SECURITY ASSESSMENT

Accretion

Provided by Accretion Labs Pte Ltd. for Privacy Cash
August 18, 2025
A25PRC1

Privacy Cash

AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Solana Auditor	Mahdi Rostami (mahdi@accretion.xyz)
Solana Auditor	Niklas Brymko (niklas@accretion.xyz)

CLIENT

Privacy Cash (https://www.privacycash.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 Privacy Cash, their implementation of this protocol.

ENGAGEMENT TIMELINE

27 Jun — Project Kickoff
Initial planning and scope definition

27 Jun — Assessment Begins
Security review and testing phase

15 Jul — Review Fixes
Security recommendations are given and implemented

16 Aug — Project Completion
Project completion and report delivery

AUDITED CODE

Program 1

ProgramID: 9fhQBbumKEFuXtMBDw8AaQyAjCorLGJQiS3skWZdQyQD

Repository: https://github.com/Privacy-Cash/privacy-cash

ASSESSMENT

The security assessment of Privacy Cash identified a clean, straightforward codebase incorporating components from Light Protocol and Tornado Cash. The contract features minimal functions for protocol configuration and a primary transfer function. Our analysis uncovered several vulnerabilities, including two critical-severity issues. The Privacy Cash team has successfully remediated all identified vulnerabilities.

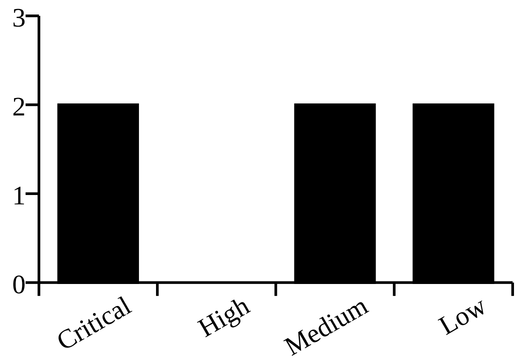
CODE ASSESSMENT

The Privacy Cash contract code demonstrated clean architecture and simplicity, incorporating critical components from previously audited programs. During the assessment, we identified several issues that led to further contract simplification. The current implementation maintains a minimal code structure that effectively aligns with its intended functionality.

KEY FINDINGS

We identified two critical-severity vulnerabilities during the review. The first vulnerability enabled frontrunning attacks and fund theft through an absent account validation. The second vulnerability allowed merkle tree creators to execute denial-of-service attacks against the program, resulting in locked user funds within the merkle tree.

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: e6070e6418a723dc8ee141de75e10580d1c7ebbb
Program ID: 9fhQBbumKEFuXtMBDw8AaQyAjCorLGJQiS3skWZdQyQD
Audit Result:
- **Audited Commit:** 188f0cd475397c4039bf126f7962282170395aad
- **Comment:** Final commit including all fixes, verified on-chain

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-C1	Tree authority can lock funds in the tree	critical	fixed
ACC-C2	Missing Recipient Validation in Withdraw Allows Front-Running and Fund Theft	critical	fixed
ACC-M1	Unnecessary Restriction in Transact	medium	fixed
ACC-M2	Missing Validation on Fee Recipient Enables Front-Running to Steal Fees	medium	fixed
ACC-L1	Initialize Vulnerable to Frontrunning	low	fixed
ACC-L2	Not checking if `tree_account` is full or not	low	fixed
ACC-I1	No need for Authority in Transact	info	fixed
ACC-I2	Unrestricted Merkle Tree Initialization Enables Abuse	info	fixed

DETAILED ISSUES

ID	ACC-C1
Title	Tree authority can lock funds in the tree
Severity	critical
Status	fixed

Description

We found that the authority of a tree can prevent users for transacting with it at any point, potentially holding user funds hostage. This is because when users want to interact with the program, they use the `Transact` instruction. It takes the `authority` account as an input with a constraint that this account is a `SystemAccount`. This means, when the `authority` account stops being a `SystemAccount`, no one can call the `Transact` instruction. In practice, this means that a malicious authority can create a tree, wait for any users to deposit funds, and then at some point change their own `authority` account owner to any other program, maybe in a way that they can revert this action. Now all deposited funds are trapped in the program until this authority changes the account owner back to `System`.

