BLOCKSEC

Security Audit

Report for zcash-wallet-wasm

Date: June 9, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	NoirWallet
Target	zcash-wallet-wasm

Version History

Version	Date	Description
1.0	June 9, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-06-10 02:59:32+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

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

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of zcash-wallet-wasm of NoirWallet.

The zcash-wallet-wasm project is a Rust/Wasm wallet SDK designed for browser execution. It is built on librustzcash, compiled with wasm-bindgen, and exposed through a single WebWallet JavaScript API. The audit scope covers two layers of code built on top of the vendored upstream library. The first is the new Wasm binding layer under src, which provides JavaScript-facing wallet operations including mnemonic and key derivation, account creation, address export, the full PCZT flow (create, sign, prove, send), transparent funds shielding, message signing, sync orchestration, and serialized in-memory database import and export. The second layer consists of a small set of project-specific modifications to librustzcash for the wallet sync and transparent-input paths, covering transparent address lookup for balance queries, transparent UTXO discovery, and incremental batch syncing. Note that for the imported librustzcash library, only the code differences relative to the commit c3425f9 (tagged zcash_primitives-0.26.4) [2] are in scope for this audit. Accordingly, this audit covers only the modified code differences in the listed librustzcash files, rather than the full contents of those files. Version 2 also introduces certain new features and functional changes. Such newly introduced functionality is outside the scope of this audit and has not been separately assessed.

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

- src
- librustzcash/zcash_client_backend/src/sync.rs
- librustzcash/zcash_client_memory/src/types/account.rs
- librustzcash/zcash_client_memory/src/types/memory_wallet/mod.rs
- librustzcash/zcash_client_memory/src/types/transparent.rs
- librustzcash/zcash_client_memory/src/wallet_read.rs
- librustzcash/zcash_client_memory/src/wallet_write.rs

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

[1] https://github.com/NoirWallet/zcash-wallet-wasm

[2] https://github.com/zcash/librustzcash/tree/zcash_primitives-0.26.4

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
zcash-wallet-wasm	Version 1	4511c41c1cb0f963e739f493df7843d821f7af6d
	Version 2	6493a7cab79c44afdb2014eef7036692cb955d87
	Version 3	e24eae7fc72e367be0bd99adafc05b5ce006645c

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 [3] and Common Weakness Enumeration [4]. 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.

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.

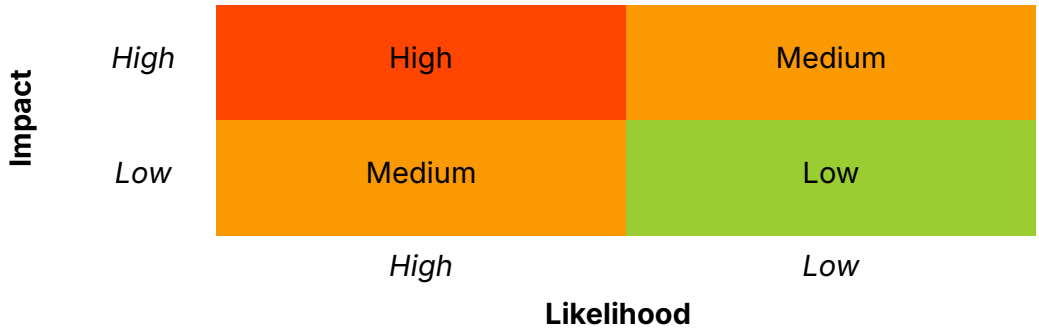
Furthermore, the status of a discovered item will fall into one of the following five categories:

- **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.

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

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

Table 1.1: Vulnerability Severity Classification



- **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 **twelve** potential security issues. Besides, we have **five** recommendations and **ten** notes.

- High Risk: 4
- Medium Risk: 3
- Low Risk: 5
- Recommendation: 5
- Note: 10

ID	Severity	Description	Category	Status
1	High	Data leakage to the remote proving server	Security Issue	Fixed
2	High	Sensitive data exposure through console logs	Security Issue	Fixed
3	High	Lack of Sapling receiver in returned unified address	Security Issue	Fixed
4	High	Incorrect implementation of function `utxo_query_height()`	Security Issue	Fixed
5	Medium	Incorrect transparent address resolution	Security Issue	Fixed
6	Medium	Lack of restriction on new mnemonic length	Security Issue	Fixed
7	Medium	Incorrect check in function `get_wallet_summary()`	Security Issue	Fixed
8	Low	Lack of support for BIP39 passphrases	Security Issue	Fixed
9	Low	Potential panics	Security Issue	Fixed
10	Low	Lack of handling for broadcast failures	Security Issue	Confirmed
11	Low	Lack of zeroization for sensitive `seed` value	Security Issue	Fixed
12	Low	Incorrect balance display during wallet recovery	Security Issue	Confirmed
13	-	Revise the incorrect annotation	Recommendation	Fixed
14	-	Add a check in function `pczt_sign_inner()`	Recommendation	Fixed
15	-	Unify implementation between payment request and PCZT creation	Recommendation	Fixed
16	-	Unify the confirmation policy	Recommendation	Fixed
17	-	Report pending and total balances	Recommendation	Fixed
18	-	Security assumptions on the host application	Note	-
19	-	Trust assumptions on the librustzcash library	Note	-

20	-	Non-standard encoding in transparent message signing	Note	-
21	-	Transfers to TEX addresses are not supported	Note	-
22	-	Address linkage during shielding	Note	-
23	-	Constraints of PCZT implementation	Note	-
24	-	Worker-based execution for multi-threaded Wasm	Note	-
25	-	Library upgrades and integration are out of scope	Note	-
26	-	Transparent receive transactions are omitted from history	Note	-
27	-	Function `get_next_shielded_address()` is limited to shielded address rotation	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Data leakage to the remote proving server

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `src/bindgen/wallet.rs`, the function `sign_prove_and_send_pczt()` uploads the fully signed PCZT to a remote proving server when the parameter `use_prove_server` is set to true. The uploaded PCZT is transmitted without redaction, exposing sensitive fields such as `dummy_ask` values, derivation data, and user metadata to the server.

```
629   async fn sign_prove_and_send_pczt(
630       &self,
631       pczt: pczt::Pczt,
632       usk: super::UnifiedSpendingKey,
633       seed_fp: SeedFingerprint,
634       sapling_proof_gen_key: Option<ProofGenerationKey>,
635       use_prove_server: Option<bool>,
636   ) -> Result<String, Error> {
637       let proof_gen_key = sapling_proof_gen_key.map(Into::into);
638
639       // Sign the PCZT
640       let signed_pczt =
641           pczt_sign_inner(self.inner.network, pczt, usk.into(), seed_fp.into()).await?;
642
643       // Prove the PCZT (remote or local)
644       let proved_pczt = if use_prove_server.unwrap_or(false) {
645           let signed_pczt_string = BASE64_STANDARD.encode(signed_pczt.serialize());
```

```
646         match send_pczt_to_prove_server(signed_pczt_string).await {
647             Ok(ProveResponse { proved_pczt }) => {
648                 let proved_pczt_bytes = BASE64_STANDARD.decode(&proved_pczt).unwrap();
649                 pczt::Pczt::parse(&proved_pczt_bytes).unwrap()
650             }
651             Err(e) => {
652                 tracing::error!("send_pczt_to_prove_server failed: {e}, try inner.pczt_prove.");
653                 self.inner.pczt_prove(signed_pczt, proof_gen_key).await?
654             }
655         }
656     } else {
657         self.inner.pczt_prove(signed_pczt, proof_gen_key).await?
658     };
```

