HashCloak

Code Review and Security Assessment
For
Privacy Cash

Updated Version 3: October 9th, 2025

Prepared For:
Privacy Cash

Prepared by:
Manish Kumar | *HashCloak Inc.*

Table Of Contents

Executive Summary

Privacy Cash engaged *HashCloak Inc*. to perform code review and security assessment for zkCash, an implementation of Tornado Cash Nova for Solana, a private transaction protocol for Solana.

During the audit period, we began familiarizing ourselves with the overall concepts of Tornado Cash, its usage in Solana, and its core components relevant to the audit scope by reviewing the code and the provided documentation. We also assessed some security vulnerabilities commonly found in Solana and Circom projects. Subsequently, we examined the code for errors, primarily focusing on logical correctness and identifying potential issues that could lead to double spending, loss of funds, or other privacy-related issues. We also ran some static analyzer tools such as X-ray, radar and circumspect to find common vulnerabilities in anchor programs and Circom circuits.

After the initial report on July 8th, the Privacy Cash team provided fixes for each issue, along with additional sets of commits on July 11th (Merkle tree height change), July 15th (ceremony script and other changes such as rent exemption), July 16th (possible private file leak), and August 6th (final commit). We reviewed each of these to ensure that no new vulnerabilities were introduced into the codebase.

In the latter half of the audit, we examined the ceremony script for generating the Groth16 CRS, the constraint condition for extDataHash in the circuit, and the rent exemption checks for accounts in the anchor program. We also reviewed the update to the Merkle tree height from 26 to 32, which was later reverted back to 26. Additionally, we verified whether the generated Circom artifacts in the GitHub repository were intended to be public, in order to prevent any accidental leakage of private files. Finally, we reviewed the last commit to confirm that everything was in place.

Overall, we found the source code is of good quality in representing the entities in the specification. The code is modular and well-structured.

After the delivery, a critical double-spending bug was identified. It was caused by the use of two different seed prefixes for nullifiers,nullifier0 and nullifier1. This allowed a second transaction (with swapped nullifiers) to be processed (see ZKC-7). Fixes were

applied in commit 7d2b39bce274df41b7d30f0aef5945c89e1a6ea7 for the anchor program and 2c6061029d72626b87cb4584937fc3d085106334 for the interface, and were later reviewed.

We also conducted a re-review of the final code commits to identify any potential bugs that might have been overlooked. During this process, we identified two low-severity issues (out-of-scope), primarily related to the generation of encryption keys for the UTXOs.

In addition, we reviewed the final commit 549686066e81c5434182f9f85b9296bb636b07e9 in the Privacy-Cash repository to ensure that all planned changes and fixes were in place and properly implemented so that it does not introduce any new bugs. We found everything is in place, with no further concerns identified within the defined scope.

Later, a new PR related to logging commitment metadata was also reviewed. No issues were identified, and all changes appeared to be in order.

We found the issues range from Critical to Informational:

Severity	**Number of Findings**
Critical	1
High	1
Medium	3
Low	1
Informational	3

Version Info

Versions	Delivery Date
Initial Delivery	July 4th, 2025
Updated Delivery	July 8th, 2025
Final Version 1	August 9th, 2025
Updated Version	September 1st, 2025
Updated Version 2	September 30th, 2025
Final Version 2	October 9th, 2025

Scope

For the audit, we considered the following repositories:

- https://github.com/SocialfiPanda/privacy-cash at commit a89a7fc6577cad66fd6ab740d6d3c48296747326
- https://github.com/Privacy-Cash/privacy-cash-interface at commit 3ede09c9c55b0b62105eabeb33ce7c67e07d706e
- https://github.com/SocialfiPanda/privacy-cash at commit 6e4dcd32e0af55012e85788bec00c6cb63fc1924 for merkle tree height from 26 to 32
- https://github.com/SocialfiPanda/privacy-cash/ at commit b05bf950433afde1b86e4cad50583357c49b4996
- https://github.com/SocialfiPanda/privacy-cash at commit 9e8e9d5dc5b843e01a35dc5b3c2cd65043282af2 which contains scripts for the ceremony
- The final commit: https://github.com/Privacy-Cash/privacy-cash/ at 188f0cd475397c4039bf126f7962282170395aad

With following files in scope:

- Privacy-cash
    - anchor/programs/*

    - circuits/*
- Privacy-cash-interface
    - src/utils/prover.ts
    - src/utils/deposit.ts
    - src/utils/withdraw.ts

Later, multiple additional commits were provided which were reviewed for any potential bugs and the above-mentioned files in scope are listed below:

