BLOCKSEC

Security Audit
Report for Shuttler

Date: August 28, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**

1.1 About Audit Target . . . 1
1.2 Disclaimer . . . 1
1.3 Procedure of Auditing . . . 2
1.3.1 Security Issues . . . 2
1.3.2 Additional Recommendation . . . 2
1.4 Security Model . . . 3

Chapter 2 Findings **4**

2.1 Security Issue . . . 5
2.1.1 Incorrect sender validation in function `received_sign_message()` . . . 5
2.1.2 Incorrect `threshold` used in the function `new_task()` . . . 7
2.1.3 Lack of rollback mechanism for removed `dkg_keys` . . . 8
2.1.4 Lack of filtered `db_round1` update in function `received_round1_packages()` 8
2.1.5 Lack of vault address validation in `Rune` deposit verification . . . 9
2.1.6 Lack of verification on the `signatures` in the function `aggregate()` with `SignWithGroupcommitment` mode . . . 11
2.1.7 Task ID collision risk due to inconsistent prefixing . . . 11
2.1.8 Incorrect block height persistence in `scan_txs_on_bitcoin()` . . . 12
2.1.9 Incorrect loop logic in function `DKGAdaptor::new_task()` . . . 12
2.1.10 Insufficient error handling logic for the function `aggregate()` . . . 13
2.1.11 Potential loss of funds due to disabled rune relaying flag . . . 14
2.1.12 Insufficient error handling logic after sending `MsgSubmitSignatures` . . . 14
2.1.13 Insufficient check on `packets.sender` in the function `received_round2_packages()` 15
2.1.14 Potential invalid task inserted into task list . . . 16
2.1.15 Lack of removing deprecated key shares in function `received_round2_packages()` 17
2.2 Recommendation . . . 18
2.2.1 Revise typos . . . 18
2.2.2 Redundant code . . . 19
2.2.3 Validate the existence of apps in the scheduled task . . . 19
2.3 Note . . . 20
2.3.1 Ensure trusted participants in the `swarm` network . . . 20

Report Manifest

Item	Description
Client	Bitway Labs
Target	Shuttler

Version History

Version	Date	Description
1.0	August 28, 2025	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-09-01 02:42:32+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 Audit Target

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

The target of this audit is the code repository [1] of Shuttler of Bitway Labs.

Shuttler is a client that synchronizes Bitcoin transactions and signs threshold signatures for Bitcoin transactions. The project implements the Frost protocol for distributed key generation (DKG) and threshold signing, supporting both Bitcoin bridge operations and lending protocols. It features a modular architecture with separate applications for bridge management, lending operations, and core TSS functionality. The system includes peer-to-peer communication, gossip protocols, and integration with Cosmos SDK chains.

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

- src/*

Other files are not within the scope of the audit. Additionally, all dependencies of the targets within the audit scope 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
shuttler	`Version 1`	`810d0938f4634afc7826eb95cc510217826d57a5`
	`Version 2`	`237fa312f5a228c82f0388cb8a68958d969edfcf`

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 war-

[1] https://github.com/bitwaylabs/shuttler

ranties on discovering all security issues of the audit targets, 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 audit targets.

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 audit targets with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of audit targets 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:

- **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.

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

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

Chapter 2 Findings

In total, we found **fifteen** potential security issues. Besides, we have **three** recommendations and **one** note.