Listing 2.1: src/bindgen/wallet.rs

```
200 for (_, spends) in keys {
201     // let usk = UnifiedSpendingKey::from_seed(&params, seed, account_index)?;
202     for keyref in spends {
203         match keyref {
204             KeyRef::Orchard { index } => {
205                 signer
206                     .sign_orchard(
207                         index,
208                         &orchard::keys::SpendAuthorizingKey::from(usk.orchard()),
209                     )
210                     .map_err(|e| {
211                         Error::PcztSign(format!(
212                             "Failed to sign Orchard spend {index}: {:?}",
213                             e
214                         ))
215                     })?;
216             }
217             KeyRef::Sapling { index } => {
218                 signer
219                     .sign_sapling(index, &usk.sapling().expsk.ask)
220                     .map_err(|e| {
221                         Error::PcztSign(format!(
222                             "Failed to sign Sapling spend {index}: {:?}",
223                             e
224                         ))
225                     })?;
226             }
```

Listing 2.2: src/bindgen/sign.rs

```
168 pub fn sign_sapling(
169     &mut self,
170     index: usize,
171     ask: &sapling::keys::SpendAuthorizingKey,
172 ) -> Result<(), Error> {
173     let spend = self
174         .sapling
```

```
175             .spends_mut()
176             .get_mut(index)
177             .ok_or(Error::InvalidIndex)?;
178
179         // Check consistency of the input being signed if we have its note components.
180         match spend.verify_nullifier(None) {
181             Err(
182                 sapling::pczt::VerifyError::MissingRecipient
183                 | sapling::pczt::VerifyError::MissingValue
184                 | sapling::pczt::VerifyError::MissingRandomSeed,
185             ) => Ok(()),
186             r => r,
187         }
188         .map_err(Error::SaplingVerify)?;
189
190         spend
191             .sign(self.shielded_sighash, ask, OsRng)
192             .map_err(Error::SaplingSign)?;
```

Listing 2.3: librustzcash/pczt/src/roles/signer/mod.rs

```
176     pub(crate) ephemeral_key: [u8; 32],
177     /// The encrypted note plaintext for the output.
178     ///
179     /// Encoded as a `Vec<u8>` because its length depends on the transaction version.
180     ///
181     /// Once we have [memo bundles], we will be able to set memos independently of
182     /// Outputs. For now, the Constructor sets both at the same time.
183     ///
184     /// [memo bundles]: https://zips.z.cash/zip-0231
185     #[getset(get = "pub")]
```

Listing 2.4: librustzcash/pczt/src/sapling.rs

Impact Sensitive fields are transmitted to a remote server, exposing transaction privacy.

Suggestion Remove fields not required for proof generation before sending a remote request.

2.1.2 Sensitive data exposure through console logs

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `wallet.rs`, multiple console logs may expose sensitive data to unintended parties.

1. At Line 318, UFVK (Unified Full Viewing Key) is logged, and its disclosure exposes incoming and outgoing transfers as well as transaction history.

2. At Lines 863, 887, and 933, the proposal and transaction objects are logged, which may contain sensitive transaction-intent details such as recipients, amounts, change outputs, input selection, and shielding sources.

```
310   async fn import_account_ufvk(
311       &self,
312       account_name: &str,
313       ufvk: &UnifiedFullViewingKey,
314       birthday_height: Option<u32>,
315       purpose: AccountPurpose,
316       key_source: Option<&str>,
317   ) -> Result<AccountId, Error> {
318       tracing::info!("Importing account with Ufvk: {:?}", ufvk);
```

Listing 2.5: src/wallet.rs

```
863       tracing::info!("Shielding proposal created: {:#?}", proposal);
```

Listing 2.6: src/wallet.rs

```
887       tracing::info!("Shielding transactions created: {:?}", transactions);
```

Listing 2.7: src/wallet.rs

```
933       tracing::info!("Proposal: {:#?}", proposal);
```

Listing 2.8: src/wallet.rs

Impact Sensitive data may be exposed to anyone with access to application logs or tracing output.

Suggestion Remove sensitive data from console output, and disable verbose tracing in production builds.

2.1.3 Lack of Sapling receiver in returned unified address

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `src/bindgen/wallet.rs`, the function `get_unified_addresses()` invokes `uivk.default_address(UAR::ALLOW_ALL)` to retrieve unified addresses, including the Orchard, Sapling, and Transparent receivers. However, the current design violates the ZIP32 specification [1], since the returned unified address may omit the Sapling receiver. Specifically, `UAR::ALLOW_ALL` only requires the returned address to contain at least one of the three receiver types. When the Sapling receiver derived at diversifier `0` is invalid, the function returns an address with only Orchard and Transparent receivers. It does not increment the diversifier for further Sapling receiver search. As a result, users may receive a unified address without a Sapling receiver, preventing users from receiving funds through the Sapling pool.

The same issue also exists in the function `mnemonic_to_addresses()`.

[1] https://zips.z.cash/zip-0032#sapling-key-path

```
472   pub async fn get_unified_addresses(&self, account_id: u32) -> Result<JsValue, Error> {
473       use zcash_address::unified::{self, Container};
474       use zcash_client_backend::data_api::Account;
475       use zcash_client_backend::encoding::AddressCodec;
476       use zcash_keys::keys::UnifiedAddressRequest as UAR;
477       use zcash_primitives::consensus::Parameters;
478       use zcash_primitives::legacy::TransparentAddress;
479
480       let db = self.inner.db.read().await;
481       let account = db
482           .get_account(account_id.into())?
483           .ok_or(Error::AccountNotFound(account_id))?;
484
485       // Get the unified incoming viewing key from the account
486       let uivk = account.uivk();
487
488       // Generate the full unified address with all available receivers
489       let (full_address, _) = uivk.default_address(UAR::ALLOW_ALL)?;
490       let unified_str = full_address.encode(&self.inner.network);
```

Listing 2.9: src/bindgen/wallet.rs

Impact Users may be unable to receive funds through the Sapling pool, violating the ZIP-32 specification.

Suggestion Replace `UAR::ALLOW_ALL` with `UAR::AllAvailableKeys` to ensure all receiver types are included in the returned address.

2.1.4 Incorrect implementation of function `utxo_query_height()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `librustzcash/zcash_client_memory/src/wallet_read.rs`, the function `utxo_query_height()` returns the starting block height to scan transparent UTXOs for a given account. However, the function ignores the parameter `_account` and instead returns the earliest height in `blocks` (the in-memory record of scanned blocks shared across all accounts), or 0 when no blocks have been scanned yet.

