



Auditing Report

Hardening Blockchain Security with Formal Methods

FOR



Privacy Cash



Veridise Inc.
November 10, 2025

► **Prepared For:**

Privacy Cash
www.privacycash.org

► **Prepared By:**

Alp Bassa
Evgeniy Shishkin
Tyler Diamond

► **Contact Us:**

contact@veridise.com

► **Version History:**

December 8, 2025	V3
December 3, 2025	V2
November 21, 2025	V1

Contents

Contents	iii
1 Executive Summary	1
2 Project Dashboard	3
3 Security Assessment Goals and Scope	4
3.1 Security Assessment Methodology	4
3.2 Identified Security Risks	4
3.3 Scope	4
3.4 Classification of Vulnerabilities	5
4 Security Review Assumptions	6
4.1 Operational Assumptions	6
5 Vulnerability Report	7
5.1 Detailed Description of Issues	8
5.1.1 V-PCS-VUL-001: Mint address is not correlated between Anchor program and circuit	8
5.1.2 V-PCS-VUL-002: Unnecessary UTXO cross-checks	9
5.1.3 V-PCS-VUL-003: Maintainability Improvements	11
6 Picus	13
6.1 Overview	13
6.2 Determinism	13
6.3 Methodology	13
6.4 Results Summary	13



From Nov. 10, 2025 to Nov. 14, 2025, Privacy Cash engaged Veridise to conduct a security assessment of the Privacy Cash protocol. The security assessment covered a Solana program that enables users to deposit funds and later privately withdraw them. Veridise conducted the assessment over 10 person-days, with 2 security analysts reviewing the project over 5 days on commit 4b2ec49. The security analysts also used Picus to prove the determinism of the circuits, see Chapter 6.

Project Summary. The Privacy Cash protocol enables users to anonymize funds denominated in either SOL or SPL tokens on Solana. It achieves this by following a simple workflow: Users deposit funds into the protocol from one address and later withdraw them to a different address. To withdraw previously deposited funds, users must prove to the protocol - via a zero-knowledge proof - that they actually own the funds they are attempting to withdraw. The proof contains all required information privately, ensuring that no sensitive data is revealed on-chain.

The protocol architecture consists of two main components:

1. **On-chain Solana program:** This program handles deposits of SOL/SPL and allows users to later withdraw their funds to any address, thereby providing anonymity. Both operations require a proof generated by the transaction circuit, which is verified on-chain using a Groth16 verifier. The program embeds a verification key tied to a specific version of the Transaction circuit.
2. **Off-chain Transaction circuit:** This Circom circuit verifies both input nullifiers (the publicly visible part of a spend) and output commitments. For each input nullifier, the circuit proves that the user knows the underlying UTXO whose hashed representation matches the given nullifier. All sensitive information - UTXO data, secrets, amounts - is supplied as private signals, ensuring it never appears on-chain. For output commitments, the circuit proves that the user correctly formed the new encrypted UTXOs and possesses the necessary information to later recover them.

The protocol also includes fee logic for deposits and withdrawals; however, only withdrawals are currently charged. Notably, the protocol does not enforce a fixed fee recipient address. While the DApp defaults to the protocol owner as the beneficiary, more advanced users can override this and redirect fees to their own address - effectively using the protocol at a minimal cost.

Code Assessment. The Privacy Cash developers provided the source code of the Privacy Cash contracts for the code review. The Circom source code is based on Tornado Nova with modifications by the Privacy Cash developers. Additionally, the Solana program appears to be mostly original code. It contains some documentation in the form of READMEs and documentation comments on functions and storage variables. To facilitate the Veridise security analysts' understanding of the code, the Privacy Cash developers met to discuss the different pieces of the protocol.

The source code included a test suite that the Veridise security analysts found to be thorough and covering most execution paths.

Summary of Issues Detected. The security assessment uncovered 3 issues, 1 of which was assessed to be a critical issue. Specifically, [V-PCS-VUL-001](#) allows an attacker to drain all funds from the shielded pool. The Veridise analysts also identified 2 warnings.