- High Risk: 6
- Medium Risk: 4
- Low Risk: 5
- Recommendation: 3
- Note: 1

ID	Severity	Description	Category	Status
1	High	Incorrect sender validation in function `received_sign_message()`	Security Issue	Fixed
2	High	Incorrect `threshold` used in the function `new_task()`	Security Issue	Fixed
3	High	Lack of rollback mechanism for removed `dkg_keys`	Security Issue	Fixed
4	High	Lack of filtered `db_round1` update in function `received_round1_packages()`	Security Issue	Fixed
5	High	Lack of vault address validation in `Rune` deposit verification	Security Issue	Fixed
6	High	Lack of verification on the `signatures` in the function `aggregate()` with `SignWithGroupcommitment` mode	Security Issue	Fixed
7	Medium	Task ID collision risk due to inconsistent prefixing	Security Issue	Fixed
8	Medium	Incorrect block height persistence in `scan_txs_on_bitcoin()`	Security Issue	Fixed
9	Medium	Incorrect loop logic in function `DKGAdaptor::new_task()`	Security Issue	Fixed
10	Medium	Insufficient error handling logic for the function `aggregate()`	Security Issue	Confirmed
11	Low	Potential loss of funds due to disabled rune relaying flag	Security Issue	Confirmed
12	Low	Insufficient error handling logic after sending `MsgSubmitSignatures`	Security Issue	Confirmed
13	Low	Insufficient check on `packets.sender` in the function `received_round2_packages()`	Security Issue	Confirmed
14	Low	Potential invalid task inserted into task list	Security Issue	Fixed
15	Low	Lack of removing deprecated key shares in function `received_round2_packages()`	Security Issue	Fixed
16	-	Revise typos	Recommendation	Fixed

17	-	Redundant code	Recommendation	Fixed
18	-	Validate the existence of apps in the scheduled task	Recommendation	Fixed
19	-	Ensure trusted participants in the DKG process	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Incorrect sender validation in function `received_sign_message()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `received_sign_message()` function, validation of sender occurs after commitment or signature share data is written to storage. This ordering flaw allows invalid messages from unauthorized senders to be persisted.

```
239          if let Some(keypair) = ctx.keystore.get(&input.key) {
240
241              if !keypair.pub_key.verifying_shares().contains_key(sender) {
242                  error!("Sender {:?} not in keypair: {:?}", sender, input.key);
243                  return;
244              }
245
246              if !keypair.pub_key.verifying_shares().contains_key(&ctx.identifier) {
247                  debug!("My identifier {:?} not in participants", ctx.identifier);
248                  ctx.clean_task_cache(task_id);
249                  return;
250              }
```

Listing 2.1: src/protocols/sign.rs

```
157      match msg.package {
158          SignPackage::Round1(commitments) => {
159
160              let mut remote_commitments = ctx.commitment_store.get(&task_id).unwrap_or(BTreeMap
                     ::new());
161              // return if msg has received.
162              if let Some(exists) = remote_commitments.get(&first) {
163                  if exists.contains_key(&msg.sender) {
164                      return
165                  }
166              }
167
168              // merge received package
169              commitments.iter().for_each(|(index, incoming)| {
```

```
170                 match remote_commitments.get_mut(index) {
171                     Some(existing) => {
172                         existing.extend(incoming);
173                     },
174                     None => {
175                         remote_commitments.insert(*index, incoming.clone());
176                     },
177                 }
178             });
179 
180             ctx.commitment_store.save(&task_id, &remote_commitments);
181 
182             self.try_generate_signature_shares(ctx, &task_id, &msg.sender);
183 
184         },
185         SignPackage::Round2(sig_shares) => {
186 
187             let mut remote_sig_shares = ctx.signature_store.get(&task_id).unwrap_or(BTreeMap::
                    new());
188             // return if msg has received.
189             if let Some(exists) = remote_sig_shares.get(&first) {
190                 if exists.contains_key(&msg.sender) {
191                     return
192                 }
193             }
194 
195             // Merge all signature shares
196             sig_shares.iter().for_each(|(index, incoming)| {
197                 match remote_sig_shares.get_mut(index) {
198                     Some(existing) => {
199                         existing.extend(incoming);
200                     },
201                     None => {
202                         remote_sig_shares.insert(*index, incoming.clone());
203                     }
204                 }
205             });
206 
207             ctx.signature_store.save(&task_id, &remote_sig_shares);
208 
209             self.try_aggregate_signature_shares(ctx, &task_id, &msg.sender);
210 
211         }
```

Listing 2.2: src/protocols/sign.rs

```
375         if !keypair.pub_key.verifying_shares().contains_key(&ctx.identifier) {
376             debug!("My identifier {:?} not in participants.", &ctx.identifier);
377             ctx.clean_task_cache(task_id);
378             return;
379         }
380 
381         if !keypair.pub_key.verifying_shares().contains_key(sender) {
```

```
382             error!("Sender {:?} not in keypair: {:?}", sender, input.key);
383             return;
384         }
```

Listing 2.3: src/protocols/sign.rs

Impact This vulnerability may lead to incorrect signature shares and aggregated signatures.