- https://github.com/Privacy-Cash/privacy-cash/ at commit 06b09014ceb7abc45fddc0d422242ff9ab5315e0
- https://github.com/Privacy-Cash/privacy-cash-interface at commit 9d14d3891217db008fd68baee72c750a7e015fc5
- https://github.com/SocialfiPanda/privacy-cash/pull/12/ for encryption keys (these files were not included in the original scope)
- https://github.com/Privacy-Cash/privacy-cash/ at 549686066e81c5434182f9f85b9296bb636b07e9
- https://github.com/Privacy-Cash/privacy-cash/pull/1 for logging commitment metadata

Overview

Privacy Cash is private transaction protocol for Solana. The protocol implements a privacy-preserving transaction system on Solana, drawing inspiration from Tornado Cash but adapted for the Solana environment using the Anchor framework. It allows users to privately deposit and withdraw funds by leveraging zero-knowledge proofs (specifically Groth16) and a Merkle tree structure. Each deposit generates a unique commitment that is inserted into a Merkle tree maintained on-chain. Withdrawals are performed by proving knowledge of a valid note inside the tree using a zero-knowledge proof and corresponding nullifier to prevent double-spending.

Methodology

The audit was conducted through a combination of manual verification and using static analysis tools and cross verifying the vulnerabilities. The primary focus was to map the available specifications and existing resources such as Tornado Nova and Light Protocol v1 to the actual implementation and also using different static analyzers for anchor program and circom circuits such as X-ray, Radar and Circomspect (see references). We also assessed the code for sufficient code coverage and various types of vulnerabilities, including:

- Authorization checks while transact
- Nullifier checks for double spending
- Merkle root validation
- Fee Logic and Transfers
- Account ownership
- Underconstrained circuits
- Integer underflow/overflow in circuits and anchor program
- Assigned but not constrained circuits
- Look for any dead code
- Any attack that can impact user funds
- Any undefined behavior
- Any transaction replay or double spending possibility
- Validation and access control of the authority
- account checks

Overview of Evaluated Components

circuits

- Integer overflow/underflow
- Underconstrained circuits
- Nondeterministic Circuits
- Assigned but not Constrained circuits
- Constant Branching (if-else, loops)
- amount invariant
- correctness of transaction outputs

anchor/programs

- Authorization checks while transact
- Nullifier checks for double spending
- Merkle root validation
- Fee Logic and Transfers
- Account ownership
- Integer underflow/overflow
- Look for any dead code
- Any attack that can impact user funds
- Any undefined behavior
- Any transaction replay or double spending possibility
- Validation and access control of the authority
- account checks

privacy-cash-interface

- Deposit and withdrawal flow
- Derivation of deterministic UTXO private key
- Groth16 proof generation flow using snarkjs
- Well-formation of UTXOs (including dummy)
- Amount invariance

Findings

ZKC-1: Underconstrained check in the circuit for creating proof for transaction

Type: High
Files affected:

- circuits/transaction.circom

Description: In `transaction.circom`, two private inputs are provided namely `inMintAddress[nIns]` and `outMintAddress[nOuts]` but there is no constrained check on the equality of the mint addresses. It should also be noted that the amount invariant in the circuit does not take into account the multiple tokens. So, either the input should be replaced with a single `MintAddress` or an equality constraint check should be enforced on the entire `inMintAddress[i]` and `outMintAddress[j]` for i, j ranging 0..2. Ideally, the constrained check should be avoided because of extra computation.

Impact: Underconstrained circuit can lead to fake proofs generation which can result in double-spending and/or unauthorized withdrawals.

Recommendation: Add a single mintAddress instead of an array of inMintAddress and outMintAddress.

Status: The team provided a fix at commit [2bc8a6722055a09bcae4910257da2c80783a81ba]

ZKC-2: Incorrect `nr_pubinputs` while defining `Groth16Verifyingkey` const

Type: Medium
Files affected:

- anchor/programs/zkcash/src/utils.rs
- anchor/programs/zkcash/tests/unit/utils_test.rs

Description: while constructing a new `Groth16Verifier`, the function takes a const generic parameter `NR_INPUTS` which for the privacy cash verifier is 7 because the `VERIFYING_KEY` defined in utils.rs has `len(vk_ic)` = 8. But while fixing issue 1 and 2, the parameter was changed to 8 which is incorrect because `nr_pubinputs + 1 == len(vk_ic)`. The public inputs along with `vk_ic` are used to prepare inputs for the groth16 verifier and an incorrect number of public inputs can cause unwanted panic of the program. Although the `nr_pubinputs` is not used directly and the verifier relies on length of number of inputs rather than using the `nr_pubinputs` field of `Groth16Verifier`, still this can be potentially be an attack vector for parts of the codebase and result into unwanted panic of the program and hence should be changed to correct value.