Privacy Cash has fixed the critical issue. They acknowledge the two warning issues and do not plan to fix them.

During the fix review, the analysts examined modifications in commits `4c293af`, `9dbf7e9`, `d1ca736`, `7a0b7cf`. The last commit `7a0b7cf` loosens the previous constraint that required the SPL token receiver to be an ATA, allowing it instead to be any token account tied to the specified mint.

It should be noted that, between the commit `4b2ec49` covered by this assessment and the latest reviewed commit `7a0b7cf`, several additional commits were introduced, the changes of which were not reviewed.

The reviewed changes were merged into the main branch at commit `b62c1e7`.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Table 2.1: Application Summary.

Name	Version	Type	Platform
Privacy Cash	4b2ec49	Anchor	Solana

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Nov. 10–Nov. 14, 2025	Manual & Tools	2	10 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	1	1	1
High-Severity Issues	0	0	0
Medium-Severity Issues	0	0	0
Low-Severity Issues	0	0	0
Warning-Severity Issues	2	2	0
Informational-Severity Issues	0	0	0
TOTAL	3	3	1

Table 2.4: Category Breakdown.

Name	Number
Maintainability	2
Data Validation	1



3.1 Security Assessment Methodology

The security assessment process consists of the following steps:

1. Gather an initial understanding of the protocol's business logic, users, and workflows by reviewing the provided documentation and consulting with the developers.
2. Identify all valuable assets in the protocol.
3. Identify the main workflows for managing these assets.
4. Identify the most significant security risks associated with these assets.
5. Systematically review the codebase for execution paths that could trigger the identified security risks, considering different assumptions.
6. Prioritize one finding over another by assigning a severity level to each.

To improve the depth and efficiency of the code review, analysts utilized the following tools:

- ▶ *Circuits Formal Verification*. The security analysts utilized Picus - a formal verification tool for Circom circuits developed by Veridise - to validate that the compiled circuit is deterministic.

3.2 Identified Security Risks

After the initial phase of the security assessment was completed, the security analysts generated a list of potential security risks. Security analysts used this list during the code review as a starting point to identify potential attack vectors. A few of these risks include the following:

- ▶ Are UTXOs correctly validated and protected against double spending?
- ▶ Are nullifiers deterministically derived from UTXOs and properly tracked?
- ▶ Can an attacker or any entity perform a DoS attack that prevents users from redeeming their funds?
- ▶ Are deposits and withdrawals handled correctly and guaranteed to maintain solvency?
- ▶ Is the fee accounting logic implemented correctly?
- ▶ Do all required Solana accounts correctly enforce their minimum balance, preventing it from being exceeded or violated?
- ▶ Can nullifiers be removed or invalidated in any unintended way?
- ▶ Are all Solana account descriptors constrained and validated as required?
- ▶ Is the Groth16 verification implemented correctly and performing all necessary checks?
- ▶ Does the Transaction circuit accurately implement the intended protocol logic?
- ▶ Is there any potential for leaking privacy-sensitive data?

3.3 Scope

The scope of this security assessment is limited to a specific set of source files from the repository, as agreed upon with the Privacy Cash developers:

- ▶ circuits/
 - keypair.circom
 - merkleProof.circom
 - transaction.circom
 - utils.rs
- ▶ anchor/programs/zkcash/src/
 - errors.rs
 - groth16.rs
 - lib.rs
 - merkle_tree.rs
 - utils.rs

3.4 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4

Security Review Assumptions

4.1 Operational Assumptions

In addition to assuming that any out-of-scope components behave correctly, Veridise analysts assumed the following properties held when modeling security for Privacy Cash.

- ▶ Even though the Solana program is upgradable, it will not be updated to a new version without a strong justification.
- ▶ The verification key of the ZK circuit is correctly provided in `utils.rs`.
- ▶ The ZK artifacts/circuits/TRUSTED_SETUP.MD document specifies that the trusted setup for the circuits was performed by 4 individuals.
- ▶ The protocol is fine with not receiving any fees, as the fee recipient account of transactions is set by users.
- ▶ The authority of the program will set a reasonable fee rate.
- ▶ During the fix review phase, the developers informed the analysts that they intended to use a Relayer component alongside the smart contract. Since this component had not been mentioned previously, its behavior was not considered or covered in this security review. Hence, the Relayer component is considered fully trusted and functionally correct.

Operational Recommendations. Highly-privileged operations should be operated by a multi-sig contract or decentralized governance system. These operations should be guarded by a timelock to ensure there is enough time for incident response. Highly-privileged, emergency operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- ▶ Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- ▶ Using separate keys for each separate function.
- ▶ Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- ▶ Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- ▶ Validating that no party has control over multiple multi-sig keys.
- ▶ Performing regularly scheduled key rotations for high-frequency operations.
- ▶ Securely storing physical, non-digital backups for critical keys.
- ▶ Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- ▶ Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5

Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-PCS-VUL-001	Mint address is not correlated between . . .	Critical	Fixed
V-PCS-VUL-002	Unnecessary UTXO cross-checks	Warning	Acknowledged
V-PCS-VUL-003	Maintainability Improvements	Warning	Acknowledged

5.1 Detailed Description of Issues

5.1.1 V-PCS-VUL-001: Mint address is not correlated between Anchor program and circuit

Severity	Critical	Commit	4b2ec49
Type	Data Validation	Status	Fixed
Location(s)	circuits/transaction.circom:28-29		
Confirmed Fix At	1863681		

Description The `mintAddress` inside the zk-circuit determines which token the input and output UTXOs correspond to. This allows the UTXO Merkle tree to support multiple tokens beyond just SOL. The `mintAddress` is kept private so that observers cannot infer which token is being spent when `transact()` is called.

However, during deposits and withdrawals, the zk-proof submitted with the call does **not** enforce that the spent or newly created UTXOs match the actual tokens provided to the protocol. Although `extDataHash` is computed on-chain and passed to the circuit as a public input, the circuit does not enforce any relationship between `extDataHash` and the private `mintAddress`. As a result, a user can deposit or withdraw one token while freely choosing an arbitrary `mintAddress` for the corresponding notes.

Impact An attacker can effectively steal the protocol's most valuable tokens. By depositing a low-value token and withdrawing SOL instead, they profit from the value difference. This process can be repeated as long as liquidity remains, allowing the attacker to systematically drain the higher-value asset from the protocol.

Recommendation Expose the `mintAddress` as a public signal so that the program can validate the correctness of the UTXOs.

Developer Response The developers now seed the merkle tree in the `transact_spl()` instruction with the `mint` address. This leads to a separate merkle tree being used for every token, and therefore there is a separation of token balances in the program.

5.1.2 V-PCS-VUL-002: Unnecessary UTXO cross-checks

Severity	Warning	Commit	4b2ec49
Type	Maintainability	Status	Acknowledged
Location(s)	anchor/programs/zkcash/src/lib.rs:505-537, 581-613		
Confirmed Fix At	N/A		

Description The Transact and TransactSpl structs used in the transact() and transact_spl() instructions define 4 nullifier accounts to represent the two nullifier produced by every transaction.

```
1 pub struct Transact<'info> {
2     // ...
3     #[account(
4         // ...
5         seeds = [b"nullifier0", proof.input_nullifiers[0].as_ref()],
6         bump
7     )]
8     pub nullifier0: Account<'info, NullifierAccount>,
9
10    #[account(
11        // ...
12        seeds = [b"nullifier1", proof.input_nullifiers[1].as_ref()],
13        bump
14    )]
15    pub nullifier1: Account<'info, NullifierAccount>,
16
17    #[account(
18        seeds = [b"nullifier0", proof.input_nullifiers[1].as_ref()],
19        bump
20    )]
21    pub nullifier2: SystemAccount<'info>,
22
23    #[account(
24        seeds = [b"nullifier1", proof.input_nullifiers[0].as_ref()],
25        bump
26    )]
27    pub nullifier3: SystemAccount<'info>,
28    // ...
29 }
```

Snippet 5.1: Snippet from anchor/programs/zkcash/src/lib.rs:Transact()

The first two accounts (nullifier0 and nullifier1) correspond to the two actual nullifiers in proof.input_nullifiers, each derived using its correct index. The additional accounts (nullifier2 and nullifier3) serve as cross-checks: they ensure that users cannot double-spend by simply swapping the order of the two nullifiers in proof.input_nullifiers.

Impact The scheme could be simplified by deriving each nullifier account from a single seed composed of the string "nullifier" concatenated with the nullifier bytes from proof.input_nullifiers. This approach would eliminate the need for the additional uniqueness checks in the circuit and the need to introduce two extra nullifier accounts.

Recommendation Use a single "nullifier" seed concatenated with the nullifier bytes for both nullifier0 and nullifier1, and remove the cross-check nullifier accounts.

Developers Response The current nullifier scheme was implemented as a mitigation for an earlier double-spend vulnerability. Consequently, applying the recommendation is not feasible at this stage.

5.1.3 V-PCS-VUL-003: Maintainability Improvements

Severity	Warning	Commit	4b2ec49
Type	Maintainability	Status	Acknowledged
Location(s)	▶ anchor/programs/zkcash/src/ • lib.rs • merkle_tree.rs • utils.rs ▶ circuits/transaction.circom		
Confirmed Fix At	N/A		

Description The security analysts identified several areas where clarity and maintainability can be improved:

lib.rs

- ▶ `initialize()`: Replace literal configuration values with well-named constants.
- ▶ `MerkleTreeAccount`: Introduce a named constant for the history length and use it instead of the hard-coded literal.
- ▶ `Transact` and `TransactSpl` structs: The `fee_recipient_account` / `fee_recipient_ata` fields can point to any account, allowing users to avoid paying fees. This behavior should be explicitly documented.
- ▶ `transact()`:
When transferring SOL during deposits, use the already-defined `deposit_amount` instead of reusing `ext_amount` as `u64`.

merkle_tree.rs

- ▶ `MerkleTree`: The `Hasher` trait should be anchored to the struct itself rather than requiring each method to specify a generic parameter.
- ▶ `append()`: The computed `max_capacity` value is currently off by one. Although the logic is correct, the naming is misleading. Subtracting 1 in the calculation and updating the `tree_account.next_index` check to use `<=` would make the intent clearer.

utils.rs

- ▶ `validate_fee()`: Introduce a constant to represent the percentage basis (currently 10000) and replace the literal with that constant.
- ▶ The two code paths conditioned on `ext_amount` duplicate much of the same logic. Consider refactoring by first determining the applicable fee rate and the absolute amount (including any required `checked_neg()`), then sharing the remaining logic outside the conditional.
- ▶ `calculate_complete_ext_data_hash_for_sol()`, `calculate_complete_ext_data_hash()`, and `calculate_complete_ext_data_hash_for_spl()`: These functions repeat the same logic and each defines `CompleteExtData` separately. Consolidate them into a single function and a single struct definition.

transaction.circom

- ▶ Add compile-time assertions ensuring that `nIns` and `nOuts` are both less than 16.

Impact The current implementation is more complex than necessary, which hurts maintainability and increases the risk of future bugs.

Recommendation It is recommended to implement the changes outlined above.

Developers Response The developers acknowledged the issue but decided not to implement changes at this point.

6.1 Overview

This section describes how the Veridise analysts used Picus to check the determinism of the Privacy Cash circuits.

6.2 Determinism

Given a circuit $C(I, O)$ with inputs signals I and output signals O , a circuit C is deterministic if and only if:

$$\forall I, O, O'. C(I, O) \wedge C(I, O') \rightarrow O \equiv O'$$

In essence, a deterministic circuit encodes a function (or partial function) from I to O . Determinism is an important property that most ZK Circuits should satisfy since most circuits are intended to encode some deterministic computation.

6.3 Methodology

Picus takes as input a circuit written in its custom circuit language called PCL. Since the Privacy Cash circuits were written in Circom, the Veridise analysts used the built-in transpiler to convert the circuits written in Circom to PCL.

6.4 Results Summary

Picus successfully proved the determinism of the `transaction2.circom` circuit, which encompasses all in-scope Circom code.