Suggestion Validate the `msg.sender` before performing any database write operations.

2.1.2 Incorrect `threshold` used in the function `new_task()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description During the refresh process, participants removed from the previous `DKG` round reduce the active set. The threshold is lowered accordingly and a new refresh task is created with this adjusted threshold. However, the current implementation of `FROST` does not allow the decrement of the `threshold`, the sign task will fail if the amount of signature shares is less than the highest historical `threshold`. This may affect the `sign` task after `refresh` task.

In the file `sign.rs`, the function `try_aggregate_signature_shares()` will aggregate received signature shares based on the reduced `threshold`, potentially producing invalid `MsgSubmitSignatures`.

Besides, adding a check to ensure that the ratio between the `threshold` and the number of participants meets the required condition before creating the task is also necessary.

The same issue also exists in the function `new_task()` for signing.

```
319                         let input = RefreshInput{
320                             id: task_id.clone(),
321                             keys: dkg_keys,
322                             threshold: first_key_pair.priv_key.min_signers().clone() - 1,
323                             remove_participants: removed_ids,
324                             new_participants: participants,
325                         };
```

Listing 2.4: src/apps/lending.rs

```
312                         let input = RefreshInput{
313                             id: task_id.clone(),
314                             keys: vault_addrs,
315                             threshold: first_key_pair.priv_key.min_signers().clone() - 1,
316                             remove_participants: removed_ids,
317                             new_participants: participants,
318                         };
```

Listing 2.5: src/apps/bridge.rs

```
397         if input.participants.len() >= threshold {
398             signing_commitments.retain(|k, _| {input.participants.contains(k)});
399         }
```

```
400
401         if signature_shares.len() < threshold || signature_shares.len() < signing_commitments.
                len() {
402             return
403         }
```

Listing 2.6: src/protocols/sign.rs

Impact This issue introduces the inconsistency between the `shuttler` client and the `FROST` implementation on the `threshold`, leading to potential failure of the `sign` task.

Suggestion Revise the logic accordingly.

2.1.3 Lack of rollback mechanism for removed `dkg_keys`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The project issues tasks triggered by `Side Chain` events to manage participant rotation without changing vault addresses. In the `refresh` module, the `new_task()` function deletes the vault address and its associated key pair from the `keystore` if the current node is listed in `removed_participants`. However, this deletion occurs prior to the successful generation of a new key pair. If the refresh operation fails afterward, the original key material has already been removed, leaving the vault address unmanaged and inaccessible.

```
296                         if removed_ids.contains(&ctx.identifier) {
297                             dkg_keys.iter().for_each(|k| {ctx.keystore.remove(k);} );
298                             continue;
299                         }
```

Listing 2.7: src/apps/lending.rs

Impact Failure during the refresh process may cause vault addresses to become uncontrollable, posing significant risks including potential loss of user assets.

Suggestion Revise the logic to ensure that the old key pair is only removed from the node after the new key pair has been successfully generated during the refresh process.

2.1.4 Lack of filtered `db_round1` update in function `received_round1_packages()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the refresh module, the function `received_round1_packages()` receives the `Round1` packages from participants. It maintains a record of `(sender, data)` pairs in `db_round1`, avoiding duplicates. However, while the function later filters this data to retain only valid participants of the current refresh round, the filtered result is not persisted back to `db_round1`.

```
277         let mut local = ctx.db_round1.get(task_id).map_or(BTreeMap::new(), |v|v);
278         if local.contains_key(&packets.sender) {
279             // already received this sender's round1 package
280             warn!("duplicated round1 package from {:?}: {}", mem_store::get_moniker(&packets.sender
                    ), task_id);
281             return;
282         }
283         // merge packets with local
284         local.insert(packets.sender, packets.data);
285         ctx.db_round1.save(&task_id, &local);
```