Impact: can potentially introduce an attack vector for incorrect proof or unwanted panic of the program.

Recommendation: change `nr_pubinputs` to 7 in the affected files.

Status: The team provided a fix at commit [61120f943dbd1b44ecbb522e791f3632f72a26ff]

ZKC-3: Validation missing for `fee_recipient_account`

Type: Medium
Files affected:

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

Description: The `fee_recipient_account` account defined in `Transact` is not properly validated i.e., there is no constraint that ensures the program's intended recipient. This lets any malicious user call `transact()` and route the fee to their own address, stealing it. Right now the fee recipient is validated in the frontend but must be checked in the program as well.

Impact: A malicious user can route the fees to their own controlled address.

Recommendation: Add checks which ensures that fees can be deposited only in the intended recipient account.

Status: The team provided an initial fix at commit 83dd65b70badd4deef239fee6591240eeb50e7bb. Then, a new fix was provided here: 63447e9baa9b1869a787de90ebdfe26b8aac7f3b where fee_recipient_account was moved into ExtData instead of being read from the global configuration. This change ensures that validation is performed during the extdatahash check, preventing the centralization of the fee recipient. The intent is to support a decentralized network of relayers in the future.

ZKC-4: Missing error handling in the verification of proof

Type: Low
Files affected:

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

Description: In the function `verify_proof`, `unwrap()` is used while handling (de)seralization of points as well as while construction of groth16 verifier inputs. unwrapping without handling the error cases can lead to unwanted panic and abort of the program. This can also be a potential DoS vector.

Impact: Transaction will fail without explanation. Can also potentially lead to DoS.

Recommendation: Handle the error cases explicitly instead of unwrapping.

Status: The team provided a fix at commit bd117d4641292fc893e529068c922a3fab1f57fd

ZKC-5: Tree Account Reinitialization in `fn initialize`

Type: Informational
Files affected:

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

Description: Running the Radar static analysis tool on the Anchor program flagged an account re-initialization issue at the line `pub fn initialize(ctx: Context<Initialize>)`, due to the presence of the line `let tree_account = &mut ctx.accounts.tree_account.load_init()?;`. The issue is likely flagged because, if the initialize function is invoked more than once, it could allow re-initialization and overwriting of state. The authority should ensure that `initialize` is called only once to avoid re-initialization and overwriting of state.

Impact: Overwriting of tree account state can erase the root history, sub-trees and index because of which a user will not be able to prove the ownership of funds and can result in loss of funds.

Recommendation: The authority should ensure that `initialize` is called only once to avoid re-initialization and overwriting of state.

Status: The team provided the fix at the commit [476618cc9af3b1d877c277d47bd3ce95e2f9e2ab]

ZKC-6: Potential Overflow while calculating `sumOuts` in the circom circuit

Type: Informational
Files affected:

- circuits/transaction.circom

Description: In `transaction.circom`, the total output amount is calculated using `sumOuts += outAmount[tx]`, where tx ranges from 0 to 1. Each `outAmount` is individually constrained with `Num2Bits(248)` to ensure it fits within 248 bits, preventing individual overflow. However, there is no constraint to check for overflow on the total sum `sumOuts`.

An overflow could occur if `sumOuts` exceeds the BN254 field size (~254 bits). Currently, in Privacy Cash, both `nOuts` and `nIns` are fixed at 2, so the sum of two `outAmount` values will be at most ~249 bits, which cannot exceed the field size, thus avoiding overflow. Even if `nOuts` and `nIns` were increased to 16, overflow would still not occur. For example, assuming a maximum value of 2^248 – 1, the sum becomes:

```
16 × (2^248 - 1) = 16 × 2^248 - 16 = 2^252 - 16
```

This remains within the field limit. However, if `nOuts` and `nIns` exceed 16, there is a potential risk of overflow, which could compromise circuit correctness and soundness.

Impact: This can result in an incorrect amount invariant check, which could be exploited to generate fake proofs and potentially steal funds from the pool.

Recommendation: Ensure that `nOuts` and `nIns` are always less than 16 and/or add an overflow check on `sumOuts`.

