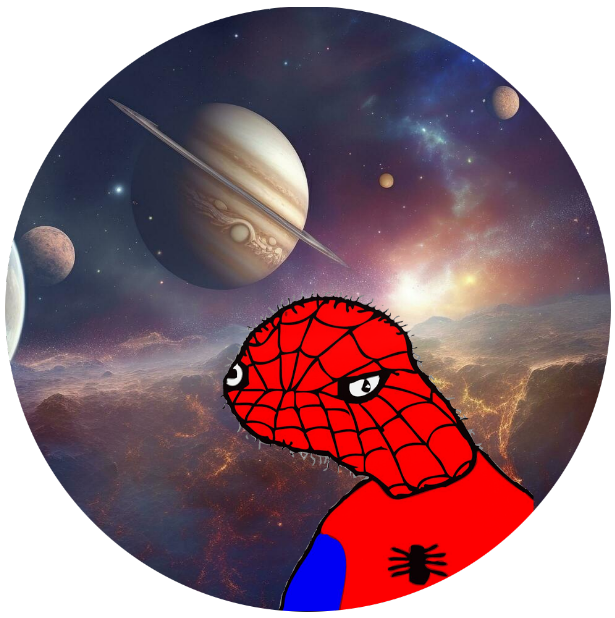

Privacy Cash - USDC integration Audit Report

Version 1.1

Zigtur

December 8, 2025

Privacy Cash - USDC integration - Audit Report

Zigtur

December 8, 2025

Prepared by: Zigtur

Table of Contents

Introduction

Disclaimer

A smart contract security review cannot guarantee the complete absence of vulnerabilities. This effort, bound by time, resources, and expertise, aims to identify as many security issues as possible. However, there is no assurance of 100% security post-review, nor is there a guarantee that the review will uncover all potential problems in the smart contracts. It is highly recommended to conduct subsequent security reviews, implement bug bounty programs, and perform on-chain monitoring.

About Zigtur

Zigtur is an independent blockchain security researcher dedicated to enhancing the security of the blockchain ecosystem. With a history of identifying numerous security vulnerabilities across various protocols in public audit contests and private audits, **Zigtur** strives to contribute to the safety and reliability of blockchain projects through meticulous security research and reviews. Explore previous work here or reach out on X @zigtur.

About Privacy Cash

PrivacyCash is a decentralized protocol for secure and anonymous transactions built on Solana.

Using zero-knowledge proofs, it lets users deposit assets and withdraw them to different addresses, breaking the link between sender and receiver. By combining advanced cryptography with decentralized infrastructure, PrivacyCash restores financial privacy and freedom on public blockchains.

This upgrade aims to support USDC SPL token.

Security Assessment Summary

Review commit hash - d2711c1f1f1deebe61278a8d3033ded85629e09e

Fixes review commit hash - PR7 - ebcc3bbb3f9628aa7ff64293120e104beb3684cf

Additional review commit hash - b62c1e733cfc43616f945de86be7b415c0ca2def

Deployment chains

- Solana

Scope

The following code is in scope of the review:

- anchor/programs/zkcash/src/lib.rs
- anchor/programs/zkcash/src/utils.rs

Risk Classification

	Impact: High	**Impact:** Medium	**Impact:** Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Issues

HIGH-01 - SOL deposits can be withdrawn as USDC

Scope:

- transaction.circom#L28-L29
- lib.rs#L176-L264
- lib.rs#L271-L390

Description

The protocol allows users to deposit assets in the same merkle tree.

However, there is a lack of constraints to ensure that the `mintAddress` being used in the proof corresponds to the `extData.mintAddress`.

This allows an attacker to deposit SOL and withdraw USDC. Due to the decimals difference, this can lead to loss of funds.

```
template Transaction(levels, nIns, nOuts) {
    signal input root;
    signal input publicAmount;
    signal input extDataHash;
    signal input mintAddress; // Single mint address for entire transaction

    // Input commitments are reconstructed using this mintAddress
    inCommitmentHasher[tx].inputs[0] <== inAmount[tx];
    inCommitmentHasher[tx].inputs[1] <== inKeypair[tx].publicKey;
    inCommitmentHasher[tx].inputs[2] <== inBlinding[tx];
    inCommitmentHasher[tx].inputs[3] <== mintAddress; // Uses same mint for inputs

    // Output commitments also use the same mintAddress
    outCommitmentHasher[tx].inputs[0] <== outAmount[tx];
    outCommitmentHasher[tx].inputs[1] <== outPubkey[tx];
    outCommitmentHasher[tx].inputs[2] <== outBlinding[tx];
    outCommitmentHasher[tx].inputs[3] <== mintAddress; // Uses same mint for
     ↪  outputs
}
```

Recommendation

Each supported SPL token should have their own merkle root to ensure correct separation of transfers. This requires modifying the program to maintain separate merkle trees per mint address, preventing cross-contamination between different token types.

Another solution would be to constrain the external data hash in the circuits to ensure that the `mintAddress` matches the expected value for the inputs being spent. This is more complex as it requires modifying the circuit and re-executing the trusted setup ceremony.

Privacy Cash

Fixed in PR7.

Zigtur

Fixed. There is now one merkle tree per asset.

INFO-01 - Incorrect USDC address

Scope:

- lib.rs#L33

Description

The program currently uses a devnet USDC address in the allowed tokens list instead of the mainnet USDC address. The code includes a comment indicating this should be changed before mainnet deployment.

```
#[cfg(any(feature = "localnet", test))]
pub const ALLOW_ALL_SPL_TOKENS: bool = true;

#[cfg(not(any(feature = "localnet", test)))]
pub const ALLOW_ALL_SPL_TOKENS: bool = false;

// IMPORTANT!!!!!!!: Change it back to EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v
↪  after devnet testing is done!!!!!!!
pub const ALLOWED_TOKENS: &[Pubkey] =
↪  &[pubkey!("4zMMC9srt5Ri5X14GAgXhaHii3GnPAEERYPJgZJDncDU")];
```

The current address `4zMMC9srt5Ri5X14GAgXhaHii3GnPAEERYPJgZJDncDU` is a devnet token mint address, while the mainnet USDC address should be `EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v`. Using the wrong address in production would prevent users from depositing or withdrawing actual USDC tokens.

Recommendation

Before mainnet deployment, update the `ALLOWED_TOKENS` array to use the correct mainnet USDC mint address `EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v`. Consider adding a compile-time check or deployment script validation to ensure the correct address is used for the target network.

Privacy Cash

Fixed in PR7.

Zigtur

Fixed. The `ALLOWED_TOKENS` now depends on a feature flag.

INFO-02 - Unused function in utils.rs

Scope:

- utils.rs#L274-L309

Description

The function `calculate_complete_ext_data_hash_for_sol` is defined but never used in the codebase. This function is identical to `calculate_complete_ext_data_hash` and appears to be redundant dead code.

```rust
/**
 * Calculate ExtData hash with encrypted outputs included
 * This matches the client-side calculation for hash verification
 */
pub fn calculate_complete_ext_data_hash_for_sol(
    recipient: Pubkey,
    ext_amount: i64,
    encrypted_output1: &[u8],
    encrypted_output2: &[u8],
    fee: u64,
    fee_recipient: Pubkey,
    mint_address: Pubkey,
) -> Result<[u8; 32]> {
    // ... identical implementation to calculate_complete_ext_data_hash
}
```

The `transact` function uses `calculate_complete_ext_data_hash` directly, making this duplicate function unnecessary. Having unused code increases maintenance burden and can cause confusion about which function should be used.

Recommendation

Remove the unused `calculate_complete_ext_data_hash_for_sol` function from the codebase. The existing `calculate_complete_ext_data_hash` function already provides the same functionality and is actively used by the `transact` instruction.

Privacy Cash

Fixed in PR7.

Zigtur

Fixed. Function has been removed.

INFO-03 - Anchor macros could be used to reduce `transact_spl` code size

Scope:

- lib.rs#L289-L297

Description

The `transact_spl` instruction manually validates that the signer's token account has the correct owner and mint address. This validation is performed in the function body, but Anchor provides constraint macros that can enforce these requirements declaratively in the account validation structure.

```
pub fn transact_spl(ctx: Context<TransactSpl>, proof: Proof, ext_data_minified:
↪  ExtDataMinified, encrypted_output1: Vec<u8>, encrypted_output2: Vec<u8>) ->
↪  Result<()> {
    let tree_account = &mut ctx.accounts.tree_account.load_mut()?;
    let global_config = &ctx.accounts.global_config;

    // Validate signer's token account ownership and mint
    require!(
        ctx.accounts.signer_token_account.owner == ctx.accounts.signer.key(),
        ErrorCode::InvalidTokenAccount
    );
    require!(
        ctx.accounts.signer_token_account.mint == ctx.accounts.mint.key(),
        ErrorCode::InvalidTokenAccountMintAddress
    );
    // ... rest of function
}
```

This approach adds unnecessary code to the instruction body and performs validation after deserialization, when it could be enforced during the account validation phase.

Recommendation

Use Anchor's constraint macros in the `TransactSpl` accounts structure to enforce these requirements declaratively:

This approach reduces the function body code, improves readability, and ensures validation happens during the account deserialization phase before the instruction logic executes.

Privacy Cash

Acknowledged.

Zigtur

Acknowledged.

INFO-04 - Global config is used as authority for tree ATA instead of tree token account

Scope:

- lib.rs#L647-L643
- lib.rs#L350-L356

Description

The tree's associated token account for SPL tokens uses `global_config` as its authority instead of the more semantically appropriate `tree_token_account`. While both are PDAs controlled by the program, using `global_config` as the authority creates an inconsistent architecture where configuration accounts have transfer authority.

```
/// Tree's associated token account (destination for deposits, source for
↪  withdrawals)
/// Created automatically if it doesn't exist
#[account(
    init_if_needed,
    payer = signer,
    associated_token::mint = mint,
    associated_token::authority = global_config // Uses global_config as authority
)]
pub tree_ata: Account<'info, TokenAccount>,

// Later in the function, transfers use global_config as the signer
let bump = &[ctx.accounts.global_config.bump];
let seeds: &[&[u8]] = &[b"global_config", bump];
let signer_seeds = &[seeds];

token::transfer(
    CpiContext::new_with_signer(
        ctx.accounts.token_program.to_account_info(),
        SplTransfer {
            from: ctx.accounts.tree_ata.to_account_info(),
            to: ctx.accounts.recipient_token_account.to_account_info(),
            authority: ctx.accounts.global_config.to_account_info(), // Signs with
            ↪  global_config
        },
        signer_seeds,
    ),
    ext_amount_abs,
)?;
```

This design choice works functionally but creates architectural confusion. The `tree_token_account` is the natural authority for tree-held assets, as it already serves this role for SOL in the `transact` instruction. Using different authorities for SOL (tree_token_account) and SPL tokens (global_config) introduces inconsistency.

Recommendation

Consider refactoring to use `tree_token_account` as the authority for the tree's ATA, maintaining consistency with the SOL transaction flow. If `global_config` must be used, document the architectural decision explaining why configuration accounts have token transfer authority. Alternatively, if the team decides to keep the current implementation, ensure this design is intentional and not an oversight.

Privacy Cash

Acknowledged.

Zigtur

Acknowledged.