



SECURITY ASSESSMENT

Provided by Accretion Labs Pte Ltd. for Privacy Cash
December 08, 2025
A25PRC2



Privacy Cash

AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Solana Auditor	Mahdi Rostami (mahdi@accretion.xyz)
Solana Auditor	David Ratiney (david@accretion.xyz)

CLIENT

Privacy Cash (<https://www.privacypcash.org/>) is developing a protocol that enables private token transfers on Solana through zero-knowledge proofs. They have commissioned Accretion to perform a security assessment of their new SPL token transfer feature.

ENGAGEMENT TIMELINE



AUDITED CODE

Program 1

ProgramID: 9fhQBbumKEFuXtMBDw8AaQyAjCorLGJQIS3skWZdQyQD
Repository: <https://github.com/Privacy-Cash/privacy-cash>

ASSESSMENT

The security assessment of Privacy Cash's new token transfer feature identified a simple, straightforward implementation following the existing layout for transferring native Sol. Our analysis uncovered only low- or info-severity vulnerabilities, except for one high severity issue that may allow users to steal rent fees from the protocol's relayer. The Privacy Cash team has successfully remediated all identified vulnerabilities.

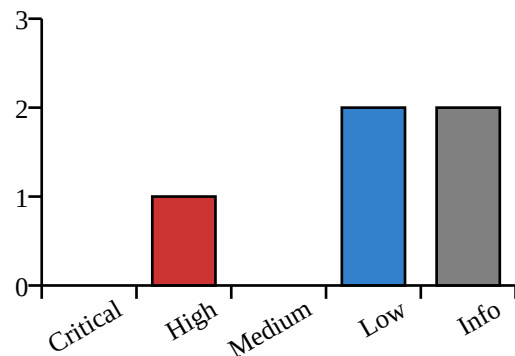
CODE ASSESSMENT

The Privacy Cash SPL token transfer code demonstrated clean architecture and simplicity written in Anchor, following the proven design of the native transfer feature. The current implementation maintains a minimal code structure that effectively aligns with its intended functionality.

KEY FINDINGS

We identified one high-severity vulnerability during the review. It allowed an attacker to request token transfers and the relayer subsequently paying for token account creation costs for those transfers. The remaining issues related mostly to the usage of the associated token program. All issues have been remediated or acknowledged by the team.

SEVERITY DISTRIBUTION



ENGAGEMENT SCOPE

The scope of this security assessment was a full review of the following items:

Item 1: Privacy Cash

Link: <https://github.com/Privacy-Cash/privacy-cash>

Commit: 34a6101a412223eb279a6964ec97cbc59bea4f14

Program ID: 9fhQBbumKEFuXtMBDw8AaQyAjCorLGJQiS3skWZdQyQD

Audit Result:

- **Audited Commit:** b62c1e733cfc43616f945de86be7b415c0ca2def
- **Comment:** Final commit including all fixes

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-H1	Users Can Rent-Grief SPL Merkle Trees and Drain Relayer Funds	HIGH	WONTFIX
ACC-L1	Pre-Initialized ATA Can Cause DoS in <code>`initialize_tree_account_for_spl_token`</code>	LOW	FIXED
ACC-L2	<code>`transactSPL`</code> Forces Use of ATA and Blocks Non-ATA Token Accounts	LOW	FIXED
ACC-I1	Logs May Be Truncated	INFO	WONTFIX
ACC-I2	TransactSPL requires recipient token account on deposits	INFO	WONTFIX

DETAILED ISSUES

ACC-H1 Users Can Rent-Grief SPL Merkle Trees and Drain Relay Funds

HIGH

WONTFIX

Description

We found that users can perform a **rent grieving attack** on SPL Merkle trees.

The issue comes from the withdrawal flow using "init if needed" for the user token account. This means:

1. A user can repeatedly **deposit 1 token** (with **0 fees**). 2. Close ATA 3. Then **withdraw** (also with **0 fees**). 4. Each cycle creates ATA that cost **rent**, which the **relayer** ends up paying for. By repeating this many times, a malicious user can slowly **drain the relayer's funds** through rent costs, even without paying any fees or taking on any risk.

Location

<https://github.com/Privacy-Cash/privacy-cash/blob/7e3fc1764d00568cb20405f65dd29540c8a38073/anchor/programs/zkcash/src/lib.rs#L716-L724>

Relevant Code

```
/// Recipient's token account (destination for withdrawals)
/// Created automatically if it doesn't exist
#[account(
    init_if_needed,
    payer = signer,
    associated_token::mint = mint,
    associated_token::authority = recipient
)]
pub recipient_token_account: Account<'info, TokenAccount>,
```

Mitigation Suggestion

Add extra fee for SPL merkle trees.

Remediation

Issue has been marked as won't fix as there will be a charge by the relayer higher than the rent cost, and the relayer will only have a limited amount of SOL in it to limit exploitability.

ACC-L1

Pre-Initialized ATA Can Cause DoS in `initialize\_tree\_account\_for\_spl\_token`

LOW

FIXED

Description

In `initialize_tree_account_for_spl_token`, the tree's associated token account (ATA) is created with `init`.

Because this uses `init`, the account **must not exist** before the instruction runs. An attacker can create this ATA *before* the protocol does (same `mint` and `global_config`), so later calls to `initialize_tree_account_for_spl_token` will always fail.

This lets anyone **cause a denial of service (DoS)** by pre-initializing the ATA and blocking honest users or the protocol from running this instruction.

Location

-

Relevant Code

```
/// Tree's associated token account for this mint
#[account(
    init,
    payer = authority,
    associated_token::mint = mint,
    associated_token::authority = global_config
)]
pub tree_ata: Account<'info, TokenAccount>,
```

Mitigation Suggestion

Replace `init` with `init_if_needed`.

Remediation

Fixed in <https://github.com/Privacy-Cash/privacy-cash/pull/5/commits/ef16f2b9690bafa36413ec8e3ef6180ebdb8b19e>.

Description

We found that the current **transactSPL** implementation **forces users to use their Associated Token Account (ATA)** for all SPL token interactions. This creates a limitation:

- Users **cannot** use their **non-ATA token accounts**, even if those accounts are valid, owned by them, and properly initialized.

This reduces flexibility and breaks use cases where users intentionally manage multiple token accounts, custodial accounts, program-owned accounts, or advanced wallet setups.

Location

<https://github.com/Privacy-Cash/privacy-cash/blob/7e3fc1764d00568cb20405f65dd29540c8a38073/anchor/programs/zkcash/src/lib.rs#L731-L732>

Relevant Code

```
associated_token::mint = mint,  
associated_token::authority = global_config
```

Mitigation Suggestion

Allow **transactSPL** to accept **any valid user token account**, not only the ATA.

Remediation

Fixed in <https://github.com/Privacy-Cash/privacy-cash/commit/7a0b7cf099f0bcac119b4cdc4a923ba51d097806>.

Description

We found that the program uses the anchor `emit!` macro to produce logs, which internally uses the solana log syscall. However, this is vulnerable to being truncated after reaching 10kb of logs within the transaction. If the product heavily relies on off-chain parsing of emitted logs, this can lead to issues in indexing and off chain interpretation of on-chain data.

Location

<https://github.com/Privacy-Cash/privacy-cash/blob/7e3fc1764d00568cb20405f65dd29540c8a38073/anchor/programs/zkcash/src/lib.rs#L489-L501>

Relevant Code

```
/// lib.rs L489-L501
emit!(SplCommitmentData {
    index: next_index_to_insert,
    mint_address: ext_data.mint_address,
    commitment: proof.output_commitments[0],
    encrypted_output: encrypted_output1.to_vec(),
});

emit!(SplCommitmentData {
    index: second_index,
    mint_address: ext_data.mint_address,
    commitment: proof.output_commitments[1],
    encrypted_output: encrypted_output2.to_vec(),
});
```

Mitigation Suggestion

Use a more reliable logging approach like CPI logging.