Example fix: After total sumOut is calculated, we can do a constrained check on the sum as follows:

```
// check that sum of outputs fits into 252 bits to prevent overflow
// the sumOuts should be less than scalar field size which is around
253.59669 bits
// so we can safely use 252 bits for the sum
component sumOutsCheck = Num2Bits(252);
sumOutsCheck.in <== sumOuts;
```

Status: The team added a security note to ensure that nOuts and nIns are always less that 16: [a22a7c874108f0280a8774d93b9a57ef4e688271]

ZKC-7: Double Spending possible in `transact` function of anchor program

Type: Critical

Files affected:

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

Description:
When a user deposits or withdraws funds, the program creates two nullifier accounts using the `init` constraint. These accounts are created based on the nullifiers provided in the proof, with seeds `[b"nullifier0", proof.input_nullifiers[0].as_ref()]` and `[b"nullifier1", proof.input_nullifiers[1].as_ref()]` for input nullifier 0 and input nullifier 1, respectively. This action marks the inputs as spent. So, if a nullifier account already exists, the transaction automatically fails with a system program error, thus preventing double spend.

However, since the seed prefixes for the two nullifiers differ (`b"nullifier0"` and `b"nullifier1"`), a user can bypass this check and double-spend by simply swapping the input nullifiers in the provided proof. A transaction with swapped nullifiers will not fail because the PDA seeds for the new nullifier accounts will be different, thus allowing the double-spend to occur.

Impact: One can deposit tokens but can withdraw twice the amount deposited.

Recommendation: a single nullifier seed prefix instead of two separate `b"nullifier0"` and `b"nullifier1"`must be used for nullifier accounts so that a single global structure is maintained and is not domain separated because of seeds prefix. To avoid any breaking changes we can keep the current nullifiers account as it is but cross nullifier account initialization check should be added with seeds `[b"nullifier0",proof.input_nullifiers[1].as_ref()]` and `[b"nullifier1",proof.input_nullifiers[0].as_ref()]`. The proposed solution discussed was to define two cross nullifier accounts, namely `nullifier2` and `nullifier3` as `SystemAccount`. This would prevent double spend by rejecting a second transaction with swapped nullifiers, since the `nullifier2` and `nullifier3` accounts will be program owned and not system owned.

Status: The team has fixed the issue at commit [7d2b39bce274df41b7d30f0aef5945c89e1a6ea7] in the anchor program by defining

cross nullifiers, namely `nullifier2` and `nullifier3` as `SystemAccount.` If a second transaction were to be attempted, it will be rejected because the swapped nullifier accounts will be program owned and not system owned thus preventing the double spend. The interface was accordingly updated to incorporate the changes at the commit 2c6061029d72626b87cb4584937fc3d085106334 .

ZKC-8: Non-standard way of generating encryption keys encrypting the UTXOs and for deposits and withdrawals

Type: Medium
Files affected:

- src/utils/encryption.ts (out-of scope)

Description:
In the current system, the encryption key is generated by creating a signature on the fixed message: SIGN_MESSAGE = `Privacy Money account sign in` using the wallet keys and then extracting the first 31 bytes as the encryption key. Since Solana uses ed25519 signature scheme, the generated signature is deterministic in nature and hence so are the encryption keys. But this way of generating keys poses a risk of losing funds because if the user signs the same message anywhere else with their wallet, that will create the same deterministic signature which is used to create the encryption key and hence that exposes entire funds encrypted using the derived encryption key. The user needs to make sure he/she does not sign the same message anywhere else with their wallet which is hard to keep track of every time. Also, this is not a standard cryptographic method of deriving keys.

Impact: If the user signs the same message anywhere else with their wallet that will expose the encryption key as they are derived from signature.

Recommendation: instead of using signature to derive encryption key, a cryptographic Key-Derivation Function should be used to generate new key using the wallet private keys.

Status: The team has made a design tradeoff that signing this way will offer much

better UX. The users must ensure that they don't sign the same messages on the phishing sites or anywhere else. Elusiv used the same mechanism for their private transfer protocol.

ZKC-9: keys used in HMAC to create an authentication tag in `encrypt` uses less than 32 bytes would decrease the security strength of the function

Type: informational
Files affected:

- src/utils/encryption.ts (out-of scope)

Description: In the current system, for encrypting and decrypting the utxos, an HMAC along with AES encryption is used. The HMAC authentication ensures the authenticity and integrity of the data(i.e. UTXOs). The key for HMAC can be of any length however, less than byte-length of hash output is strongly discouraged as it would decrease the security strength of the function. In the current system, 15bytes key is used to create the hmac. This would decrease the security of the hmac authentication.

Impact: Reduces the security of HMAC to 120 bit can potentially have a forgery and/or collision on authentication tags for malicious encrypted data which reduces the overall security of encryption of the UTXOs. While the real life impact is still computationally infeasible it is less than the standard requirement.