Consequently, if an account is imported with a birthday older than the wallet's earliest scanned block, any UTXOs created before that height will be missed entirely during scanning.

```
803   #[cfg(feature = "transparent-inputs")]
804   fn utxo_query_height(&self, _account: Self::AccountId) -> Result<BlockHeight, Self::Error> {
805       // Return the height of the first block in the wallet, or a default starting height
806       // This tells the wallet to query for transparent UTXOs starting from the earliest
807       // block we've scanned. For a more sophisticated implementation, this should track
808       // transparent address usage and gaps similar to zcash_client_sqlite.
809       Ok(self
810           .blocks
811           .keys()
```

```
812             .next()
813             .copied()
814             .unwrap_or_else(|| BlockHeight::from_u32(0)))
815     }
```

Listing 2.10: librustzcash/zcash_client_memory/src/wallet_read.rs

Impact Accounts may have understated balances due to omitted UTXOs.

Suggestion Use the given account's birthday height as a lower bound.

2.1.5 Incorrect transparent address resolution

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `librustzcash/zcash_client_memory/src/types/account.rs`, the function `find_account_for_transparent_address()` searches for the account ID that owns a given transparent address for UTXO attribution. However, the fallback branch (lines 151 - 155) iterates through accounts and returns the first account ID that owns any legacy transparent address, without comparing the target address. This misattributes incoming UTXOs to an unrelated account, causing incorrect balance reporting.

```
128     #[cfg(feature = "transparent-inputs")]
129     pub(crate) fn find_account_for_transparent_address(
130         &self,
131         address: &TransparentAddress,
132     ) -> Result<Option<AccountId>, Error> {
133         // Look for transparent receivers generated as part of a Unified Address
134         if let Some(id) = self
135             .accounts
136             .iter()
137             .find(|(_, account)| {
138                 account
139                     .addresses()
140                     .iter()
141                     .any(|(_, unified_address)| unified_address.transparent() == Some(address))
142             })
143             .map(|(id, _)| *id)
144         {
145             Ok(Some(id))
146         } else {
147             // then look at ephemeral addresses
148             if let Some(id) = self.find_account_for_ephemeral_address(address)? {
149                 Ok(Some(id))
150             } else {
151                 for (account_id, account) in self.accounts.iter() {
152                     if account.get_legacy_transparent_address()?.is_some() {
153                         return Ok(Some(*account_id));
154                     }
155                 }
```

```
156                 Ok(None)
157             }
158         }
159     }
```

Listing 2.11: librustzcash/zcash_client_memory/src/types/account.rs

Impact Incoming transparent UTXOs are incorrectly credited to the first available account in multi-account wallets.

Suggestion Compare the target address against each account's derived legacy address and only return the account ID upon an exact match.

Clarification from BlockSec Note this issue is not within the audit scope. We are disclosing it in accordance with our thorough audit methodology and responsible disclosure principles.

2.1.6 Lack of restriction on new mnemonic length

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description ZIP-315 [2] requires that wallets generate at least 24-word mnemonics for new wallet creation, while permitting shorter mnemonics only for recovery. However, the function `generate_mnemonic()` allows the wallet to generate mnemonics with fewer than 24 words, allowing the creation of new wallets with non-compliant mnemonic lengths that provide less entropy.

```
39pub fn generate_mnemonic(word_count: u32) -> Result<String, Error> {
40    let count = match word_count {
41        12 => Count::Words12,
42        15 => Count::Words15,
43        18 => Count::Words18,
44        21 => Count::Words21,
45        24 => Count::Words24,
46        _ => return Err(Error::InvalidSeedPhrase),
47    };
48
49    let mnemonic: Mnemonic<English> = Mnemonic::generate(count);
50    Ok(mnemonic.phrase().to_string())
51}
```

Listing 2.12: src/bindgen/mnemonic.rs

Impact The wallet may generate mnemonics with insufficient length.

Suggestion Enforce a minimum of 24 words in the function `generate_mnemonic()`.

[2] https://zips.z.cash/zip-0315#wallet-seeds

2.1.7 Incorrect check in function `get_wallet_summary()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `librustzcash/zcash_client_memory/src/wallet_read.rs`, the function `get_wallet_summary()` iterates over the user's received notes to aggregate the account balance. For each note, the function retrieves its containing transaction and filters the note out when `note_tx.expiry_height() < target_height` (line 233). However, the check never inspects whether the transaction has been mined. Once the current block height exceeds a transaction's expiry height, confirmed on-chain transactions are incorrectly filtered out. The corresponding notes are excluded from the balance aggregation, under-reporting the user's spendable funds.

```
193   fn get_wallet_summary(
194       &self,
195       confirmations_policy: ConfirmationsPolicy,
196   ) -> Result<Option<WalletSummary<Self::AccountId>>, Self::Error> {
197       tracing::debug!("get_wallet_summary");
198       let chain_tip_height = match self.chain_height()? {
199           Some(height) => height,
200           None => return Ok(None),
201       };
202       let target_height = TargetHeight::from(chain_tip_height + 1);
203       let birthday_height = self
204           .get_wallet_birthday()?
205           .expect("If a scan range exists, we know the wallet birthday.");
206
207       let fully_scanned_height = self
208           .block_fully_scanned()?
209           .map_or(birthday_height - 1, |m| m.block_height());
210
211       let mut account_balances = self
212           .accounts
213           .values()
214           .map(|account| (account.account_id(), AccountBalance::ZERO))
215           .collect::<HashMap<AccountId, AccountBalance>>();
216
217       for note in self.get_received_notes().iter() {
218           // don't count spent notes
219           if self.note_is_spent(note, target_height)? {
220               continue;
221           }
222           // don't count notes in unscanned ranges
223           let unscanned_ranges = self.unscanned_ranges();
224           for (_, _, start_position, end_position_exclusive) in unscanned_ranges {
225               if note.commitment_tree_position >= start_position
226                   && note.commitment_tree_position < end_position_exclusive
227               {
228                   continue; // note is in an unscanned range. Skip it
```

```
229             }
230         }
231         // don't count notes in unmined transactions or that have expired
232         if let Ok(Some(note_tx)) = self.get_transaction(note.txid) {
233             if note_tx.expiry_height() < BlockHeight::from(target_height) {
234                 continue;
235             }
236         }
```

Listing 2.13: librustzcash/zcash_client_memory/src/wallet_read.rs

Impact The function `get_wallet_summary()` reports an incorrect balance to the user.

Suggestion Add a check on the mined status of the transaction.

Clarification from BlockSec Note this issue is not within the audit scope. We are disclosing it in accordance with our thorough audit methodology and responsible disclosure principles.

