Privacy Cash - Privacy Cash SDK Audit Report

Version 1.1

Zigtur

October 3, 2025

Privacy Cash - Privacy Cash SDK - Audit Report

Zigtur

October 03, 2025

Prepared by: Zigtur

Table of Contents

- INFO-03 - Duplicated logic to filter UTXOs
- INFO-04 - SDK does not allow multiple non-empty UTXOs per transfer
- INFO-05 - SDK does not allow multiple mint addresses
- INFO-06 - Nullifier PDA calculation functions are not in the same module
- INFO-07 - Adding field size before modulo is useless
- INFO-08 - SDK does not allow UTXO transfer

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 - c971b6923d4b4a766df84a3b46d18c801956bbd1

Fixes review commit hash - f9a8c7465dfbfbc1f0cc1150c621805553aa828e

Deployment chains

- Solana

Scope

The following code is in scope of the review:

- src/*
- circuit2/*

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 - Fee amount and rates are hardcoded in SDK

Scope:

- deposit.ts#L63
- constants.ts#L20-L22

Description

The SDK hardcodes fee amount and rates directly in the source code rather than retrieving them dynamically from the on-chain global configuration. For deposits, the fee is set to zero, while withdraw fees are calculated using hardcoded constants.

```
// In deposit.ts
const fee_amount_in_lamports = 0; // Hardcoded fee for deposits

// In constants.ts
export const WITHDRAW_FEE_RATE = 35 / 10000;

export const WITHDRAW_RENT_FEE = 0.006; // sol
```

This approach creates several issues. If the protocol implements deposit fees in the future or modifies withdraw fee rates, the SDK will become incompatible without code updates. Users may experience transaction failures when the hardcoded values don't match the on-chain configuration. The inflexibility also prevents dynamic fee adjustments.

Recommendation

Replace hardcoded fee constants with dynamic retrieval from the global configuration account. Implement a fee fetching mechanism that queries the on-chain GlobalConfig account before constructing transactions.

This ensures the SDK always uses current fee rates and maintains compatibility with protocol updates.

PrivacyCash

Fixed in PR10.

Zigtur

Fixed. The configuration is retrieved from the indexer.

LOW-01 - Blinding factor random range is small

Scope:

- utxo.ts#L37

Description

The UTXO class generates blinding factors using JavaScript's `Math.random()` function, which provides only 53 bits of entropy due to JavaScript's number precision limitations. This reduced entropy space makes brute force attacks more feasible than intended.

Moreover, the range is further reduced to be between `0` and `1_000_000`.

```
// In utxo.ts
blinding = new BN(Math.floor(Math.random() * 1000000000)),
```

The blinding factor is critical for UTXO privacy as it ensures that even identical amounts produce different commitments. With only 53 bits of entropy instead of the full 254-bit field size, an attacker could potentially enumerate all possible blinding factors to link UTXOs or break privacy guarantees. The limited entropy also increases the probability of blinding factor collisions, which could lead to identical commitments for different UTXOs.

Recommendation

Replace `Math.random()` with a cryptographically secure random number generator that provides full field-size entropy. Use `crypto.getRandomValues()` in browsers or `crypto.randomBytes()` in Node.js to generate the required 32 bytes of random data.

PrivacyCash

Acknowledged. "If the public key stored in the encrypted notes is never revealed, it would act as a sort of blinding factor". Also even in the worst case scenario, user funds are still safe.

We will probably add more bytes to the blinding factor after we save more tx bytes in future optimizations.

Zigtur

Acknowledged. As long as the pubkey used inside of the zk proof remains private (which is the case for Privacy Cash current codebase), it acts as a sort of blinding factor so it is good for the actual blinding factor to be weaker.

It is still recommended to get a greater range for the blinding factor.

LOW-02 - Local storage key collision could occur

Scope:

- getUtxos.ts#L42-L44

Description

The local storage key generation function creates keys using only the first 8 characters of the base58-encoded public key, significantly increasing the probability of collisions between different users.

```
export function localstorageKey(key: PublicKey) {
    return PROGRAM_ID.toString().substring(0, 6) + key.toString().substring(0, 6)
    ↪  // @audit only 6 characters from key
}
```

With base58 encoding having 58 possible characters per position, the collision probability is substantial with only 6 characters. This creates a scenario where a user with multiple keys could share the same local storage key, leading to data corruption where one address encrypted outputs could overwrite another's.

Users might see incorrect UTXO data or experience transaction failures due to cached data belonging to different accounts.

Note: this can be particularly true as vanity address are common. Even Solana docs indicate how to do it in the docs.

Recommendation

Use the full public key string to generate unique local storage keys.

PrivacyCash

Fixed in PR11.

Zigtur

Fixed. The full public key is now used.

LOW-03 - Privacy leakage through round start index correlation

Scope:

- getUtxos.ts#L69-L81

Description

The UTXO fetching mechanism uses a round-based pagination system that reveals timing information about when users first interacted with the protocol. The `roundStartIndex` is calculated based on the user's public key and used to determine which UTXOs to fetch.

```
let roundStartIndex = publicKey.toString().split('').reduce((acc, char) => {
    return acc + char.charCodeAt(0);
}, 0) % 20000;

if (roundStartIndex === 0) {
    roundStartIndex = fetch_utxo_offset;
}
```

This deterministic calculation means that an observer can correlate a user's public key with their round start index, potentially revealing when the user deposited into the protocol. The predictable nature of this index allows external parties to track user activity patterns and estimate account creation times, reducing the privacy guarantees of the protocol.

Recommendation

Implement a more privacy-preserving pagination mechanism that doesn't leak timing information. Consider implementing a full-scan approach that doesn't rely on predictable indices. Note that this may lead to performance issues.

PrivacyCash

Acknowledged and won't fix. Trading off for much better UX (users wait way less time to get the full balance).

Zigtur

Acknowledged.

LOW-04 - Privacy leakage through commitment and nullifier account accesses

Scope:

- getUtxos.ts#L355-L390
- getUtxos.ts#L234
- getUtxos.ts#L247

Description

The SDK queries commitment and nullifier accounts directly on-chain to retrieve real index values and check UTXO spent status. These queries create observable patterns that can be monitored by blockchain observers to correlate user activity.

```
// Querying commitment accounts reveals which UTXOs belong to the user
const [commitment0PDA] = PublicKey.findProgramAddressSync(
    [Buffer.from("commitment0"), Buffer.from(commitmentBytes)],
    PROGRAM_ID
);
const account0Info = await connection.getAccountInfo(commitment0PDA);

// Checking nullifier accounts reveals when UTXOs are being spent
const nullifierPDA = PublicKey.findProgramAddressSync(
    [Buffer.from("nullifier"), nullifierBuffer],
    PROGRAM_ID
);
```

These account queries leave traces that can be analyzed to build user profiles and transaction graphs. An observer monitoring RPC calls could correlate commitment queries with specific users, breaking the privacy assumptions of the protocol. The timing and frequency of these queries also reveal user behavior patterns.

Recommendation

Users should be aware that the RPC server they are using can correlate data to leak privacy informations.

PrivacyCash

Acknowledged and won't fix. Odds of RPC nodes watching those are super low, especially frontend is using reputable RPC nodes via Helius. It's also highly recommended to SDK users to use reputable RPC node providers.

Zigtur

Acknowledged.

LOW-05 - Privacy leakage through merkle proof generation

Scope:

- utils.ts#L122-L128

Description

The SDK requests merkle proofs from an external indexer service, revealing which specific UTXOs a user intends to spend. This creates a direct correlation between user requests and their transaction intentions.

```
export async function generateMerkleProof(utxo: Utxo, lightWasm: hasher.LightWasm)
↪  {
    const commitment = await utxo.getCommitment();
    logger.debug('generating merkle proof for commitment: ', commitment);
    const url = `${INDEXER_URL}/merkle-proof/${commitment}`;

    const res = await fetch(url);
    // The indexer now knows this user is about to spend this specific UTXO
}
```

The indexer service can monitor incoming merkle proof requests to build comprehensive profiles of user spending patterns. By correlating proof requests with subsequent on-chain transactions, the indexer can definitively link users to their transactions, completely breaking the privacy model. The timing between proof generation and transaction submission also reveals user behavior patterns.

Recommendation

The merkle proof generation should be done locally. The indexer should only provide the full merkle tree data.

This would ensure that the indexer can't correlate IP address with commitment spent.

PrivacyCash

Acknowledged and won't fix. Trading off for way quicker merkle tree build time and proof generation time. Relayers are highly reputational.

Zigtur

Acknowledged.

INFO-01 - Typographical issues

Scope:

- constants.ts#L32
- getUtxos.ts#L129

Description

- At constants.ts#L32, `LSK_ENCRPTED_OUTPUTS` should be `LSK_ENCRYPTED_OUTPUTS`.
- At getUtxos.ts#L129, `hashMore` should be `hasMore`.

PrivacyCash

Fixed in PR12.

Zigtur

Fixed.

INFO-02 - Encryption mechanism does not have forward secrecy

Description

The UTXO encryption system derives keys deterministically from the user's wallet keypair, meaning the same encryption key is used consistently across all UTXOs for a given user. This design lacks forward secrecy, where compromise of current keys should not affect the security of past communications.

```
// The same key is derived every time for the same wallet
const utxoPrivateKey = encryptionService.deriveUtxoPrivateKey();
const encryptionKey = keccak256(signature); // Always the same for the same wallet
```

If a user's wallet private key is compromised or the signature used for key generation leaks, an attacker can decrypt all historical UTXOs associated with that wallet.

This violates the principle of forward secrecy where past encrypted data should remain secure even after key compromise. The deterministic key derivation also means that all UTXOs for a user can be linked cryptographically.

Recommendation

Consider incorporating time-based or transaction-based key derivation where each UTXO uses a unique encryption key that cannot be derived from future compromised keys.

This could involve using a hash chain or incorporating additional entropy sources to ensure that compromise of current keys does not affect the confidentiality of historical UTXOs.

PrivacyCash

Acknowledged and won't fix. We use the same mechanism as Tornado Cash Nova. Trading off for way better UX that users don't need to remember a different note for different txs.

Zigtur

Acknowledged.

INFO-03 - Duplicated logic to filter UTXOs

Scope:

- getUtxos.ts#L82-L105
- deposit.ts#L97-L116
- withdraw.ts#L82-L98

Description

The `getUtxos` function returns the user UTXOs that are not yet spent and non zero amount.

However, both `deposit` and `withdraw` call `getUtxos` but re-execute the filtering to exclude zero amount and spent UTXOs. This logic is duplicated.

```
// ...
// Fetch existing UTXOs for this user
logger.debug('\nFetching existing UTXOs...');
const allUtxos = await getUtxos({ connection, publicKey, encryptionService });
↪  // @audit getUtxos filter out the zero or spent utxos
logger.debug(`Found ${allUtxos.length} total UTXOs`);

// Filter out zero-amount UTXOs (dummy UTXOs that can't be spent)
const nonZeroUtxos = allUtxos.filter(utxo => utxo.amount.gt(new BN(0)));
logger.debug(`Found ${nonZeroUtxos.length} non-zero UTXOs`);

// Check which non-zero UTXOs are unspent (sequentially to avoid rate limiting)
logger.debug('Checking which UTXOs are unspent...');
const utxoSpentStatuses: boolean[] = [];
for (let i = 0; i < nonZeroUtxos.length; i++) {
    const isSpent = await isUtxoSpent(connection, nonZeroUtxos[i]);
    utxoSpentStatuses.push(isSpent);

    // Add a longer delay between checks to prevent rate limiting
    if (i < nonZeroUtxos.length - 1) {
        await new Promise(resolve => setTimeout(resolve, 1000)); // 1 second
        ↪  delay
    }
}
```

Recommendation

The filter logic can be removed from `withdraw` and `deposit` as this is executed in `getUtxos`.

PrivacyCash

Fixed in PR14.

Zigtur

Fixed. The duplicated logic has been removed from `deposit` and `withdraw`.

INFO-04 - SDK does not allow multiple non-empty UTXOs per transfer

Scope:

- deposit.ts#L237-L243
- withdraw.ts#L155-L169

Description

The current SDK implementation automatically merges user UTXOs into a single UTXO during deposit and withdraw operations, preventing users from maintaining multiple separate UTXOs with different amounts. The deposit and withdraw logics consolidate existing UTXOs rather than preserving them as distinct outputs.

```
    const outputs = [
        new Utxo({
            lightWasm,
            amount: outputAmount,
            keypair: utxoKeypair,
            index: currentNextIndex // This UTXO will be inserted at
            ↪  currentNextIndex
        }), // Output with value (either deposit amount minus fee, or input amount
        ↪  minus fee)
        new Utxo({
            lightWasm,
            amount: '0',
            keypair: utxoKeypair,
            index: currentNextIndex + 1 // This UTXO will be inserted at
            ↪  currentNextIndex + 1
        }) // Empty UTXO
    ];
```

This design limits the protocol's flexibility and privacy features. Users cannot maintain multiple UTXOs for different purposes or amounts, which reduces the effectiveness of mixing and makes transaction patterns more predictable. The lack of UTXO diversity also limits the protocol's resistance to amount-based transaction analysis.

Recommendation

Implement support for multiple UTXO management in the SDK where users can choose to keep separate UTXOs instead of always merging them. Add configuration options that allow users to specify whether they want to consolidate UTXOs or maintain them separately. This would enable better privacy through diverse UTXO sets and provide users with more control over their privacy strategy.

PrivacyCash

Acknowledged. Support for multiple non zero output utxo is not a priority, and currently not needed for frontend/SDK.

Zigtur

Acknowledged.

INFO-05 - SDK does not allow multiple mint addresses

Scope:

- deposit.ts#L324-L325

Description

The SDK does not currently support multiple `mintAddress` for UTXOs.

This SDK will require code modifications when multiple mint address are supported by the program.

Recommendation

Consider implementing the logic for multiple mint address.

PrivacyCash

Acknowledged. Non-SOL mint address will be supported after the protocol layer supports SPL tokens.

Zigtur

Acknowledged.

INFO-06 - Nullifier PDA calculation functions are not in the same module

Scope:

- utils.ts#L139-L151
- encryption.ts#L408-L420

Description

The codebase contains duplicate nullifier PDA calculation functions in different modules, creating inconsistency and potential maintenance issues. Both `utils.ts` and `encryption.ts` implement similar functionality for computing nullifier-based Program Derived Addresses.

```
// In utils.ts
export function findNullifierPDAs(proof: any) {

// In encryption.ts
export function findCrossCheckNullifierPDAs(proof: any) {
```

These two functions show similar logic but are in two different modules.

Recommendation

Consider moving `findCrossCheckNullifierPDAs` to `utils.rs`.

PrivacyCash

Fixed in PR13.

Zigtur

Fixed.

INFO-07 - Adding field size before modulo is useless

Scope:

- deposit.ts#L226
- withdraw.ts#L173

Description

The code contains unnecessary addition of `FIELD_SIZE` before applying modulo operations when computing external amounts for the zero-knowledge proofs.

```
// In both deposit.ts and withdraw.ts
new BN(extAmount).sub(new BN(fee_in_lamports)).add(FIELD_SIZE).mod(FIELD_SIZE);
```

Adding the field size before taking modulo is mathematically redundant because `(a + n) mod n = a mod n` for any values where `a >= 0`. Since the external amount minus fees should always be non-negative in valid transactions, the addition of `FIELD_SIZE` serves no purpose and only adds computational overhead. This pattern suggests a misunderstanding of modular arithmetic or defensive programming that is unnecessarily complex.

Recommendation

Remove the unnecessary `.add(FIELD_SIZE)` operation from both locations. The code should simply use `new BN(extAmount).sub(new BN(fee_in_lamports)).mod(FIELD_SIZE)` which produces the same result with better performance.

PrivacyCash

Acknowledged. Not much down side to keep it here, given it's the same math code as Tornado.

Zigtur

Acknowledged.

INFO-08 - SDK does not allow UTXO transfer

Scope:

- index.ts

Description

One of the powerful features of zero-knowledge UTXO systems is the ability to create UTXOs with specific private keys that can be shared with other users for direct private transfers. However, the current SDK implementation does not support this functionality.

```
// All UTXOs use the same derived private key for a user
const utxoPrivateKey = encryptionService.deriveUtxoPrivateKey();
const utxoKeypair = new UtxoKeypair(utxoPrivateKey, lightWasm);
```

The SDK currently derives all UTXO private keys deterministically from the user's wallet keypair, meaning every UTXO for a given user uses the same private key. This prevents the creation of UTXOs with custom private keys that could be shared with other parties for direct transfers. This limitation reduces the protocol's flexibility and prevents certain privacy-enhancing use cases like creating UTXOs specifically intended for transfer to other users without going through the deposit/withdraw cycle.

Recommendation

Add support for custom UTXO private key generation in the SDK. Allow users to specify custom private keys when creating UTXOs, enabling direct UTXO transfers between parties.

This feature would enhance the protocol's utility by supporting more sophisticated privacy patterns and reducing the need for intermediate on-chain transactions for certain transfer scenarios.

PrivacyCash

Acknowledged. That feature won't be in the near roadmap.

Zigtur

Acknowledged.