BLOCKSEC

Security Audit

Report for atoshi-privacy-sdk

Date: July 31, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Atoshi
Target	atoshi-privacy-sdk

Version History

Version	Date	Description
1.0	July 31, 2026	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-08-17 03:22:13+0000

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Infrastructure
Language	TypeScript
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of atoshi-privacy-sdk of Atoshi.

The atoshi-privacy-sdk project is built as a TypeScript client library for shielded transactions on Atoshi Chain. This project introduces the off-chain components needed to hold user secrets, build notes, and produce the proofs the privacy pool consumes. 1. Key Derivation: Turns an EIP-712 signature, mnemonic, or passphrase into a master seed and derives the spending, viewing, and backup-encryption keys from it. 2. Wallet: Manages the Poseidon keypair, the note store and its lifecycle, nullifier computation, and AES-256-GCM encrypted backup and restore. 3. Note: Defines the note structure and commitment, blinding generation, serialization, and per-token balance tracking. 4. Transaction Builder: Assembles deposit, withdraw, and transfer flows

Note this audit only focuses on the following files and directories:

- `src/wallet/derivation.ts`
- `src/wallet/index.ts`
- `src/note`
- `src/tx`

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
atoshi-privacy-sdk	`Version 1`	`a530a739209c9c0030b100acc6e013c59dfa91d7`
	`Version 2`	`70881c2f3d23dd838500ef4c7513f0f59fa1fb2c`

[1] https://github.com/ATOSHI-ORG/atoshi-privacy-sdk

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation

- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **three** potential security issues. Besides, we have **five** notes.

- Medium Risk: 3
- Note: 5

ID	Severity	Description	Category	Status
1	Medium	Lack of import path for scanned notes	Security Issue	Fixed
2	Medium	Lack of check on the recipient in function `deposit()`	Security Issue	Fixed
3	Medium	Lack of relayer forwarding when submitting withdrawals	Security Issue	Fixed
4	-	Management of sensitive wallet data	Note	-
5	-	Operational assumptions of relayer services	Note	-
6	-	Fixed EIP-712 payload for key deviation and recovery	Note	-
7	-	Trust assumptions of integration layer	Note	-
8	-	Correctness of the refactored merkle module is assumed	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of import path for scanned notes

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Class `TransactionBuilder` tracks notes created or consumed through local SDK operations, while `ChainScanner` recovers the user's notes from on-chain events. However, the SDK does not provide a mechanism to import the recovered notes into the wallet's local state. Consequently, notes found during wallet recovery or rescanning may be excluded from the displayed balance and unavailable for subsequent transfers or withdrawals.

```
61export class TransactionBuilder {
```

Listing 2.1: src/tx/index.ts

```
92  async scanForViewer(
93    viewingKey: bigint,
94    spendingKey: bigint,
95    toBlock?: number
96  ): Promise<RecoveredNote[]> {
97    const end = toBlock ?? (await this.getLatestBlock());
98    const recovered: RecoveredNote[] = [];
```

Listing 2.2: src/scanner/index.ts

Impact The wallet may display an incomplete balance and omit spendable notes recovered from on-chain events.

Suggestion Add a function to import the notes to the local records.

2.1.2 Lack of check on the recipient in function `deposit()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `src/tx/index.ts`, the function `deposit()` adds the newly created note to the local wallet after the deposit succeeds. However, the function does not confirm that the note owner matches the current wallet before updating the local records. When a user deposits funds to another recipient, the sender's wallet records the recipient's note as locally owned even though the sender cannot spend it.

```
208   // Notify privacy node
209   const nodeResult = await this.rpcClient.submitDeposit(
210     commitment,
211     params.tokenAddress,
212     amount,
213     receipt.hash
214   );
215
216   // The deposit is already on-chain, so always track the note -otherwise a
217   // node that hasn't returned a leafIndex yet would drop it from the wallet
218   // entirely (audit Issue 11 scenario 1). It stays Pending until the leafIndex
219   // is known (from the node here, or later via a chain scan), then upgrades
220   // to Committed.
221   this.wallet.addNote(note);
222   if (nodeResult.success && nodeResult.leafIndex !== undefined) {
223     note.setLeafIndex(nodeResult.leafIndex);
224     this.wallet.markNoteCommitted(
225       commitment,
226       nodeResult.leafIndex,
227       receipt.hash,
228       receipt.blockNumber
229     );
230   }
```

Listing 2.3: src/tx/index.ts

Impact The wallet may report an inflated balance and select an unspendable note for a subsequent transaction.

Suggestion Add the note to the local wallet only when its owner matches the current wallet's public key.