Recommendation: According to standard recommendation 32 bytes keys should be used for SHA-256 based HMAC.

Status: This issue has been fixed in this PR: https://github.com/SocialfiPanda/privacy-cash/pull/12(file: scripts/utils/encryption.ts)

References

- Circomspect: Circom static analysis tool
- X-ray: Static analysis tool for solana programs
- Radar: A static analysis tool for anchor rust programs

- SolSec GitHub Repository by Sanny Kim
- Solana Documentation
- Helius Blog: A Hitchhiker's Guide to Solana Program Security
- Tornado Nova

Appendix

Results from Static Analysis tools: Most of them are found to be false positives

- **X-ray Static Tool**

```
**************** attack surface #1: sol.initialize ****************
**************** attack surface #2: sol.update_deposit_limit
****************
**************** attack surface #3: sol.transact ****************
Found a potential vulnerability at line 271, column 8 in
programs/zkcash/src/lib.rs
The account may not be properly validated and may be untrustful:
 265|    pub tree_token_account: Account<'info, TreeTokenAccount>,
 266|
 267|    #[account(mut)]
 268|    pub recipient: SystemAccount<'info>,
 269|
 270|    #[account(mut)]
>271|    pub fee_recipient_account: SystemAccount<'info>,
 272|
 273|    /// The authority account is the account that created the
tree and fee recipient PDAs
 274|    pub authority: SystemAccount<'info>,
 275|
 276|    /// The account that is signing the transaction
 277|    #[account(mut)]
>>>Stack Trace:
```

```
For more info, see
https://medium.com/coinmonks/the-wormhole-hack-how-soteria-detects-th
e-vulnerability-automatically-eb0f433e8071

Found a potential vulnerability at line 268, column 8 in
programs/zkcash/src/lib.rs
The account may not be properly validated and may be untrustful:
 262|        bump = tree_token_account.bump,
 263|        has_one = authority @ ErrorCode::Unauthorized
 264|    )]
 265|    pub tree_token_account: Account<'info, TreeTokenAccount>,
 266|
 267|    #[account(mut)]
>268|    pub recipient: SystemAccount<'info>,
 269|
 270|    #[account(mut)]
 271|    pub fee_recipient_account: SystemAccount<'info>,
 272|
 273|    /// The authority account is the account that created the
tree and fee recipient PDAs
 274|    pub authority: SystemAccount<'info>,
>>>Stack Trace:
For more info, see
https://medium.com/coinmonks/the-wormhole-hack-how-soteria-detects-th
e-vulnerability-automatically-eb0f433e8071

--------The summary of potential vulnerabilities in
programs_zkcash.ll--------

      2 untrustful account issues
```

- **Radar**

```
[ Medium ] Account Reinitialization found at:
 * privacy-cash/anchor/programs/zkcash/src/lib.rs:22:12-22

[ Low ] Missing Owner Check found at:
```

```
 * privacy-cash/anchor/programs/zkcash/src/lib.rs:183:3-9
```

- Account Reinitialization issue is flagged at line `pub fn initialize(ctx: Context<Initialize>)` because of the presence of the line `let tree_account = &mut ctx.accounts.tree_account.load_init()?;` so if the initialize function is re-invoked then this could allow reinitialization and state overwrite. **The authority should make sure that initialize is called only once**.
- Missing Owner check is for `Proof` which is a false positive.

- **Circomspect**

```
circomspect: analyzing template 'MerkleProof'
warning: Using `Num2Bits` to convert field elements to bits may lead
to aliasing issues.
   ┌─ privacy-cash/circuits/merkleProof.circom:18:27
   │
18 │     component indexBits = Num2Bits(levels);
   │                           ^^^^^^^^^^^^^^^^ Circomlib template
`Num2Bits` instantiated here.
   │
   = Consider using `Num2Bits_strict` if the input size may be >=
than the prime size.
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_pas
ses.md#non-strict-binary-conversion.

circomspect: 1 issue found.
```

```
circomspect: analyzing template 'Transaction'
warning: Intermediate signals should typically occur in at least two
separate constraints.
    ┌─
```

```
/Users/manishspc/HashCloak/audit/privacyCash/privacy-cash/circuits/transact
ion.circom:132:5
    |
132 |     signal extDataSquare <== extDataHash * extDataHash;
    |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |     |
    |     The intermediate signal `extDataSquare` is declared here.
    |     The intermediate signal `extDataSquare` is constrained here.
    |
    = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md
#under-constrained-signal.

circomspect: 1 issue found.
```

- All the warnings are redundant