Privacy Cash - Privacy Cash MultiSPL Audit Report

Version 1.0

Zigtur

December 23, 2025

Privacy Cash - Privacy Cash MultiSPL - Audit Report

Zigtur

December 22, 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.

Security Assessment Summary

Review commit hash - PR 66 - commit 9c047607 & PR 9 - commit aa60ccff

Fixes review commit hash - PR 67

Additional review commit hash - 420e7bc

Deployment chains

- Solana

Scope

The scope of this review is the implementation of multiple SPL tokens. This includes PrivacyCash PR 9 and PrivacyCash SDK PR 66.

The following code is in scope of the review:

- anchor/programs/zkcash/src/lib.rs
- src/config.ts
- src/deposit.ts
- src/depositSPL.ts
- src/exportUtils.ts
- src/getUtxosSPL.ts
- src/index.ts
- src/utils/constants.ts
- src/withdrawSPL.ts

Note: Issues reported and acknowledged in the previous SDK reviews will not be reported in this report. Multiple privacy leak vectors were previously reported, this impact is out of scope.

Risk Classification

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

Issues

MEDIUM-01 - Incorrect ORE Solana address

Scope:

- constants.ts#L68
- lib.rs#L52

Description

The Privacy Cash SDK configures the ORE token address as `oreo7nRnUq6W2G6Nr4s2959seHveG7ojfXpWH9yUCnC`.

However, the Privacy Cash Solana program uses the correct ORE address: `oreoU2P8bN6jkk3jbaiVxYnG1dCXcYxwhwyK9j`

Recommendation

Use the correct `oreoU2P8bN6jkk3jbaiVxYnG1dCXcYxwhwyK9jSybcp` address for ORE.

Privacy Cash

Fixed in PR67.

Zigtur

Fixed. Correct address has been set.

LOW-01 - Incorrect ZEC environment variable

Scope:

- constants.ts#L61

Description

The ZEC token configuration incorrectly uses the `NEXT_PUBLIC_USDT_MINT` environment variable instead of a dedicated `NEXT_PUBLIC_ZEC_MINT` environment variable:

```
{
    name: 'zec',
    pubkey: process.env.NEXT_PUBLIC_USDT_MINT  // Should be NEXT_PUBLIC_ZEC_MINT
        ? new PublicKey(process.env.NEXT_PUBLIC_USDT_MINT)
        : new PublicKey('A7bdiYdS5GjqGFtxf17ppRHtDKPkkRqbKtR27dxvQXaS'),
    prefix: 'zec_',
    units_per_token: 1e8
}
```

This means that if a user sets `NEXT_PUBLIC_USDT_MINT` to override the USDT token address, it will also override the ZEC token address, causing ZEC operations to interact with the wrong token.

Recommendation

Use a dedicated `NEXT_PUBLIC_ZEC_MINT` environment variable for the ZEC token configuration:

```
{
    name: 'zec',
    pubkey: process.env.NEXT_PUBLIC_ZEC_MINT
        ? new PublicKey(process.env.NEXT_PUBLIC_ZEC_MINT)
        : new PublicKey('A7bdiYdS5GjqGFtxf17ppRHtDKPkkRqbKtR27dxvQXaS'),
    prefix: 'zec_',
    units_per_token: 1e8
}
```

Privacy Cash

Fixed in PR67.

Zigtur

Fixed. `NEXT_PUBLIC_ZEC_MINT` variable is used.

LOW-02 - `checkDepositLimit` remainder calculations are incorrect

Scope:

- depositSPL.ts#L546-L560

Description

The `checkDepositLimit` function calculates fractional token amounts using a hardcoded `1e6` divisor instead of using the token's actual `units_per_token` value:

```
const unitesPerToken = new BN(token.units_per_token);
const maxDepositSpl = maxDepositAmount.div(unitesPerToken);
const remainder = maxDepositAmount.mod(unitesPerToken);

let amountFormatted = '1';
if (remainder.eq(new BN(0))) {
    amountFormatted = maxDepositSpl.toString();
} else {
    // Handle fractional SOL by converting remainder to decimal
    const fractional = remainder.toNumber() / 1e6;  // Hardcoded 1e6 is incorrect
    amountFormatted =
    ↪  `${maxDepositSpl.toString()}${fractional.toFixed(9).substring(1)}`;
}
```

The `fractional` calculation divides the remainder by `1e6`, but this is only correct for tokens with 6 decimals (USDC, USDT). For other tokens: - SOL (9 decimals): Should divide by `1e9` - ZEC (8 decimals): Should divide by `1e8` - ORE (11 decimals): Should divide by `1e11`

This results in incorrectly formatted deposit limit amounts being returned for non-6-decimal tokens.

Recommendation

Use the token's `units_per_token` value consistently for the fractional calculation:

```
const fractional = remainder.toNumber() / token.units_per_token;
```

Privacy Cash

Fixed in PR67.

Zigtur

Fixed. The scaling is now token dependent.

INFO-01 - Logs and error messages use hardcoded “USDC” token naming

Scope:

- depositSPL.ts#L122-L141

Description

The `depositSPL` function uses hardcoded “USDC” strings in log messages and error messages, even when operating on other SPL tokens:

```
if (limitAmount && base_units > limitAmount * token.units_per_token) {
    throw new Error(`Don't deposit more than ${limitAmount} USDC`)  // Hardcoded
    ↪  "USDC"
}

logger.debug(`Deposit amount: ${base_units} base_units (${base_units /
↪  units_per_token} USDC)`);
logger.debug(`Calculated fee: ${fee_base_units} base_units (${fee_base_units /
↪  units_per_token} USDC)`);
logger.debug(`USDC wallet balance: ${balance / units_per_token} USDC`);

if (balance < (base_units + fee_base_units)) {
    throw new Error(`Insufficient balance. Need at least ${(base_units +
    ↪  fee_base_units) / units_per_token} USDC.`);
}
```

This causes confusing output when users deposit other tokens like USDT, ZEC, or ORE, as the logs will still display “USDC”.

Recommendation

Use the token’s name dynamically in all log and error messages:

```
throw new Error(`Don't deposit more than ${limitAmount}
↪  ${token.name.toUpperCase()}`);
logger.debug(`Deposit amount: ${base_units} base_units (${base_units /
↪  units_per_token} ${token.name.toUpperCase()})`);
```

Privacy Cash

Fixed in PR67.

Zigtur

Fixed. The naming is now token dependent.

INFO-02 - Typo in error messages

Scope:

- depositSPL.ts#L92
- withdrawSPL.ts#L74

Description

The error messages in both `depositSPL` and `withdrawSPL` contain a typo - “leaset” instead of “least”:

```
throw new Error('You must input at leaset one of "base_units" or "amount"')
```

Recommendation

Correct the typo:

```
throw new Error('You must input at least one of "base_units" or "amount"')
```

Privacy Cash

Fixed in PR67.

Zigtur

Fixed.

INFO-03 - Inconsistent `units_per_token` sources

Scope:

- depositSPL.ts#L93
- withdrawSPL.ts#L77

Description

The code retrieves `units_per_token` from two different sources: 1. From `token.units_per_token` (hardcoded in constants.ts) 2. From `mintInfo.decimals` via `10 ** mintInfo.decimals`

```
let mintInfo = await getMint(connection, token.pubkey)
let units_per_token = 10 ** mintInfo.decimals  // From on-chain mint info

// But later uses:
base_units = amount * token.units_per_token    // From hardcoded constants
```

These values could potentially diverge if: - The hardcoded `units_per_token` in constants.ts is incorrect - The token's on-chain decimals differ from expectations

Recommendation

Use a single source of truth for `units_per_token`. Preferably use the on-chain `mintInfo.decimals` as the authoritative source, or validate that both values match and throw an error if they diverge.

Privacy Cash

Fixed in PR67.

Zigtur

Fixed. `units_per_token` variable initialization is now commented out. The `token.units_per_token` is used instead.

Note: The fix is correct but depends on SDK configuration and not on-chain state. It is OK as decimals modification are not likely to occur.