2.1.3 Lack of relayer forwarding when submitting withdrawals

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `src/tx/index.ts`, the interface `WithdrawParams` allows the caller to specify a `relayer` address. However, the function `withdraw()` does not forward this parameter when invoking `submitWithdraw()` to submit withdrawals to the RPC client. As a result, the specified relayer is ignored, and the withdrawal may be submitted through a different relayer.

```
112export interface WithdrawParams {
113  noteIndex: number;
114  recipient: string;
115  relayer?: string;
116  fee?: BigNumberish;
117}
```

Listing 2.4: src/types/index.ts

```
303   // Submit to privacy node
304   const result = await this.rpcClient.submitWithdraw(
305     proof,
306     merkleProof.root,
307     nullifier,
308     params.recipient,
309     note.tokenId === 0n ? ethers.ZeroAddress : toHex(note.tokenId, 40),
310     note.amount,
311     params.fee ? BigInt(params.fee.toString()) : 0n
312   );
```

Listing 2.5: src/tx/index.ts

Impact The withdrawal may use an unintended relayer, resulting in incorrect fee allocation and transaction routing.

Suggestion Forward the caller-specified relayer to function `submitWithdraw()` and include it in the withdrawal request.

2.2 Note

2.2.1 Management of sensitive wallet data

Introduced by `Version 1`

Description Under the integration model, the host application is responsible for encrypting, storing, and controlling access to sensitive wallet data exposed by the SDK. This includes protecting key material, plaintext note data, and wallet backups from unintended access by other application components or third-party scripts. Integrators should be aware that the SDK's sensitive interfaces may be further restricted or revised in a future version.

```
259 /**
260  * Get current keypair (includes the private key).
261  *
262  * @internal Exposes secret key material; used across SDK modules, not part
263  * of the public API. Consumers should use getPublicKey()/getViewingPubKey().
264  */
265 getKeypair(): Keypair | null {
266   // Return a copy so callers cannot mutate the wallet's keypair (audit Issue 16).
267   return this.keypair ? { ...this.keypair } : null;
268 }
```

Listing 2.6: src/wallet/index.ts

Feedback from the project The project states that the integration layer securely encrypts and manages sensitive data, and that the SDK interfaces will be revised in a future version.

2.2.2 Operational assumptions of relayer services

Introduced by Version 1

Description The privacy and availability of shielded transactions depend on the relayer service. Relayers are expected to enforce effective rate limits, use private transaction channels to reduce information leakage, and appropriately handle failed or pending submissions.

2.2.3 Fixed EIP-712 payload for key deviation and recovery

Introduced by Version 1

Description The EIP-712 typed-data payload used to derive privacy keys is treated as an immutable protocol-level constant. This ensures SDK upgrades never change the signature-derived keys or prevent recovery of an existing privacy wallet. Since the resulting signature is equivalent to long-term privacy key material, the host application is expected to display explicit anti-phishing guidance and verify that signing occurs only on an official project domain.

```
56// Empty chainId means the signature is portable across L1 / L2 -the
57// note-decrypt key is the SAME no matter which chain the user is on.
58export const SEED_DERIVATION_TYPED_DATA = {
59 domain: {
60   name: DOMAIN_NAME,
61   version: DOMAIN_VERSION,
62 },
63 types: {
64   AtoshiPrivacyKeyDerivation: [
65     { name: "purpose", type: "string" },
66     { name: "version", type: "uint256" },
67   ],
68 },
69 primaryType: "AtoshiPrivacyKeyDerivation",
70 message: {
71   purpose:
72     "Sign this message ONLY on the official Atoshi DApp to derive your privacy keys. " +
```

```
73     "DO NOT sign this on any other site -the signer can decrypt your notes.",
74   version: 1n,
75 },
76};
```

Listing 2.7: src/wallet/derivation.ts

```
160export async function seedFromEIP712Signature(signature: string): Promise<Seed> {
161  const sig = getBytes(signature);
162  if (sig.length !== 65) {
163    throw new Error(
164      `expected 65-byte EIP-712 signature, got ${sig.length} bytes`,
165    );
166  }
167  // Canonicalize to low-s so high-s / low-s equivalents map to one seed.
168  const sBig = bytesToBig(sig.slice(32, 64));
169  const sNorm = sBig > SECP256K1_HALF_N ? SECP256K1_N - sBig : sBig;
170  const sigNoV = new Uint8Array(64);
171  sigNoV.set(sig.slice(0, 32), 0);
172  sigNoV.set(bigTo32Bytes(sNorm), 32);
173  const hash = await crypto.subtle.digest("SHA-256", sigNoV);
174  return new Uint8Array(hash);
175}
```

Listing 2.8: src/wallet/derivation.ts

Feedback from the project The project states that the EIP-712 signing payload will remain unchanged and that the host application displays anti-phishing guidance and validates the official domain.

2.2.4 Trust assumptions of integration layer

Introduced by Version 1

Description Certain validation and error-handling responsibilities are delegated to the SDK's integration layer. The integration layer is expected to validate basic user-controlled inputs before invoking security-sensitive SDK functions, handle errors returned or thrown by SDK operations, and prevent failed operations from leaving user-visible state inconsistent.

Feedback from the project The project states that the integration layer performs the required baseline input validation and error handling.

2.2.5 Correctness of the refactored merkle module is assumed

Introduced by Version 2

Description The `merkle` module was refactored in Version 2, introducing logic changes to tree reconstruction, note reconciliation, and leaf-index handling. This review treats the refactored `merkle` module as a trusted component and assumes that its correctness. A full independent verification of the refactored `merkle` implementation is outside the scope of this audit.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS