stored as ciphertexts; external observers see encrypted garbage

- **Hidden transfers:** Send encrypted amounts to other vaults without revealing transfer sizes
- **Hidden withdrawals:** Withdrawal amounts remain encrypted throughout execution
- **Secret key ownership:** Vault control independent of wallet addresses; access from any wallet

> Think of SolSeal as a **private bank vault on Solana.** Your funds don't sit as openly visible numbers—they exist as encrypted balances only you can decrypt. Privacy persists throughout the entire lifecycle of your funds.

2. HOW ENCRYPTED BALANCE VAULTS WORK

2.1 Vault Structure and Creation

A SolSeal vault is a Solana program-derived address (PDA) storing encrypted state:

```
struct Vault {
    balance_c1: [u8; 64],        // ElGamal ciphertext component 1
    balance_c2: [u8; 64],        // ElGamal ciphertext component 2
    owner_pubkey: [u8; 32],      // BabyJubJub public key
    nonce: u64,                  // Replay protection counter
    hint_ciphertext: [u8; 32],   // Encrypted hint for O(1) discovery
    bump: u8,                    // PDA bump seed
}
// Total: 161 bytes on-chain storage
```

Creating a vault:

1. User generates **secret key** (256-bit random value, client-side only)

2. Compute public key: $pk = sk \cdot G$ (BabyJubJub curve point)

3. Initialize vault PDA with ~0.0004 SOL rent

4. Store encrypted public key and initial zero balance ciphertext

> **Critical:** The secret key NEVER leaves your device. It's not stored on-chain, not transmitted to servers, and not recoverable if lost. This is true self-custody.

2.2 The Dual Authentication Model

SolSeal implements two-factor authentication for maximum security:

Factor	Purpose	Technology	Loss Impact
Wallet Signature	Authorize transactions	Solana ed25519	Cannot submit TXs
Secret Key (SK)	Generate ZK proofs	BabyJubJub scalar	Cannot decrypt/prove ownership

Table 1: Two-factor authentication model

An attacker gaining access to your Solana wallet CANNOT decrypt balances or prove vault ownership without the secret key. Conversely, possessing the secret key without wallet access cannot submit transactions. Both factors required for fund movement.

2.3 Deposit Operation

Depositing SOL into your encrypted vault:

1. **Client encrypts amount:** Compute *Enc(amount)* using ElGamal with your vault public key

2. **Generate ZK proof:** Prove you know the secret key and encryption is correct (without revealing amount)

3. **Transfer SOL:** Send *amount* SOL from wallet to vault PDA

4. **Update vault:** Store encrypted ciphertext on-chain (balance += amount, but encrypted)

5. **Emit encrypted hint:** Enable O(1) balance discovery for next access

Fee structure: 0.25% deposit fee. Example: Deposit 10 SOL → 9.975 SOL encrypted in vault.

> **Observer view:** Sees transaction to vault PDA, sees encrypted ciphertext update, but CANNOT determine deposit amount.

2.4 Withdrawal Operation

Withdrawing SOL from encrypted vault back to transparent wallet:

1. **Client decrypts balance:** Use secret key to decrypt current balance, verify sufficient funds

2. **Compute new encrypted balance:** Calculate *Enc(balance - withdrawal_amount)*

3. **Generate ZK proof:** Prove balance sufficiency and correct update (takes 5-15 seconds client-side)

4. **Submit transaction:** Transfer *withdrawal_amount* SOL from vault PDA to wallet

5. **Update vault:** Store new encrypted balance on-chain

> **Why withdrawals take longer:** Generating a Groth16 zero-knowledge proof client-side requires 5-15 seconds of computation. This proves balance sufficiency without revealing the amount to validators.

2.5 Transfer Operation (Stealth Payments)

Send encrypted amounts between SolSeal vaults without revealing transfer sizes:

1. **Two-transaction model:** Split into Prepare and Execute due to Solana's 1232-byte transaction size limit

2. **Prepare TX:** Lock funds from sender, generate ZK proof of sender balance

3. **Execute TX:** Deliver encrypted amount to recipient, update both vault balances

4. **Expiration:** If Execute fails, funds automatically return after ~2 minutes

Privacy properties: Neither sender nor recipient amounts visible on-chain. Only encrypted ciphertexts and validity proofs are public. Both parties receive encrypted hints enabling instant balance updates.

3. CRYPTOGRAPHIC FOUNDATIONS

3.1 ElGamal Encryption on BabyJubJub

SolSeal uses ElGamal encryption over the BabyJubJub twisted Edwards curve, optimized for zero-knowledge circuits:

```
Curve Equation: a·x² + y² = 1 + d·x²·y²
Parameters:
  a = 168700
  d = 168696
  Field: Fr of BN254 (~2^254 bits)
  Subgroup Order: l ≈ 2^251

ElGamal Encryption:
  Public Key: pk = sk · G
  Encrypt(value, pk, r):
    C1 = r · G           // 64 bytes
    C2 = value · H + r · pk  // 64 bytes
  Total Ciphertext: 128 bytes

Decryption:
  value · H = C2 - sk · C1
  Solve discrete log to recover value
```

Homomorphic property: ElGamal enables addition on encrypted values:

$$Enc(a) \oplus Enc(b) = Enc(a + b)$$

This allows operations like balance updates (adding deposits, subtracting withdrawals) directly on ciphertexts without decryption.

3.2 Zero-Knowledge Proof System

SolSeal employs Groth16 zkSNARKs to prove transaction validity without revealing amounts:

Circuit	Public Inputs	Purpose
kosk_proof	2 (pk)	Prove knowledge of secret key
deposit_equality_proof	7 (amount, pk, c1, c2)	Prove ciphertext encrypts claimed amount
transfer_sender_proof	6 (pk, c_new)	Prove sender new balance ≥ 0
withdraw_proof_v2	13 (pk, c_old, c_new, amount, nonce)	Prove valid withdrawal with nonce

Table 2: Zero-knowledge proof circuits in production deployment

3.3 The O(1) Hint System

Traditional privacy systems force users to scan the entire blockchain to find their transactions (O(N) complexity). For a balance of 48 bits, this requires ~16 million operations taking 5-120 seconds.

SolSeal's solution: ECDH-encrypted hints

```
// Hint Generation (transaction execution)
shared_secret = randomness · pk_vault
hint = value + Poseidon(shared_secret.x)
emit HintEvent(C1, hint)

// Hint Decryption (client-side, O(1))
shared_secret = sk_vault · C1
value = hint - Poseidon(shared_secret.x)

Time Complexity: O(1) - one scalar multiplication + one hash
Typical Performance: ~50ms per hint verification
```

Security: Without the secret key, hints are computationally indistinguishable from random values under the Computational Diffie-Hellman assumption. Observers learn nothing about vault updates.

3.4 Replay Protection via Nonce Binding

Each vault maintains a nonce counter incremented with every transaction. ZK proofs bind to the current nonce, preventing proof reuse:

• **Nonce binding:** Each proof includes *slot.nonce* as public input

• **Auto-invalidation:** *nonce*++ after successful transaction execution

• **Proof hash check:** On-chain verification ensures nonce matches current state

4. PRODUCTION DEPLOYMENT SPECIFICATIONS

4.1 Deployed System Details

```
Program ID: EbhjvV5oXvrkN5zVjd6GE1NXe8aNaWfgRLmGgaL2r5K3
Network: Solana Mainnet
Status: Production Active

On-Chain Events:
  DepositHintEmitted: 144 bytes
    (slot, c_deposit_c1, value_hint, timestamp)

  WithdrawHintEmitted: 144 bytes
    (slot, c_new_balance_c1, value_hint, timestamp)

  TransferHintEmitted: 176 bytes
    (sender_slot, receiver_slot, c_transfer_c1, value_hint)

  SenderHintEmitted: 144 bytes
    (sender_slot, c_new_balance_c1, sender_value_hint)
```

4.2 Performance Characteristics

Operation	Proof Time	On-Chain Time	Total Latency	Cost
Deposit	~1s	400ms	~1.4s	0.25% + gas
Withdrawal	5-15s	400ms	5.4-15.4s	Gas only
Transfer (Prepare)	3-5s	400ms	3.4-5.4s	Gas only
Transfer (Execute)	N/A	400ms	400ms	Gas only
Hint Verification	N/A	Off-chain	~50ms	Free

Table 3: Production performance metrics

4.3 Practical Constraints and Limits

• **Maximum balance:** ~281,000 SOL (encryption scheme limit)

• **Default decryption range:** Up to 100 SOL (configurable)

• **Minimum deposit:** Technically none, but 0.25% fee makes tiny deposits uneconomical

• **Slot opening cost:** ~0.0004 SOL (refundable rent)

• **Transaction size limit:** 1232 bytes (why transfers require two TXs)

• **BSGS fallback:** 5-120 seconds when hints unavailable (rare)

4.4 User Experience Balance Discovery

Three methods for balance synchronization, used automatically in optimal order:

Method	Speed	When Used
Cached State	Instant	When stored state matches on-chain
Value Hints	~50ms	After deposits, withdrawals, or transfers
BSGS Search	5-120s	When no hints available (rare fallback)

Table 4: Balance discovery methods

5. SECURITY ANALYSIS

5.1 Privacy Guarantees

• **Semantic security (IND-CPA):** ElGamal ciphertexts are computationally indistinguishable from random group elements under the Decisional Diffie-Hellman assumption

• **Transaction unlinkability:** Cannot link deposits to withdrawals except through timing/amount correlation analysis

• **Ownership privacy:** Secret key ownership prevents attribution; same vault accessible from multiple wallets

• **Zero-knowledge soundness:** Cannot forge proofs for invalid transactions under knowledge-of-exponent assumption

5.2 What SolSeal Is NOT

Important clarifications to set correct expectations:

• **NOT a mixer:** Your vault is linked to your Solana wallet at creation. Transaction history is visible (not amounts).

• **NOT a tumbler:** You cannot mix funds with others; each vault is independent.

• **NOT complete anonymity:** Sophisticated analysis may correlate deposits/withdrawals via timing or behavioral patterns.

• **NOT regulatory evasion:** Designed for legitimate privacy, not illicit concealment. Viewing key extensions planned.

5.3 Attack Resistance

• **Front-running attacks:** Encrypted amounts prevent MEV bots from extracting value based on transaction sizes

• **Inflation attacks:** ZK proofs ensure balance updates conserve total supply; no unauthorized minting

• **Replay attacks:** Nonce binding prevents proof reuse; each proof valid for exactly one nonce value

• **Correlation attacks:** While timing analysis possible, amount privacy prevents direct linkage of specific deposits to withdrawals

PART II

PRIVACY YIELD EXTENSION

Earning DeFi Yields with Encrypted Balances

Status: Roadmap / Technical Design

6. EXTENDING ENCRYPTED VAULTS TO EARN YIELDS

6.1 The Institutional DeFi Opportunity

Part I provides encrypted balance infrastructure for confidential transactions. Part II extends this foundation to enable institutions to earn DeFi yields while maintaining position privacy. The market opportunity is substantial:

• $130+ billion in DeFi lending protocols (Aave, Morpho, Compound)

• $306 billion stablecoin market cap, <5% in DeFi yields

• Corporate treasuries want 8-15% vs. 2-3% TradFi, blocked by transparency

• Institutions accept 2-5x transaction costs for compliant privacy

6.2 The Encrypted-Balance-Yield Problem

Part I's encrypted vaults create a technical challenge: **yield protocols need to see balances to calculate interest.**

Traditional approach: User deposits 1000 SOL → Protocol sees 1000 → Calculates 8% → Returns 1080 SOL.

Encrypted approach problem: User deposits *Enc(1000)* → Protocol sees ciphertext (gibberish) → Cannot calculate interest.

> **Solution:** Privacy pools with delegation tokens (adapted from Penumbra).

6.3 Privacy Pool Architecture

Privacy pools aggregate encrypted deposits and deploy capital to yield sources as single large institutional positions:

```
User Flow:
1. Alice deposits Enc(100 SOL) to privacy pool
```

```
2. Bob deposits Enc(200 SOL) to privacy pool
3. Carol deposits Enc(700 SOL) to privacy pool

Privacy Pool State:
  Total: 1000 SOL (pool knows plaintext aggregate)
  Alice owns: Enc(10% shares)
  Bob owns: Enc(20% shares)
  Carol owns: Enc(70% shares)

Pool → Kamino Lending:
  Single deposit: 1000 SOL
  Kamino sees: One institutional depositor
  Kamino does NOT see: Individual users or amounts

Yield Accrual (8% APY):
  Kamino returns: 1080 SOL to pool
  Alice's shares now worth: 10% of 1080 = 108 SOL
  Bob's shares now worth: 20% of 1080 = 216 SOL
  Carol's shares now worth: 70% of 1080 = 756 SOL

Shares remain encrypted; only redemption values visible to users.
```

6.4 Why This Works

• **For privacy:** Individual deposits encrypted, share ownership encrypted, withdrawal amounts encrypted

• **For yield calculation:** Kamino operates on pool total (1000 SOL), no encryption complexity on their end

• **For composability:** Works with existing Solana protocols without modifications

• **For efficiency:** Single large position vs. many encrypted micro-positions

7. SOLANA YIELD SOURCE INTEGRATION

7.1 Target Integration Partners

Privacy pools will integrate with established Solana yield protocols:

Protocol	TVL	Category	Target APY	Integration Status
Kamino Finance	$3.7B	Lending	8-12%	Priority integration
Marinade Finance	$2.2B	Liquid staking	6-8%	Priority integration
Jito	$1.87B	Liquid staking	7-9%	Under evaluation
Jupiter Lend	$1.65B	Lending	10-15%	Under evaluation
Orca	$500M+	DEX LP	12-18%	Future consideration

Table 5: Target Solana yield protocol integrations

7.2 Integration Requirements

For a yield protocol to be compatible with SolSeal privacy pools:

✓ Accept standard SOL/USDC deposits

✓ Return deposits + yield to same address

✓ No permanent lockups preventing withdrawals

✓ Transparent yield calculation (even if balances aren't)

✓ Support for large institutional-sized positions

> **Critically:** Yield protocols require ZERO modifications. Privacy pools interact as normal large depositors.

7.3 Homomorphic Yield Application

Leveraging ElGamal's additive homomorphism, interest can be applied directly to encrypted balances:

```
// Pool operator (or smart contract) applies yield
current_pool_total = 1000 SOL
yield_earned = 80 SOL (8% APY)
new_pool_total = 1080 SOL

// User's encrypted shares don't change
// But redemption value increases proportionally
Alice_shares = Enc(10%)
Redemption_value = 10% × 1080 = 108 SOL

// No decryption needed for yield accrual
// Shares appreciate automatically with pool value
```

8. INSTITUTIONAL COMPLIANCE FEATURES

8.1 Viewing Keys for Selective Disclosure

Institutional adoption requires regulatory auditability without market exposure. Privacy yield extends Part I's architecture with viewing keys:

```
struct InstitutionalVault {
    // User ciphertext (only user can decrypt)
    user_ciphertext: ElGamalCiphertext,

    // Auditor ciphertext (same balance, auditor key)
    auditor_ciphertext: ElGamalCiphertext,

    // Compliance proof (screened via oracle)
    compliance_proof: Groth16Proof,

    // Yield commitment (for solvency proofs)
    yield_commitment: PedersenCommitment,
}
```

• **Read-only access:** Auditors decrypt balances but cannot move funds

• **Threshold decryption:** 5-of-9 committee required for involuntary access

• **Selective disclosure:** User controls which auditors receive viewing access

• **Regulatory compliance:** Satisfies GENIUS Act requirement for 'technical capability' to provide regulatory access

8.2 Compliance Oracle Integration

Pre-transaction screening prevents illicit fund entry while maintaining depositor privacy:

1. User requests screening attestation from oracle (Chainalysis/TRM)

2. Oracle verifies address against AML/OFAC sanctions lists

3. If clean, oracle signs attestation

4. User generates ZK proof: 'I belong to approved set' without revealing identity

5. Smart contract verifies proof before accepting deposit

> **Result:** Protocol demonstrates regulatory screening without maintaining permanent user databases. Privacy from market, transparency to regulators.

8.3 Regulatory Framework Compliance

• **GENIUS Act (July 2025):** Federal stablecoin framework; compliance via viewing keys and screening

• **CLARITY Act (July 2025):** DeFi safe harbor provisions; privacy pools operate as automated protocols

• **Tornado Cash precedent (March 2025):** Immutable code unsanctionable; compliance built in from day one avoids liability

• **SEC endorsement (Dec 2025):** Chair Atkins explicitly supports 'zero-knowledge proofs and selective disclosure systems'

9. IMPLEMENTATION ROADMAP

9.1 Technical Development Phases

Phase 1: Viewing Key Integration (2-3 months)

• Multi-recipient encryption for vault balances

• Threshold decryption committee (5-of-9 multisig)

• Viewing key rotation mechanisms

• Testnet deployment and testing

Phase 2: Compliance Infrastructure (1-2 months)