Location

https://github.com/SocialfiPanda/privacy-cash/blob/61120f943dbd1b44ecbb522e791f3632f72a26ff/anchor/programs/zkcash/src/lib.rs#L280-L280

Relevant Code

```
/// lib.rs L280-L280
    pub authority: SystemAccount<'info>,
```

Mitigation Suggestion

Change the constraint for `authority` in Transact from `SystemAccount` to `AccountInfo`, which allows it to belong to any program.

Remediation

Fixed in commit `606942c4ec77d680f0cf8164834c80cba6fbbfeb`.

ID	ACC-C2
Title	Missing Recipient Validation in Withdraw Allows Front-Running and Fund Theft
Severity	critical
Status	fixed

Description

We found that during the withdraw process, the recipient account is not validated against `ExtData::recipient`. This allows an attacker to front-run a legitimate transaction and replace the recipient with their own address to steal funds.

• **Test:**

```
@@ -75,6 +75,7 @@ describe("zkcash", () => {
   let recipient: anchor.web3.Keypair;
   let fundingAccount: anchor.web3.Keypair;
   let randomUser: anchor.web3.Keypair; // Random user for signing transactions
+  let attacker: anchor.web3.Keypair; // Random user for signing transactions

   // Initialize variables for tree token account
   let treeTokenAccountPDA: PublicKey;

@@ -186,10 +187,10 @@ describe("zkcash", () => {
         lastValidBlockHeight: latestBlockHash2.lastValidBlockHeight,
         signature: treeTokenAirdropSignature,
       });
       // Generate a random user for signing transactions
       randomUser = anchor.web3.Keypair.generate();
+     attacker = anchor.web3.Keypair.generate();
       // Fund the random user with SOL
       const randomUserAirdropSignature = await provider.connection.requestAirdrop(randomUser.publicKey, 1 *
LAMPORTS_PER_SOL);
       const latestBlockHash4 = await provider.connection.getLatestBlockhash();

@@ -198,7 +199,14 @@ describe("zkcash", () => {
         lastValidBlockHeight: latestBlockHash4.lastValidBlockHeight,
         signature: randomUserAirdropSignature,
       });

+      const attackerAirdropSignature = await provider.connection.requestAirdrop(attacker.publicKey, 1 * LAMP
ORTS_PER_SOL);
+      await provider.connection.confirmTransaction({
+        blockhash: latestBlockHash4.blockhash,
+        lastValidBlockHeight: latestBlockHash4.lastValidBlockHeight,
+        signature: attackerAirdropSignature,
+      });
```