Listing 2.8: src/protocols/refresh.rs

As a result, in the function `received_round2_packages()`, when loading `round1_packages` from `db_round1`, it may retrieve unfiltered entries including packages from nodes that are no longer valid participants.

```
388         let mut round1_packages = ctx.db_round1.get(task_id).unwrap_or(BTreeMap::new());
389
390         // frost does not need its own package to compute the threshold key
391         round1_packages.remove(&ctx.identifier);
392         // let round2_secret_package = match ctx.sec_round2.get(task_id) {
393         let round2_secret_package = match mem_store::get_dkg_round2_secret_packet(task_id) {
394             Some(secret_package) => secret_package,
395             None => {
396                 error!("No secret packet found for DKG: {}", task_id);
397                 return;
398             }
399         };
```

Listing 2.9: src/protocols/refresh.rs

Impact The refresh functionality may fail to execute properly due to unfiltered `Round1` packages from non-participants being used in later stages.

Suggestion Revise the logic to ensure that the filtered `Round1` packages are recorded in `db_round1`.

2.1.5 Lack of vault address validation in `Rune` deposit verification

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When the `Rune` relay feature is enabled, the system verifies `Rune` deposits based on the decoded `edict` data. However, the current validation logic only checks whether the deposited `Rune` amount is sufficient and does not ensure that the `Rune` was sent to the designated vault address. The output index specified in the `edict` is used to locate the destination of the `Rune` transfer, but no further checks are performed to confirm that this output corresponds to the expected vault address.

This omission introduces the risk of accepting Rune transfers directed to unintended or incorrect addresses, which may lead to inconsistencies between actual vault holdings on the Bitcoin network and their reflected state on the Side Chain.

```
378         let rune = match relayer.ordinals_client.get_rune(edict.id).await {
379             Ok(rune) => rune.entry.spaced_rune,
380             Err(e) => {
381                 error!("Failed to get rune {}: {}", edict.id, e);
382 
383                 // continue due to the deposit may be invalid
384                 // or this can be correctly handled for other relayers
385                 return true;
386             }
387         };
388 
389         // get the runes output
390         let output = match relayer
391             .ordinals_client
392             .get_output(OutPoint::new(tx.compute_txid(), edict.output))
393             .await
394         {
395             Ok(output) => output,
396             Err(e) => {
397                 error!(
398                     "Failed to get output {}:{} from ord: {}",
399                     tx.compute_txid(),
400                     edict.output,
401                     e
402                 );
403 
404                 // continue due to the deposit may be invalid
405                 // or this can be correctly handled for other relayers
406                 return true;
407             }
408         };
409 
410         // validate if the runes deposit is valid
411         if !bitcoin_utils::validate_runes(&edict, &rune, &output) {
412             debug!("Failed to validate runes deposit tx {}", tx.compute_txid());
413 
414             // continue due to the deposit is invalid
415             return true;
416         }
```

Listing 2.10: src/apps/relayer/bridge.rs

Impact Invalid deposits may be accepted, leading to asset mismatch or loss.

Suggestion Verify the edict's output address matches the vault address.

2.1.6 Lack of verification on the `signatures` in the function `aggregate()` with `SignWithGroupcommitment` mode

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `sign.rs`, the function `aggregate()` will try to aggregate the received signature shares by invoking the `FROST` api based on the sign mode. However, the current implementation of `SignWithGroupcommitment` mode does not validate the aggregated `signature` and just returns `OK(signature)` while the other modes will verify the aggregated signature. The unverified aggregated signatures will be submitted to the `Cosmos` under the `SignWithGroupcommitment` mode.

```
496       SignMode::SignWithGroupcommitment(group_commitment) => {
497           let frost_signature = frost_adaptor_signature::aggregate_with_group_commitment(&
                  signing_package, signature_shares, &keypair.pub_key, &group_commitment)?;
498           Ok(FrostSignature::Standard(frost_signature))
499       },
500       SignMode::SignWithAdaptorPoint(adaptor_point) => {
501           let frost_signature = frost_adaptor_signature::aggregate_with_adaptor_point(&
                  signing_package, signature_shares, &keypair.pub_key, adaptor_point)?;
502           Ok(FrostSignature::Adaptor(frost_signature))
503       }
```

Listing 2.11: src/protocols/sign.rs

Impact The unverified aggregated signatures will be submitted to the `Cosmos` under the `SignWithGroupcommitment` mode.