Remediation

Marked as won't fix.

Description

We found that the TransactSPL instruction always requires a recipient, and a recipient token account, which is marked with `init_if_needed`. However, a recipient is only really needed for withdrawals, while on deposits it will get ignored. This may become an issue when the protocol enables all token mints. In such a case, on deposit, the depositor will need to `init_if_needed` a throwaway token account, if they create a deposit for a new token, wasting user funds.

Location

<https://github.com/Privacy-Cash/privacy-cash/blob/7e3fc1764d00568cb20405f65dd29540c8a38073/anchor/programs/zkcash/src/lib.rs#L718-L724>

Relevant Code

```
/// lib.rs L718-L724
#[account(
    init_if_needed,
    payer = signer,
    associated_token::mint = mint,
    associated_token::authority = recipient
)]
pub recipient_token_account: Account<'info, TokenAccount>
```

Mitigation Suggestion

We recommend making the `recipient_token_account` an `Option<>` account, so that on deposits, it doesn't have to exist or be created. The `recipient` account is still used for the proof on deposits and should remain as-is. Another approach would be to combine the `signer_token_account` and the `recipient_token_account` into a single account, depending on the action being a deposit or withdrawal. An even better solution may be to split deposit and withdrawal into separate instructions with their own interfaces, that may still share the core logic implementation.

Remediation

Marked as won't fix, as the protocol will only has plans to allow USDC and USDT.

APPENDIX

Vulnerability Classification

We rate our issues according to the following scale. Informational issues are reported informally to the developers and are not included in this report.

Severity	Description
Critical	Vulnerabilities that can be easily exploited and result in loss of user funds, or directly violate the protocol's integrity. Immediate action is required.
High	Vulnerabilities that can lead to loss of user funds under non-trivial preconditions, loss of fees, or permanent denial of service that requires a program upgrade. These issues require attention and should be resolved in the short term.
Medium	Vulnerabilities that may be more difficult to exploit but could still lead to some compromise of the system's functionality. For example, partial denial of service attacks, or such attacks that do not require a program upgrade to resolve, but may require manual intervention. These issues should be addressed as part of the normal development cycle.
Low	Vulnerabilities that have a minimal impact on the system's operations and can be fixed over time. These issues may include inconsistencies in state, or require such high capital investments that they are not exploitable profitably.
Informational	Findings that do not pose an immediate risk but could affect the system's efficiency, maintainability, or best practices.

Audit Methodology

Accretion is a boutique security auditor specializing in Solana's ecosystem. We employ a customized approach for each client, strategically allocating our resources to maximize code review effectiveness. Our auditors dedicate substantial time to developing a comprehensive understanding of each program under review, examining design decisions, expected and edge-case behaviors, invariants, optimizations, and data structures, while meticulously verifying mathematical correctness—all within the context of the developers' intentions.

Our audit scope extends beyond on-chain components to include associated infrastructure, such as user interfaces and supporting systems. Every audit encompasses both a holistic protocol design review and detailed line-by-line code analysis.

During our assessment, we focus on identifying:

- Solana-specific vulnerabilities
- Access control issues
- Arithmetic errors and precision loss
- Race conditions and MEV opportunities
- Logic errors and edge cases
- Performance optimization opportunities
- Invariant violations
- Account confusion vulnerabilities
- Authority check omissions
- Token22 implementation risks and SPL-related pitfalls
- Deviations from best practices

Our approach transcends conventional vulnerability classifications. We continuously conduct ecosystem-wide security research to identify and mitigate emerging threat vectors, ensuring our audits remain at the forefront of Solana security practices.