

Privacy Cash security review

Intro

This security review was conducted by krik0.eth. The subject was the upgrade of Privacy Cash Solana program allowing users to transact with SPL tokens at commit 4b2ec49.

Disclaimer

This security review represents a best-effort analysis of the Privacy Cash protocol based on the provided codebase and documentation. While comprehensive testing methodologies and manual code review were employed to identify potential vulnerabilities, no security review can guarantee the absence of all possible issues. The findings presented herein should be considered alongside additional security measures, including but not limited to formal verification, extended testing periods, and ongoing security monitoring. Users and developers should exercise appropriate caution and implement additional safeguards as deemed necessary for their specific use cases.

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	1
High	0
Medium	0
Low	1
Info	5

Engagement Summary

The subject Solana program and circuits were reviewed. One critical vulnerability was identified allowing cross-mint token swapping. Additionally, one low-severity issue and several informational findings regarding code quality, optimization and documentation were identified.

The team addressed the critical and low-severity issues, along with three informational findings. Two informational findings were acknowledged.

Following the initial security review, a mitigation review was conducted to verify fixes implemented in commit `b62c1e7`.

A subsequent review verified the safe integration of newly added tokens in commit `6027242` without identifying additional security issues.

Critical Severity Findings

PRC2-C01: Users can withdraw different mint than deposited

Summary

Insufficient validation between proof generation parameters and transaction execution allows users to deposit SOL and withdraw SPL tokens, resulting in protocol funds drainage.

Finding Description

The protocol lacks validation to ensure that the mint address used in the external data hash matches the mint address passed to the ZK circuit. This allows attackers to generate the external data hash with one mint (e.g., SOL) while passing a different mint (e.g., USDC) to the ZK proof, effectively swapping tokens at a 1:1 ratio regardless of their actual market values.

The vulnerability stems from the fact that the external data hash which is part of the ZK proof is not required to match the mint passed into the ZK circuit. This allows attackers to generate the external data hash with one mint while passing a different mint to the ZK proof, then withdrawing a different token type than what was deposited.

This enables an attacker to:

1. Deposit 1 SOL (1,000,000,000 lamports, ~\$154 USD at the time of writing)
2. Generate external data hash using USDC mint while passing SOL mint to the ZK circuit
3. Execute a withdrawal using `transact_spl` with USDC mint
4. Withdraw 1,000,000,000 USDC units (1,000 USDC, around 6.5x the value)

The same attack vector applies to any combination of supported tokens, allowing arbitrary cross-mint swapping at incorrect exchange rates.

Impact Explanation

High. This vulnerability allows attackers to extract significantly more value than they deposit.

Likelihood Explanation

High. The attack requires no special privileges and can be executed by any user with basic understanding of the proof generation and transaction flow.

Proof of Concept

The proof of concept of this issue can be found on the following gist link: <https://gist.github.com/coregon11/ee79ff21b22ab9fd51feba0ddfd6a6b6>

Recommendation

Consider implementing additional validation to ensure consistency between the mint addresses used in the external data hash and the ZK proof.

Status

Fixed

Low Severity Findings

PRC2-L01: Unnecessary 31-byte constraint applied to mint address in hash calculation

Finding Description

The code constrains the mint address to 31 bytes during external data hash calculation based on the 254-bit circuit constraint:

```
let mint_bytes_for_hash: &[u8] = &ext_data.mint_address.to_bytes()[..31];
```

Additionally, if the intention was to limit data to exactly 31 bytes, the current approach actually results in 35 bytes since Borsh serialization adds 4 bytes for the vector size.

However, the 254-bit constraint only applies to the mint address signal passed to the circuit, not the mint address used in hash calculation. The hash calculation operates off-chain and is not subject to circuit field constraints. While this truncation doesn't create security vulnerabilities due to the limited token allowlist, using the full 32-byte mint address for hash calculation would be more consistent.

Recommendation

Consider using the full 32-byte mint address for hash calculation since circuit constraints do not apply to hash generation parameters. Alternatively, should 31 bytes be used, consider using the `[u8; 31]` type instead, since arrays of fixed size are not prepended with any other data. Ensure clear documentation distinguishing between circuit signal constraints and hash calculation requirements.

Status

Fixed

Informational Findings

PRC2-I01: Misleading function comment claims SOL-only functionality

Finding Description

The `update_global_config` function contains a misleading comment that claims it updates “global configuration for SOL” when in reality the function affects both SOL and SPL token operations. While the function behaves correctly and the current implementation is expected, the inline documentation does not reflect the true scope of its impact.

Consider updating the function comment to accurately reflect that it updates global configuration parameters that affect both SOL and SPL token operations, rather than being SOL-specific.

Status

Fixed

PRC2-I02: Inconsistent authority design between SOL and SPL token storage

Finding Description

The protocol uses inconsistent escrowing of assets. SOL is escrowed in the `tree_token_account` while SPL tokens use the associated token accounts of the `global_config` authority. This design mixes configuration management with asset custody, creating a single point of control over both protocol parameters and user funds. Consider establishing consistent authority patterns between SOL and SPL token storage.

Status

Acknowledged

PRC2-I03: Hardcoded development IDs exposed in production code

Finding Description

The codebase contains multiple hardcoded development identifiers with explicit comments indicating they should be changed back to production values after devnet testing. These include the program ID (`declare_id!`), `ADMIN_PUBKEY` constant, and `ALLOWED_TOKENS` array, all accompanied by comments requiring manual changes. This approach creates deployment risks where developers must remember to manually change these values before production deployment, potentially leading to incorrect identifiers being used in production. Consider implementing proper feature flag management for all environment-specific identifiers using conditional compilation to automatically select appropriate values based on build target.

Status

Fixed

PRC2-I04: Unused function in utils module

Finding Description

The `calculate_complete_ext_data_hash_for_sol` function is declared in the `utils` module but appears to be unused throughout the codebase. Consider removing the unused function to reduce codebase complexity and prevent potential confusion during future development.

Status

Fixed

PRC2-I05: Redundant and conditionally unused accounts in TransactSpl struct

Finding Description

The `TransactSpl` accounts struct declares several accounts that are either redundant or only conditionally used, leading to inefficient account usage. The `recipient` account is completely redundant as it is only used to derive the `recipient_token_account` ATA while the specific token account address is already included in the external data hash. Additionally, `signer_token_account` is only needed for deposits and `recipient_token_account` is only needed for withdrawals, yet both must be provided in every transaction regardless of operation type.

Recommendation

Consider removing the completely redundant `recipient` account since recipient information is already validated through the external data hash verification. For conditionally used accounts like `signer_token_account` and `recipient_token_account`, consider implementing optional account handling using `Option` types to avoid requiring unused accounts in transactions.

Status

Acknowledged