change the `Can execute both deposit and withdraw instruction for correct input, with 0 fee` test with following:

```
    // Create Address Lookup Table for transaction size optimization
    const testProtocolAddresses = getTestProtocolAddresses(
      program.programId,
      authority.publicKey,
      treeAccountPDA,
      treeTokenAccountPDA,
      nullifier0PDA,
      nullifier1PDA,
      commitment0PDA,
      commitment1PDA,
      recipient.publicKey,
      feeRecipient.publicKey,
```



```
      randomUser.publicKey,
    );
+    testProtocolAddresses.push(attacker.publicKey); // Add attacker to the lookup table

    const lookupTableAddress = await createGlobalTestALT(provider.connection, authority, testProtocolAddresse
s);
   .
   .
   .
    // Get balances after transaction
    const treeTokenAccountBalanceAfter = await provider.connection.getBalance(treeTokenAccountPDA);
    const feeRecipientBalanceAfter = await provider.connection.getBalance(feeRecipient.publicKey);
    const recipientBalanceAfter = await provider.connection.getBalance(recipient.publicKey);
    const randomUserBalanceAfter = await provider.connection.getBalance(randomUser.publicKey);
+    const attackerBalanceBefore = await provider.connection.getBalance(attacker.publicKey);
   .
   .
   .
    // Execute the withdrawal transaction
    const withdrawTx = await program.methods
      .transact(withdrawProofToSubmit, withdrawExtData)
      .accounts({
        treeAccount: treeAccountPDA,
        nullifier0: withdrawNullifiers.nullifier0PDA,
        nullifier1: withdrawNullifiers.nullifier1PDA,
        commitment0: withdrawCommitments.commitment0PDA,
        commitment1: withdrawCommitments.commitment1PDA,
+       recipient: attacker.publicKey,
        feeRecipientAccount: feeRecipient.publicKey,
        treeTokenAccount: treeTokenAccountPDA,
        authority: authority.publicKey,
        signer: randomUser.publicKey,
        systemProgram: anchor.web3.SystemProgram.programId
      })
      .signers([randomUser])
      .preInstructions([modifyComputeUnits])
      .transaction();
   .
   .
   .
    // Get final balances after both transactions
    const finalTreeTokenBalance = await provider.connection.getBalance(treeTokenAccountPDA);
    const finalFeeRecipientBalance = await provider.connection.getBalance(feeRecipient.publicKey);
    const finalRandomUserBalance = await provider.connection.getBalance(randomUser.publicKey);
+    const finalAttackerBalance = await provider.connection.getBalance(attacker.publicKey);
    // Calculate the withdrawal diffs specifically
    const treeTokenWithdrawDiff = finalTreeTokenBalance - treeTokenAccountBalanceAfter;
    const feeRecipientWithdrawDiff = finalFeeRecipientBalance - feeRecipientBalanceAfter;
    const randomUserWithdrawDiff = finalRandomUserBalance - randomUserBalanceAfter;
+    const attackerWithdrawDiff = finalAttackerBalance - attackerBalanceBefore;
    // Verify withdrawal logic worked correctly
    expect(treeTokenWithdrawDiff).to.be.equals(extAmount.toNumber() - withdrawFee.toNumber()); // Tree decrea
ses by withdraw amount
    expect(feeRecipientWithdrawDiff).to.be.equals(withdrawFee.toNumber()); // Fee recipient unchanged
    expect(randomUserWithdrawDiff).to.be.lessThan(-extAmount.toNumber()); // User gets withdraw amount minus
tx fee
+    // Attacker gets the full amount
+    expect(attackerWithdrawDiff).to.be.equals(-extAmount.toNumber());
    // Calculate overall diffs for the full cycle
    const treeTokenTotalDiff = finalTreeTokenBalance - treeTokenAccountBalanceBefore;
    const feeRecipientTotalDiff = finalFeeRecipientBalance - feeRecipientBalanceBefore;
    const randomUserTotalDiff = finalRandomUserBalance - randomUserBalanceBefore;
```

Location

https://github.com/SocialfiPanda/privacy-cash/blob/e6070e6418a723dc8ee141de75e10580d1c7ebbb/anchor/programs/zkcash/src/lib.rs#L273-L274

Relevant Code


```
#[account(mut)]
    pub recipient: SystemAccount<'info>,
```

Mitigation Suggestion

Ensure the recipient account in the transaction matches the `ExtData::recipient` value before processing the withdrawal.

Remediation

Fixed in commit 62c47f5ba56be309cffede2d4617345c7c1fd94a.

ID	ACC-M1
Title	Unnecessary Restriction in Transact
Severity	medium
Status	fixed

Description

In Transact, we discovered that the `systemAccount` is used for both the `recipient` and the `fee_recipient_account`. This creates an unnecessary restriction, as users are unable to deposit or withdraw funds from a PDA or any other account that is not a `systemAccount`. Additionally, if the `fee_recipient_account` is a PDA, users are unable to deposit or withdraw funds.

Location

https://github.com/SocialfiPanda/privacy-cash/blob/dc6516a1bf79aabb8773831af61c5f27c95c9029/anchor/programs/zkcash/src/lib.rs#L354-L358

Relevant Code

```
#[account(mut)]
    pub recipient: SystemAccount<'info>,

    #[account(mut)]
    pub fee_recipient_account: SystemAccount<'info>,
```

Mitigation Suggestion

Use `UncheckedAccount` instead of `SystemAccount`.

Remediation

Fixed in commit 9c07fa1726058864e5aef025d0c666aa5c7e2559.

ID	ACC-M2
Title	Missing Validation on Fee Recipient Enables Front-Running to Steal Fees
Severity	medium
Status	fixed

Description

During the withdrawal process, the contract is intended to send the fee to a `fee_recipient_account` controlled by the authority. However, there is no validation ensuring that this account is legitimate. As a result, an attacker can front-run a withdrawal and replace the `fee_recipient_account` with their own address to steal the fee.

Location

https://github.com/SocialfiPanda/privacy-cash/blob/bd117d4641292fc893e529068c922a3fab1f57fd/anchor/programs/zkcash/src/lib.rs#L276-L279

Relevant Code

```
#[account(mut)]
    pub fee_recipient_account: SystemAccount<'info>,

    /// The authority account is the account that created the tree and fee recipient PDAs
```

Mitigation Suggestion

Add a `fee_recipient` field to the `MerkleTreeAccount` and validate the `fee_recipient_account` during the withdrawal process.

```
#[account(zero_copy)]
pub struct MerkleTreeAccount {
    pub authority: Pubkey,
.
.
.
    pub bump: u8,
+   pub fee_recipient: Pubkey,
    pub _padding: [u8; 7],
}
```

Remediation

Fixed in commit 63447e9baa9b1869a787de90ebdfe26b8aac7f3b.

ID	ACC-L1
Title	Initialize Vulnerable to Frontrunning
Severity	low
Status	fixed

Description

The initialize function can only be called once, and an attacker could front run this call. If Initialize has not been invoked when the contract is deployed, an attacker can preemptively invoke it through a front-running attack. This would set the authority address to the attacker's address, forcing the contract to be redeployed.

Location

https://github.com/SocialfiPanda/privacy-cash/blob/476618cc9af3b1d877c277d47bd3ce95e2f9e2ab/anchor/programs/zkcash/src/lib.rs#L25

Mitigation Suggestion

Hardcode the authority address.

Remediation

Fixed in commit c7d7dc4c40101074203bd93c006a5dbcc08a2c0f.

ID	ACC-L2
Title	Not checking if `tree_account` is full or not
Severity	low
Status	fixed

Description

We found that in `MerkleTree::append` it doesn't check if `tree_account` is full or not. In a full tree, these updates overwrite valid intermediate nodes with incorrect hashes, as the tree's structure assumes no more leaves can be added. This disrupts the tree's integrity, as the subtrees no longer accurately represent the tree's state.

Location

https://github.com/SocialfiPanda/privacy-cash/blob/e6070e6418a723dc8ee141de75e10580d1c7ebbb/anchor/programs/zkcash/src/merkle_tree.rs#L29

Relevant Code

```
let mut current_index = tree_account.next_index as usize;
```

Mitigation Suggestion

Add a check to ensure that it's less than `2**DEFAULT_HEIGHT`.

Remediation

Fixed in commit 1acfa3724ecfa2b43e7bdde03d63763b29961bb8.

ID	ACC-I1
Title	No need for Authority in Transact
Severity	info
Status	fixed

Description

We determined that since there is only one Merkle tree account, there is no need to check authority in that account during the transaction process, allowing us to eliminate the authority requirement.

Location

https://github.com/SocialfiPanda/privacy-cash/blob/476618cc9af3b1d877c277d47bd3ce95e2f9e2ab/anchor/programs/zkcash/src/lib.rs#L358

Mitigation Suggestion

Remove authority from Transact.

Remediation

Fixed in commit 1e7497f99bd59b70d9a6c8542a9958f5ae892f95.

ID	ACC-I2
Title	Unrestricted Merkle Tree Initialization Enables Abuse
Severity	info
Status	fixed

Description

Typically, a single Merkle tree should be managed per contract, and if multiple are allowed, they must be controlled by a trusted authority. Currently, anyone can call `initialize` to create a Merkle tree account under this program. This opens the door for further attacks or might denial-of-service (DoS) attacks—for example, by setting deposit limits to block legitimate users.

Location

https://github.com/SocialfiPanda/privacy-cash/blob/bd117d4641292fc893e529068c922a3fab1f57fd/anchor/programs/zkcash/src/lib.rs#L309-L310

Mitigation Suggestion

Restrict the `initialize` function so only a trusted authority can create Merkle tree accounts. Hardcode or validate the `authority` parameter to ensure only one (or a limited number of) valid Merkle tree(s) can be created and controlled securely.

Remediation

Fixed in 476618cc9af3b1d877c277d47bd3ce95e2f9e2ab.

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.