• Chainalysis/TRM oracle API integration

• Privacy Pools-style association sets

• Pre-deposit screening smart contracts

• Regulatory reporting dashboard

Phase 3: Yield Pool Implementation (2-3 months)

• Delegation token smart contracts

• Kamino/Marinade integration wrappers

• Homomorphic interest calculation

• Solvency proof systems (Merkle sum trees)

Phase 4: Security & Launch (2-3 months)

• Trail of Bits security audit

• Zellic Solana-specific audit

• Bug bounty program ($500K pool)

• Mainnet deployment with institutional pilots

9.2 Success Metrics

• **TVL target:** $100M+ within 12 months (sufficient anonymity set)

• **Institutional clients:** 10+ funds/treasuries using privacy yield

• **Yield competitiveness:** Within 50bps of transparent alternatives

• **Zero enforcement actions:** Proactive regulatory compliance

- **Transaction privacy:** >95% of volume in shielded pools

10. CONCLUSION

SolSeal presents a two-part privacy infrastructure for Solana addressing immediate confidentiality needs and enabling future institutional DeFi adoption.

10.1 Part I: Proven Production Infrastructure

The core encrypted balance system is production-deployed on Solana mainnet (Program ID: EbhjvV5oXvrkN5zVjd6GE1NXe8aNaWfgRLmGgaL2r5K3), demonstrating that encrypted balance vaults can operate at scale. Key achievements:

• Permanently encrypted balances using ElGamal over BabyJubJub

• O(1) balance discovery via ECDH-encrypted hints (~50ms typical)

• Secret key ownership preventing wallet-based attribution

• Production performance: ~240,000 compute units, sub-second finality

• Zero-knowledge soundness via Groth16 proofs

10.2 Part II: Institutional Privacy Yield Pathway

The privacy yield extension leverages Part I's proven infrastructure to enable institutional DeFi participation. The technical approach is sound, regulatory framework is clarified, and market demand is documented. Implementation phases target 8-10 month development cycle with institutional pilots beginning in Phase 4.

10.3 The Convergence Moment

Three factors create a unique opportunity for institutional privacy infrastructure:

• **Regulatory clarity:** GENIUS Act, CLARITY Act, and SEC endorsement establish frameworks for compliant privacy

• **Proven technology:** Part I demonstrates encrypted balances work in production; Part II extends existing foundation

• **Market demand:** $306B stablecoins seeking yields, institutional DeFi adoption accelerating, privacy premium documented

SolSeal provides both immediate privacy infrastructure (Part I, available now) and institutional-grade privacy yield capabilities (Part II, roadmap). The two-part architecture enables iterative development: prove the core technology works, then extend to institutional use cases with viewing keys and compliance integration.

The code, documentation, and technical specifications are available at **solseal.xyz** and **github.com/solseal**.

REFERENCES

[1] Baylina, J. and Bellés-Muñoz, M. 'Baby Jubjub Elliptic Curve.' Iden3 Documentation, 2018.

[2] Groth, J. 'On the Size of Pairing-Based Non-interactive Arguments.' EUROCRYPT 2016.

[3] Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., and Schofnegger, M. 'Poseidon: A New Hash Function for Zero-Knowledge Proof Systems.' USENIX Security 2021.

[4] U.S. Congress. 'Guiding and Establishing National Innovation for U.S. Stablecoins (GENIUS) Act.' July 18, 2025.

[5] U.S. Congress. 'Clarity for Payment Stablecoins Act of 2025.' Passed House 294-134, July 17, 2025.

[6] SEC Chair Paul Atkins. 'Remarks at Crypto Task Force Roundtable on Financial Surveillance and Privacy.' SEC.gov, December 15, 2025.

[7] DeFi Llama. 'DeFi TVL Rankings.' DefiLlama.com, accessed January 2026.

[8] CoinGecko. '2025 Annual Cryptocurrency Report.' Coingecko.com, December 2025.

[9] Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. 'Zerocash: Decentralized Anonymous Payments from Bitcoin.' IEEE S&P; 2014.

[10] Storm, R., Semenov, R., and Pertsev, A. 'Tornado Cash.' Tornado.cash, 2019.

[11] Hopwood, D., Bowe, S., Hornby, T., and Wilcox, N. 'Zcash Protocol Specification, Version 2020.1.15 [Sapling].' Electric Coin Company, 2020.

[12] a16z Crypto. 'Privacy-Protecting Regulatory Solutions Using Zero-Knowledge Proofs.' A16zcrypto.com, 2024.