2.1.8 Lack of support for BIP39 passphrases

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `src/bindgen/mnemonic.rs`, the function `usk_from_seed_str()` is invoked by the function `create_account()` to derive a Unified Spending Key from a mnemonic phrase. However, the function statically hardcodes the BIP-39 passphrase to an empty string via `mnemonic.to_seed("")`. This behavior is inconsistent with the function `mnemonic_to_seed()`, which accepts a passphrase and derives the seed via `mnemonic.to_seed(passphrase)`.

```
72  pub async fn create_account(
73      &self,
74      account_name: &str,
75      seed_phrase: &str,
76      account_hd_index: u32,
77      birthday_height: Option<u32>,
78  ) -> Result<u32, Error> {
79      tracing::info!("Create account called");
80      self.inner
81          .create_account(
82              account_name,
83              seed_phrase,
84              account_hd_index,
85              birthday_height,
86              None,
87          )
88          .await
89          .map(|id| *id)
90  }
```

Listing 2.14: src/bindgen/wallet.rs

```
1079pub(crate) fn usk_from_seed_str(
1080    seed: &str,
1081    account_id: u32,
1082    network: &Network,
1083) -> Result<(UnifiedSpendingKey, SeedFingerprint), Error> {
1084    let mnemonic = <Mnemonic<English>>::from_phrase(seed).map_err(|_| Error::InvalidSeedPhrase)?;
1085    let seed = {
1086        let mut seed = mnemonic.to_seed("");
1087        let secret = seed.to_vec();
1088        seed.zeroize();
1089        SecretVec::new(secret)
1090    };
1091    let seed_fingerprint =
1092        SeedFingerprint::from_seed(seed.expose_secret()).expect("seed fingerprint");
1093    let usk = UnifiedSpendingKey::from_seed(network, seed.expose_secret(), account_id.try_into()?)
          ?;
1094    Ok((usk, seed_fingerprint))
1095}
```

Listing 2.15: src/wallet.rs

```
145#[wasm_bindgen]
146pub fn mnemonic_to_seed(phrase: &str, passphrase: &str) -> Result<Vec<u8>, Error> {
147    let mnemonic: Mnemonic<English> =
148        Mnemonic::from_phrase(phrase).map_err(|_| Error::InvalidSeedPhrase)?;
149
150    let seed = mnemonic.to_seed(passphrase);
151    Ok(seed.to_vec())
152}
```

Listing 2.16: src/bindgen/mnemonic.rs

Impact Users cannot import mnemonics protected by BIP39 passphrases in the project.

Suggestion Revise the logic accordingly.

2.1.9 Potential panics

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the project, several functions trigger panic in certain cases, causing the WASM runtime to abort immediately.

1. In the function `create_account()`, the birthday derivation performs unchecked `u32` subtraction. If the caller passes `birthday_height` as 0 or lightwalletd returns a `chain_tip` below 100, the subtraction underflows and triggers panic.

2. In the function `sign_prove_and_send_pczt()`, the returned `proved_pczt` is decoded with two `unwrap()` invocations. If the proving server returns a non-base64 string or malformed PCZT bytes, the function will panic.

3. In the function `parse_unified_address()`, receiver variants are handled with `unreachable!()`. If a malformed or unsupported unified address reaches this path, the wallet may panic instead of returning a recoverable parsing error.

4. The function `usk_from_seed_str()` uses `expect()` on seed-derived key material. If the supplied seed string is malformed or otherwise invalid, the function may panic instead of returning a recoverable error.

5. In the function `pczt_sign_inner()`, the statement `let mut signer = Signer::new(pczt).unwrap()` assumes the provided `pczt` is always valid. If an invalid or malformed `pczt` reaches this path, the wallet may panic during signing.

6. In the `Pczt` binding methods (i.e., `to_json()`, `from_json()`, `from_bytes()`), serialization and parsing operations use `unwrap()` without error propagation. Malformed JSON or invalid postcard bytes will therefore trigger a panic and abort the WASM runtime.

```
336     let birthday = match birthday_height {
337         Some(height) => height,
338         None => {
339             let chain_tip: u32 = client
340                 .get_latest_block(service::ChainSpec::default())
341                 .await?
342                 .into_inner()
343                 .height
344                 .try_into()
345                 .expect("block heights must fit into u32");
346             chain_tip - 100
347         }
348     };
349     // Construct an `AccountBirthday` for the account's birthday.
350     let birthday = {
351         // Fetch the tree state corresponding to the last block prior to the wallet's
352         // birthday height. NOTE: THIS APPROACH LEAKS THE BIRTHDAY TO THE SERVER!
353         let request = service::BlockId {
354             height: (birthday - 1).into(),
355             ..Default::default()
356         };
```

Listing 2.17: src/wallet.rs

```
647             Ok(ProveResponse { proved_pczt }) => {
648                 let proved_pczt_bytes = BASE64_STANDARD.decode(&proved_pczt).unwrap();
649                 pczt::Pczt::parse(&proved_pczt_bytes).unwrap()
650             }
```

Listing 2.18: src/bindgen/wallet.rs

```
470         unified::Receiver::P2sh(_) => {
471             unreachable!();
472         }
473         unified::Receiver::Unknown { .. } => {
474             unreachable!();
475         }
```

Listing 2.19: src/bindgen/mnemonic.rs

```
1091    let seed_fingerprint =
1092        SeedFingerprint::from_seed(seed.expose_secret()).expect("seed fingerprint");
1093    let usk = UnifiedSpendingKey::from_seed(network, seed.expose_secret(), account_id.try_into()?)
            ?;
1094    Ok((usk, seed_fingerprint))
```

Listing 2.20: src/wallet.rs

```
198    let mut signer = Signer::new(pczt).unwrap();
```

Listing 2.21: src/bindgen/sign.rs

```
21#[wasm_bindgen]
22impl Pczt {
23    /// Returns a JSON object with the details of the Pczt.
24    pub fn to_json(&self) -> JsValue {
25        serde_wasm_bindgen::to_value(&self).unwrap()
26    }
27
28    /// Returns a Pczt from a JSON object
29    pub fn from_json(s: JsValue) -> Pczt {
30        serde_wasm_bindgen::from_value(s).unwrap()
31    }
32
33    /// Returns the postcard serialization of the Pczt.
34    pub fn serialize(&self) -> Vec<u8> {
35        self.0.serialize()
36    }
37
38    /// Deserialize to a Pczt from postcard bytes.
39    pub fn from_bytes(bytes: &[u8]) -> Pczt {
40        Self(pczt::Pczt::parse(bytes).unwrap())
41    }
42}
```

Listing 2.22: src/bindgen/pczt.rs

Impact Unrecoverable panics force immediate host page reloads and may interrupt critical database mutations, leaving wallet state inconsistent.

Suggestion Revise the logic accordingly.

2.1.10 Lack of handling for broadcast failures

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the file `src/wallet.rs`, both functions `shield_funds()` and `pczt_send()` persist transactions to local storage before broadcast. The consumed notes or UTXOs are marked as spent at the point of local storage. If the subsequent invocation to `send_authorized_transactions()`

fails, an error is returned, but no rollback occurs. The local transaction record is neither removed nor flagged as broadcast-failed. Consequently, the affected funds remain locked until the transaction expires or a retry succeeds.

```
1035  pub async fn pczt_send(&self, pczt: Pczt) -> Result<String, Error> {
1036      let prover = get_prover();
1037      let (spend_vk, output_vk) = prover.verifying_keys();
1038      let mut db = self.db.write().await;
1039
1040      let orchard_verifying_key = get_orchard_verifying_key();
1041      tracing::info!("  Extracting and storing transaction from PCZT...");
1042      let start_time = js_sys::Date::now();
1043      let txid = extract_and_store_transaction_from_pczt::<_, ()>(
1044          &mut *db,
1045          pczt,
1046          Some((&spend_vk, &output_vk)),
1047          Some(orchard_verifying_key),
1048      )
1049      .map_err(|e| {
1050          Error::PcztSend(format!(
1051              "Failed to extract and store transaction from PCZT: {:?}",
1052              e
1053          ))
1054      })?;
1055      let end_time = js_sys::Date::now();
1056      let elapsed_seconds = (end_time - start_time) / 1000.0;
1057      tracing::info!(
1058          "  extract_and_store_transaction_from_pczt completed in {:.2} seconds",
1059          elapsed_seconds
1060      );
1061      tracing::info!("  Transaction stored with txid: {:?}", txid);
1062      drop(db);
1063
1064      tracing::info!("  Broadcasting transaction to network...");
1065      self.send_authorized_transactions(&NonEmpty::new(txid))
1066          .await?;
1067
1068      tracing::info!("  Transaction broadcast successfully!");
1069      Ok(txid.to_string())
1070  }
```

Listing 2.23: src/wallet.rs

```
868
869       let transactions = create_proposed_transactions::<
870           _,
871           _,
872           <MemoryWalletDb<Network> as InputSource>::Error,
873           _,
874           <StandardFeeRule as FeeRule>::Error,
875           _,
876       >(
877           &mut *db,
```

```
878             &self.network,
879             prover,
880             prover,
881             &spending_keys,
882             OvkPolicy::Sender,
883             &proposal,
884         )
885         .map_err(|e| Error::Generic(format!("Failed to create shielding transaction: {:?}", e)))?;
886 
887         tracing::info!("Shielding transactions created: {:?}", transactions);
888 
889         // Transactions are automatically stored in the database by create_proposed_transactions
890         // Now we need to broadcast them
891         drop(db); // Release the lock before sending
892 
893         self.send_authorized_transactions(&transactions).await?;
```

Listing 2.24: src/wallet.rs

Impact A broadcast failure leaves the wallet in an inconsistent state.

Suggestion Revise the logic accordingly.

Feedback from the project The project states that the frontend creates a database snapshot at critical points in the transaction workflow. When the broadcast fails, the frontend rolls back to the pre-send snapshot.

2.1.11 Lack of zeroization for sensitive `seed` value

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `src/bindgen/mnemonic.rs`, the functions `mnemonic_to_seed()`, `mnemonic_to_spending_key()`, `mnemonic_to_seed_fingerprint()`, and `mnemonic_to_addresses()` derive cryptographic keys and seeds from mnemonic phrases. However, these functions allocate unprotected copies of the raw 64-byte `seed` via `seed.to_vec()` and transfer them across the WASM boundary without utilizing secure memory wrappers or zeroizing the original buffers.

Additionally, in the file `src/bindgen/keys.rs`, the constructor for `UnifiedSpendingKey` accepts the seed as a plain `Box<[u8]>` and fails to scrub the memory after derivation, leaving the root secret resident in both WASM linear memory and the JavaScript heap until garbage collection occurs.

```
145#[wasm_bindgen]
146pub fn mnemonic_to_seed(phrase: &str, passphrase: &str) -> Result<Vec<u8>, Error> {
147    let mnemonic: Mnemonic<English> =
148        Mnemonic::from_phrase(phrase).map_err(|_| Error::InvalidSeedPhrase)?;
149 
150    let seed = mnemonic.to_seed(passphrase);
151    Ok(seed.to_vec())
152}
```

Listing 2.25: src/bindgen/mnemonic.rs

```
178#[wasm_bindgen]
179pub fn mnemonic_to_spending_key(
180    network: &str,
181    phrase: &str,
182    passphrase: &str,
183    account_index: u32,
184) -> Result<UnifiedSpendingKey, Error> {
185    // Convert mnemonic to seed
186    let mnemonic: Mnemonic<English> =
187        Mnemonic::from_phrase(phrase).map_err(|_| Error::InvalidSeedPhrase)?;
188
189    let seed = mnemonic.to_seed(passphrase);
190
191    // Create UnifiedSpendingKey from seed
192    UnifiedSpendingKey::new(network, seed.to_vec().into_boxed_slice(), account_index)
193}
```

Listing 2.26: src/bindgen/mnemonic.rs

```
215#[wasm_bindgen]
216pub fn mnemonic_to_seed_fingerprint(
217    phrase: &str,
218    passphrase: &str,
219) -> Result<SeedFingerprint, Error> {
220    // Convert mnemonic to seed
221    let mnemonic: Mnemonic<English> =
222        Mnemonic::from_phrase(phrase).map_err(|_| Error::InvalidSeedPhrase)?;
223
224    let seed = mnemonic.to_seed(passphrase);
225
226    // Create SeedFingerprint from seed
227    SeedFingerprint::new(seed.to_vec().into_boxed_slice())
228}
```

Listing 2.27: src/bindgen/mnemonic.rs

```
317    let seed = mnemonic.to_seed(passphrase);
318
319    // Create UnifiedSpendingKey from seed
320    let usk = UnifiedSpendingKey::new(network, seed.to_vec().into_boxed_slice(), account_index)?;
```

Listing 2.28: src/bindgen/mnemonic.rs

```
 79    pub fn new(network: &str, seed: Box<[u8]>, hd_index: u32) -> Result<UnifiedSpendingKey, Error>
           {
 80        let network = Network::from_str(network)?;
 81        Ok(Self {
 82            inner: zcash_keys::keys::UnifiedSpendingKey::from_seed(
 83                &network,
 84                &seed,
```

```
85              AccountId::try_from(hd_index)?,
86          )?,
87      })
88  }
```

Listing 2.29: src/bindgen/keys.rs

Impact The persistence of the master seed in unmanaged memory exposes the wallet to total compromise through heap inspection or cross-site scripting vulnerabilities.

Suggestion Enforce the memory hygiene by wrapping all `seed.to_vec()` with `Secret::new()`.

2.1.12 Incorrect balance display during wallet recovery

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the file `src/wallet.rs`, the function `create_account()` uses `chain_tip - 100` as the default account birthday when `birthday_height` is `None`. During account setup, function `add_account()` marks all blocks from Sapling activation up to `birthday.height()` as ignored in the scan queue (line 227), and schedules only the range `birthday.height()..(self.chain_height + 1)` for scanning (line 242). For wallet recovery, if the account received funds before the default birthday height, those blocks will never be scanned, causing historical transactions to be missed and the balance display to be incomplete.

```
336     let birthday = match birthday_height {
337         Some(height) => height,
338         None => {
339             let chain_tip: u32 = client
340                 .get_latest_block(service::ChainSpec::default())
341                 .await?
342                 .into_inner()
343                 .height
344                 .try_into()
345                 .expect("block heights must fit into u32");
346             chain_tip - 100
347         }
348     };
349     // Construct an `AccountBirthday` for the account's birthday.
350     let birthday = {
351         // Fetch the tree state corresponding to the last block prior to the wallet's
352         // birthday height. NOTE: THIS APPROACH LEAKS THE BIRTHDAY TO THE SERVER!
353         let request = service::BlockId {
354             height: (birthday - 1).into(),
355             ..Default::default()
356         };
```

Listing 2.30: src/wallet.rs

```
226     if sapling_activation_height < birthday.height() {
227         let ignored_range = sapling_activation_height..birthday.height();
```

```
228             self.scan_queue.replace_queue_entries(
229                 &ignored_range,
230                 Some(ScanRange::from_parts(
231                     ignored_range.clone(),
232                     ScanPriority::Ignored,
233                 ))
234                 .into_iter(),
235                 false,
236             )?;
237         };
238
239         // Rewrite the scan ranges from the birthday height up to the chain tip so that we'll
                ensure we
240         // re-scan to find any notes that might belong to the newly added account.
241         if let Some(t) = self.chain_height()? {
242             let rescan_range = birthday.height()..(t + 1);
243             self.scan_queue.replace_queue_entries(
244                 &rescan_range,
245                 Some(ScanRange::from_parts(
246                     rescan_range.clone(),
247                     ScanPriority::Historic,
248                 ))
249                 .into_iter(),
250                 true, // force rescan
251             )?;
```

Listing 2.31: librustzcash/zcash_client_memory/src/types/memory_wallet/mod.rs

```
152     let mut scan_ranges = db_data.suggest_scan_ranges().map_err(Error::Wallet)?;
153     loop {
154         match scan_ranges.first() {
155             Some(scan_range) if scan_range.priority() == ScanPriority::Verify => {
156                 download_blocks(client, db_cache, scan_range).await?;
157                 let chain_state =
158                     download_chain_state(client, scan_range.block_range().start - 1).await?;
159                 let scan_ranges_updated =
160                     scan_blocks(params, db_cache, db_data, &chain_state, scan_range).await?;
161                 db_cache
162                     .delete(scan_range.clone())
163                     .await
164                     .map_err(Error::Cache)?;
165                 if scan_ranges_updated {
166                     scan_ranges = db_data.suggest_scan_ranges().map_err(Error::Wallet)?;
167                 } else {
168                     break;
169                 }
170             }
171             _ => break,
172         }
```

Listing 2.32: librustzcash/zcash_client_backend/src/sync.rs

Impact Wallet recovery without an explicit birthday may miss historical transactions and display incomplete balances.

Suggestion Require the caller to provide an explicit birthday height or use `280000` for the default birthday.

Feedback from the project The project states that the frontend supports specifying the birthday by block height or by year and month. It also alerts users that using the default birthday may cause earlier transactions to be omitted.

2.2 Recommendation

2.2.1 Revise the incorrect annotation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The annotation for the function `run_next_batch()` states: "Returns 'true' if there are more blocks, 'false' if the wallet is fully synced." However, this annotation is inconsistent with the function implementation. The function returns true whenever it processes a batch, and false only when the function `suggest_scan_ranges()` returns no ranges.

```
85/// Syncs a single batch of blocks and returns whether there is more work to do.
86///
87/// Unlike [`run`], which loops until the wallet is fully synced, this function
88/// processes at most `batch_size` blocks (plus any Verify-priority ranges for
89/// chain continuity) and then returns. Returns `true` if there are more blocks
90/// to sync, `false` if the wallet is fully synced.
```

Listing 2.33: librustzcash/zcash_client_backend/src/sync.rs

```
175   // Get scan ranges and take only the first batch_size chunk
176   let scan_ranges = db_data.suggest_scan_ranges().map_err(Error::Wallet)?;
177   debug!("Suggested ranges: {:?}", scan_ranges);
178
179   // Get the first non-empty scan range and limit it to batch_size
180   let first_batch = scan_ranges
181       .into_iter()
182       .flat_map(|r| {
183           (0..).scan(r, |acc, _| {
184               if acc.is_empty() {
185                   None
186               } else if let Some((cur, next)) = acc.split_at(acc.block_range().start + batch_size)
187               {
188                   *acc = next;
189                   Some(cur)
190               } else {
191                   let cur = acc.clone();
192                   let end = acc.block_range().end;
193                   *acc = ScanRange::from_parts(end..end, acc.priority());
194                   Some(cur)
195               }
196           })
197       })
198       .next();
```

```
199
200   match first_batch {
201       Some(scan_range) => {
202           download_blocks(client, db_cache, &scan_range).await?;
203           let chain_state =
204               download_chain_state(client, scan_range.block_range().start - 1).await?;
205           scan_blocks(params, db_cache, db_data, &chain_state, &scan_range).await?;
206           db_cache.delete(scan_range).await.map_err(Error::Cache)?;
207           Ok(true)
208       }
209       None => Ok(false),
210   }
```

Listing 2.34: librustzcash/zcash_client_backend/src/sync.rs

Suggestion Revise the annotation.

2.2.2 Add a check in function `pczt_sign_inner()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `src/bindgen/sign.rs`, function `pczt_sign()` signs all transparent inputs in the PCZT that match a `seed_fp`, without verifying whether these inputs originate from the original proposal.

```
 97       .with_transparent::<Infallible, _>(|bundle| {
 98           for (index, input) in bundle.inputs().iter().enumerate() {
 99               for derivation in input.bip32_derivation().values() {
100                   // The HARDENED_FLAG constant (2^31)
101                   const HARDENED_FLAG: u32 = 1 << 31;
102
103                   // Create ChildNumber directly from u32 using the internal representation
104                   // This matches the coin_type with hardened flag set
105                   let coin_type_value = network.network_type().coin_type() | HARDENED_FLAG;
106
107                   if let Some((account_index, scope, address_index)) =
108                       derivation.extract_bip_44_fields(&seed_fp, coin_type_value.into())
109                   {
110                       keys.entry(account_index)
111                           .or_default()
112                           .push(KeyRef::Transparent {
113                               index,
114                               scope,
115                               address_index,
116                           });
117                   }
118               }
119           }
120           Ok(())
121       })
```

Listing 2.35: src/bindgen/sign.rs

```
227             KeyRef::Transparent {
228                 index,
229                 scope,
230                 address_index,
231             } => {
232                 signer
233                     .sign_transparent(
234                         index,
235                         &usk.transparent()
236                             .derive_secret_key(scope, address_index)
237                             .map_err(|e| {
238                                 Error::PcztSign(format!(
239                                     "Failed to derive transparent key at .../{:?}/{:?}: {:?}",
240                                     scope, address_index, e,
241                                 ))
242                             })?,
243                     )
244                     .map_err(|e| {
245                         Error::PcztSign(format!(
246                             "Failed to sign transparent input {index}: {:?}",
247                             e
248                         ))
249                     })?;
```

Listing 2.36: src/bindgen/sign.rs

Suggestion Add related validation logic with `MemoryWalletDb.tx_table`.

2.2.3 Unify implementation between payment request and PCZT creation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `src/bindgen/requests.rs`, the constructor `PaymentRequest::new()` accepts the `memo`, indicating that the wallet payment supports memo. However, the function `pczt_create()` does not have the corresponding `memo` parameter, leaving memo support absent from the PCZT creation path.

```
50  pub fn new(
51      recipient_address: &str,
52      amount: u64,
53      memo: Option<Vec<u8>>,
54      label: Option<String>,
55      message: Option<String>,
56      other_params: JsValue,
57  ) -> Result<PaymentRequest, Error> {
```

Listing 2.37: src/bindgen/requests.rs

```
665 pub async fn pczt_create(
666     &self,
667     account_id: u32,
```

```
668         to_address: String,
669         value: u64,
670     ) -> Result<Pczt, Error> {
671         let to_address = ZcashAddress::try_from_encoded(&to_address)?;
672         self.inner
673             .pczt_create(AccountId::from(account_id), to_address, value)
674             .await
675             .map(Into::into)
676     }
```

Listing 2.38: src/bindgen/wallet.rs

Suggestion Add memo support to the function `pczt_create()`.

2.2.4 Unify the confirmation policy

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The wallet applies confirmation policies inconsistently in two shielding flows. Specifically, the function `pczt_shield()` uses the `self.confirmations_policy()` for both balance queries and shielding proposals. However, function `shield_funds()` uses `self.confirmations_policy()` for balance queries (line 839) but then hardcodes a 1-confirmation policy for the shielding proposal (line 859).

Additionally, the wallet-wide `confirmations_policy()` (line 165-170) applies a single threshold to both trusted and untrusted TXOs, discarding ZIP-315 [3]'s distinction of 3 confirmations for trusted and 10 for untrusted outputs (ConfirmationsPolicy::default()).

```
758         let transparent_balances = db.get_transparent_balances(
759             account_id,
760             (max_height + 1).into(),
761             self.confirmations_policy(),
762         )?;
763         let from_addrs = transparent_balances.into_keys().collect::<Vec<_>>();
764
765         if from_addrs.is_empty() {
766             return Err(Error::Generic(
767                 "No transparent addresses found for this account. Make sure the account has received
                        funds to a transparent address.".to_string()
768             ));
769         }
770
771         tracing::info!("Shielding from {} transparent addresses", from_addrs.len());
772
773         let proposal = propose_shielding::<_, _, _, _, <W as WalletCommitmentTrees>::Error>(
774             &mut *db,
775             &self.network,
776             &input_selector,
777             &change_strategy,
778             SHIELDING_THRESHOLD,
```

[3] https://zips.z.cash/zip-0315#trusted-and-untrusted-txos

```
779             &from_addrs,
780             account_id,
781             self.confirmations_policy(),
782         )
783         .map_err(|e| Error::Generic(format!("Error when shielding: {:?}", e)))?;
```

Listing 2.39: src/wallet.rs

```
836         let transparent_balances = db.get_transparent_balances(
837             account_id,
838             (max_height + 1).into(),
839             self.confirmations_policy(),
840         )?;
841         let from_addrs = transparent_balances.into_keys().collect::<Vec<_>>();
842
843         if from_addrs.is_empty() {
844             return Err(Error::Generic(
845                 "No transparent addresses found for this account. Make sure the account has received
                        funds to a transparent address.".to_string()
846             ));
847         }
848
849         tracing::info!("Shielding from {} transparent addresses", from_addrs.len());
850
851         let proposal = propose_shielding::<_, _, _, _, <W as WalletCommitmentTrees>::Error>(
852             &mut *db,
853             &self.network,
854             &input_selector,
855             &change_strategy,
856             SHIELDING_THRESHOLD,
857             &from_addrs,
858             account_id,
859             ConfirmationsPolicy::new_symmetrical(NonZeroU32::new(1).unwrap(), true),
860         )
861         .map_err(|e| Error::Generic(format!("Error when proposing shielding: {:?}", e)))?;
```

Listing 2.40: src/wallet.rs

```
162     /// Creates a confirmations policy using the wallet's min_confirmations setting.
163     /// This applies the same confirmation requirement to both trusted and untrusted transactions,
164     /// and allows zero-confirmation shielding of transparent UTXOs (matching ZIP 315 defaults).
165     fn confirmations_policy(&self) -> ConfirmationsPolicy {
166         ConfirmationsPolicy::new_symmetrical(
167             self.min_confirmations,
168             true, // allow zero-conf shielding (matches default behavior)
169         )
170     }
```

Listing 2.41: src/wallet.rs

Suggestion Fix the confirmation policy.

2.2.5 Report pending and total balances

Status Fixed in `Version 2`

Introduced by `Version 1`

Description According to ZIP-315, wallets should report the spendable balance, along with either the pending balance or the total balance. However, the function `AccountBalance::from()`, which displays account balances, only returns the `spendable_value` field. As a result, users can not confirm their pending funds during the confirmation window.

```
779#[derive(Debug, Serialize, Deserialize)]
780pub struct AccountBalance {
781    pub sapling_balance: u64,
782    pub orchard_balance: u64,
783    pub unshielded_balance: u64,
784}
785
786impl From<zcash_client_backend::data_api::AccountBalance> for AccountBalance {
787    fn from(balance: zcash_client_backend::data_api::AccountBalance) -> Self {
788        AccountBalance {
789            sapling_balance: balance.sapling_balance().spendable_value().into(),
790            orchard_balance: balance.orchard_balance().spendable_value().into(),
791            unshielded_balance: balance.unshielded_balance().spendable_value().into(),
792        }
793    }
```

Listing 2.42: src/bindgen/wallet.rs

Suggestion Report the pending balance and total balance.

2.3 Note

2.3.1 Security assumptions on the host application

Introduced by `Version 1`

Description Certain critical responsibilities are performed by the wallet SDK's host application and fall outside the scope of this audit. The host application is assumed to be a Chrome extension and is expected to handle the following robustly:

1. The ZIP-317 transaction fee is disclosed to users with explicit confirmation before transaction submission.

2. The in-memory database returned by the function `db_to_bytes()` is encrypted before persisting or transmitting it.

```
356    pub async fn db_to_bytes(&self) -> Result<Box<[u8]>, Error> {
357        let bytes = self.inner.db_to_bytes().await?;
358        Ok(bytes.into_boxed_slice())
359    }
```

Listing 2.43: src/bindgen/wallet.rs

```
// pub fn encrypt_string(plaintext: &str, passphrase: &str) -> Result<String, Error> {
//     let passphrase = SecretString::from(passphrase);
//     let recipient = age::scrypt::Recipient::new(passphrase.clone());
//     let encrypted = age::encrypt(&recipient, plaintext.as_bytes())
//         .map_err(|e| Error::Encryption(e.to_string()))?;

//     // Return base64-encoded encrypted data
//     Ok(BASE64_STANDARD.encode(encrypted))
// }
```

Listing 2.44: src/bindgen/encryption.rs

3. HTTPS is enforced for all outbound connections from the SDK, including the gRPC endpoint specified by `lightwalletd_url`.

```
pub fn new(
    network: &str,
    lightwalletd_url: &str,
    min_confirmations: u32,
    db_bytes: Option<Box<[u8]>>,
) -> Result<WebWallet, Error> {
    let network = Network::from_str(network)?;
    let min_confirmations = NonZeroU32::try_from(min_confirmations)
        .map_err(|_| Error::InvalidMinConformations(min_confirmations))?;
    let client = Client::new(lightwalletd_url.to_string());
```

Listing 2.45: src/bindgen/wallet.rs

4. Critical parameters, e.g., the initialization parameters `network` and `lightwalletd_url`, are properly validated. The sensitive interfaces, e.g., function `to_unified_full_viewing_key()`, have proper access controls in place.

Feedback from the project The project states that the host application fulfills the above responsibilities. Additionally, the `lightwalletd_url` is hardcoded and uses an HTTPS endpoint.

2.3.2 Trust assumptions on the librustzcash library

Introduced by `Version 1`

Description The project relies heavily on librustzcash as its foundational Rust library for Zcash protocol operations. As described in Section 1.1, this review includes only the project-specific modifications to librustzcash. The remaining upstream librustzcash codebase, including most of its core protocol logic, cryptographic implementations, consensus rule enforcement, and wallet state management behavior, is treated as a trusted dependency and falls outside the scope of this audit.

2.3.3 Non-standard encoding in transparent message signing

Introduced by `Version 3`

Description In the file `src/bindgen/sign.rs`, the function `sign_message_with_transparent()` generates a recoverable ECDSA signature to prove ownership of a transparent address. The

recovery byte is encoded as `27 + recovery_id`, indicating an uncompressed public key under the Zcash message-signing convention. However, the function returns a transparent address derived from the compressed public key. Integrators should note that signatures produced by this function may not be verifiable through the Zcash RPC interface.

Additionally, function `mnemonic_to_transparent_address()` returns an `address` field whose P2PKH hash is derived from the compressed public key, and a `pubkey` field serialized in uncompressed form. The two representations are not interchangeable, and integrators should compress the returned `pubkey` manually to reconstruct or verify the `address`.

Feedback from the project The project states that the signature produced by this interface is verified by the project's custom verification logic, and the application-side integration has been completed.

2.3.4 Transfers to TEX addresses are not supported

Introduced by `Version 1`

Description The project does not support transfers to TEX (Transparent-Source-Only Address) addresses as defined in ZIP-320 [4]. The current exposed PCZT transfer path only accepts single-step proposals, while TEX transfers require multi-step proposals. Integrators should be aware that the project only supports non-TEX address transfers.

2.3.5 Address linkage during shielding

Introduced by `Version 2`

Description The function `pczt_shield()` consumes all spendable transparent UTXOs in a single transaction. Users should note that when a wallet holds funds across multiple transparent addresses, invoking this function may reveal on-chain linkage between those addresses.

```
960   pub async fn pczt_shield(&self, account_id: AccountId) -> Result<Pczt, Error> {
961       let change_strategy = MultiOutputChangeStrategy::new(
962           StandardFeeRule::Zip317,
963           None,
964           ShieldedProtocol::Orchard,
965           DustOutputPolicy::default(),
966           SplitPolicy::with_min_output_value(
967               NonZeroUsize::new(self.target_note_count)
968                   .ok_or(Error::FailedToCreateTransaction)?,
969               Zatoshis::from_u64(self.min_split_output_value)?,
970           ),
971       );
972
973       let input_selector = GreedyInputSelector::new();
974       let mut db = self.db.write().await;
975
976       // Get all transparent receivers for this account to use as shielding sources
977       let max_height = match db.chain_height()? {
978           Some(max_height) => max_height,
```

[4] https://zips.z.cash/zip-0320

```
979             None => {
980                 return Err(Error::Generic(
981                     "Havent scanned yet, cant shield".to_string(),
982                 ));
983             }
984         };
985
986         let transparent_balances = db.get_transparent_balances(
987             account_id,
988             (max_height + 1).into(),
989             self.confirmations_policy(),
990         )?;
991         let from_addrs = transparent_balances.into_keys().collect::<Vec<_>>();
992
993         if from_addrs.is_empty() {
994             return Err(Error::Generic(
995                 "No transparent addresses found for this account. Make sure the account has received
                      funds to a transparent address.".to_string()
996             ));
997         }
998
999         tracing::info!("Shielding from {} transparent addresses", from_addrs.len());
1000
1001         let proposal = propose_shielding::<_, _, _, _, <W as WalletCommitmentTrees>::Error>(
1002             &mut *db,
1003             &self.network,
1004             &input_selector,
1005             &change_strategy,
1006             SHIELDING_THRESHOLD,
1007             &from_addrs,
1008             account_id,
1009             self.confirmations_policy(),
1010         )
1011         .map_err(|e| Error::Generic(format!("Error when shielding: {:?}", e)))?;
1012
1013         let pczt = create_pczt_from_proposal::<
1014             _,
1015             _,
1016             <MemoryWalletDb<Network> as InputSource>::Error,
1017             _,
1018             <StandardFeeRule as FeeRule>::Error,
1019             _,
1020         >(
1021             &mut *db,
1022             &self.network,
1023             account_id,
1024             OvkPolicy::Sender,
1025             &proposal,
1026         )
1027         .map_err(|e| Error::Generic(format!("Failed to create PCZT for shielding: {:?}", e)))?;
1028
1029         Ok(pczt)
1030     }
```

Listing 2.46: src/wallet.rs

2.3.6 Constraints of PCZT implementation

Introduced by `Version 1`

Description Although the project introduces the PCZT (Partially Created Zcash Transaction) workflow, its implementation does not fully conform to the standard design. In particular, the project does not support multisignature workflows or hardware-wallet signing. Users should be aware of these limitations.

2.3.7 Worker-based execution for multi-threaded Wasm

Introduced by `Version 1`

Description In the host Chrome extension, several cryptographic or scanning functions (e.g., function `sync()`, `sync_next_batch()`, and `pczt_prove()`) are expected to utilize multi-threaded Wasm to optimize performance. Meanwhile, multi-threaded Wasm relies on blocking methods, e.g., `Atomics.wait()`, for thread coordination, and the Chrome browser disallows these methods on the main UI thread. Therefore, to ensure these multi-threaded functions can operate correctly, the Wasm module must run within a dedicated background worker.

Feedback from the project The project confirmed that multi-threaded Wasm is not executed on the main UI thread. Instead, multi-threaded Wasm execution is isolated via an offscreen document and the Web Worker architecture, with a fallback when the environment does not support worker-based thread pool initialization.

2.3.8 Library upgrades and integration are out of scope

Introduced by `Version 3`

Description `Version 3` incorporates an update to the underlying librustzcash library and its SDK integration. Both fall outside the scope of this audit and do not undergo security assessments. As a result, vulnerabilities inherent to the library itself, or in integration code that depends on its implementation, may remain unidentified and undisclosed.

2.3.9 Transparent receive transactions are omitted from history

Introduced by `Version 3`

Description In file `src/bindgen/wallet.rs`, function `get_transaction_history()` derives every entry solely from the shielded note tables, none of which hold incoming transparent UTXOs. As a result, transparent received funds are not counted, and a transaction that only pays into one of the account's transparent addresses, with no shielded inputs or outputs, is dropped entirely and never appears in the returned history. Integrators should not treat the returned history as a complete record of transparent address activity.

Feedback from the project The project states that transaction history for transparent receivers is fetched separately from a full node via the `getaddresstxids()` RPC.

2.3.10 Function `get_next_shielded_address()` is limited to shielded address rotation

Introduced by `Version 3`

Description In file `src/bindgen/wallet.rs`, function `get_next_shielded_address()` is intended solely for shielded address rotation. The function constructs the address request with the transparent receiver explicitly set to `Omit`, so the `full_unified` field of the returned value contains only the Sapling and Orchard receivers. However, this field is documented as "the full unified address (contains all available receivers)" in file `src/bindgen/mnemonic.rs`. Integrators should be aware of this discrepancy and must not rely on this function to obtain a full unified address or transparent receiver.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS