# Privacy Cash security review

## Intro

This security review was conducted by kriko.eth. The subject was the `Privacy Cash` Solana program and its ZK circuit at commit `5496860`.

Subsequently, changes and additions from commits `2ac0e89` and `616f4bd` were also reviewed.

## Risk Classification

The risk is assessed based on the severity matrix:

| Likelihood | | Impact | |
|---|---|---|---|
| | HIGH | MEDIUM | LOW |
| HIGH | **CRITICAL** | **HIGH** | **MEDIUM** |
| MEDIUM | **HIGH** | **MEDIUM** | **LOW** |
| LOW | **MEDIUM** | **LOW** | **INFO** |

## Findings Summary

Total issues discovered:

| Severity | Count |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 0 |
| Info | 2 |

## Engagement Summary

The subject Solana program and circuits were reviewed and no critical security issue or an issue leading to loss of user funds was discovered. One informational finding was identified regarding fee recipient bypassing, which the client confirmed is intentional by design to maintain decentralization.

During the upgrade review, another informational finding was reported regarding vulnerable crate usage.

# Informational Findings

## PRCS-I01: Users can bypass protocol fees by controlling fee recipient in proof generation

### Summary

The unconstrained fee recipient account allows users to generate proofs with their own accounts as fee recipients, enabling fee bypass and avoidance.

### Finding Description

The `transact` function accepts an unconstrained `fee_recipient_account` that must match the fee recipient included in the user-generated zero-knowledge proof's external data hash. Since users generate their own proofs off-chain, they can specify the fee recipient account to an account they control:

```
#[account(mut)]
/// CHECK: user should be able to send fees to any types of accounts
pub fee_recipient_account: UncheckedAccount<'info>,
```

While the protocol likely expects users to use a designated fee recipient via the frontend, technically sophisticated users can bypass the frontend and generate proofs with their own wallet as the fee recipient. The proof will validate successfully since the fee recipient was correctly included in the hash, but fees flow to the user's account instead of the protocol.

### Impact Explanation

This vulnerability allows users to completely bypass and avoid protocol fees by redirecting them to accounts they control, undermining the protocol's economic model and sustainability.

### Likelihood Explanation

The issue requires technical knowledge to generate custom proofs and bypass the frontend, but provides strong economic incentive through substantial fee savings on high-value transactions.

### Recommendation

Consider creating a fee recipient PDA that accumulates all fees, with admin-controlled withdrawal functionality and constrain the `fee_recipient` account to match this PDA's seeds. This would ensure all fees flow to a deterministic protocol-controlled account.

### Client response

The client indicated this behavior is by design, emphasizing that the protocol should remain fully decentralized.

## PRCS-I02: Usage of vulnerable crate in dependencies

### Finding Description

Running `cargo-audit` results in 1 error and three warnings. The error (produced by the usage of vulnerable crates) can be seen below:

```
Crate:     curve25519-dalek
Version:   3.2.1
Title:     Timing variability in `curve25519-dalek`'s `Scalar29::sub`/`Scalar52::sub`
Date:      2024-06-18
ID:        RUSTSEC-2024-0344
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0344
Solution:  Upgrade to >=4.1.3
```

This vulnerable crate is used by the `light-hasher` crate `v2.0.0`. A new version of this crate is available (see here).

While the crate contains a timing side-channel vulnerability that could theoretically leak sensitive cryptographic material, the Solana runtime environment makes practical exploitation infeasible. The BPF VM isolation, nondeterministic execution timing due to network conditions and validator load, and lack of precise timing measurement capabilities in the Solana context significantly mitigate this vulnerability. However, it remains good practice to use the latest secure versions of dependencies.

### Recommendation

Consider updating the program's dependencies to the latest versions. Additionally, always ensure to use safe dependencies by running `cargo-audit` regularly.