Suggestion Add verification of the aggregated signature in the `SignWithGroupcommitment` mode.

2.1.7 Task ID collision risk due to inconsistent prefixing

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `task_id` is used as a unique identifier for each task entry in the task database. However, the current implementation generates these `task_id`s inconsistently, using varying prefix formats across different modules. Some identifiers use simple prefixes (e.g., "`lending-`"), while others use more specific ones (e.g., "`lending-refresh-`", "`lending-dkg-`"). This design introduces the possibility of identifier collisions, especially when one prefix is a substring of another.

```
236                       let task= Task::new_signing(format!("lending-{}", id), "" , sign_inputs);
```

Listing 2.12: src/apps/lending.rs

```
318                         let task_id = format!("lending-refresh-{}", id);
```

Listing 2.13: src/apps/lending.rs

```
100                                 tasks.push(Task::new_dkg(format!("lending-dkg-{}", id),
                                        participants, threshold, batch_size));
```

Listing 2.14: src/apps/lending.rs

Impact The creation of new tasks may fail due to the collision of the `task_id`s.

Suggestion Standardize the construction of `task_id`s across all modules by enforcing unique and non-overlapping prefixes.

2.1.8 Incorrect block height persistence in `scan_txs_on_bitcoin()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `lending.rs`, the function `scan_txs_on_bitcoin()` is responsible for scanning `Bitcoin` blocks for deposit transactions. For each block height, it invokes the function `scan_bitcoin_txs_by_height()` to parse relevant transactions and then persists the scanned height using the function `save_last_scanned_height_bitcoin()`. However, the current implementation does not check whether relevant operations in scan were successful before updating the height. As a result, if the function `scan_bitcoin_txs_by_height()` fails or returns an error, the corresponding block height is still marked as processed. This causes the relayer to skip over blocks that may contain user deposit transactions,

```
133       scan_bitcoin_txs_by_height(relayer, height).await;
134       save_last_scanned_height_bitcoin(relayer, height);
```

Listing 2.15: src/apps/relayer/lending.rs

Impact Skipping blocks that contain user deposits will lead to loss of user assets.

Suggestion Revise the logic to ensure the block height is updated correctly.

2.1.9 Incorrect loop logic in function `DKGAdaptor::new_task()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `bridge.rs`, the function `DKGAdaptor::new_task()` iterates over a set of proposed participants to construct a valid participant map for a `DKG` task. The number of participants must meet or exceed the threshold requirement.

However, the loop breaks prematurely upon encountering the first peer not present in `live_peers`, instead of skipping that peer and continuing. This logic results in valid peers being excluded from the participant map.

```
93                              if !live_peers.contains(&identifier) {
94                                  break;
95                              }
```

Listing 2.16: src/apps/bridge.rs

Impact The participants map may not contain all qualified peers and the creation of the `DKG` task may fail.

Suggestion Replace `break` with `continue` to ensure all peers are checked and only invalid ones are skipped.

2.1.10 Insufficient error handling logic for the function `aggregate()`

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the file `sign.rs`, the function `aggregate()` will try to aggregate the received signature shares and construct the `Cosmos` message based on it. The current error handling logic just throws an error and returns afterwards. If the verification failure is caused by the malicious behavior of participants, the `shuttler` client should recognize the malicious node and block them in the following tasks. While the `cheater-detection` mechanism identifies malicious nodes in `SignWithTweak` and default modes, it becomes ineffective under `SignWithGroupcommitment` and `SignWithAdaptorPoint` modes due to cryptographic algorithm changes. As a result, malicious nodes remain undetected in subsequent tasks.

```
416          match aggregate(&signing_package, &signature_shares, &keypair, &input.mode) {
417              Ok(s) => {
418                  verifies.push(true);
419                  input.signature = Some(s);
420              },
421              Err(e) => {
422                  error!("aggregate error: {}", e);
423                  metrics::counter!("signing_failure").increment(1);
424                  return
425              }
426          };
```

Listing 2.17: src/protocols/sign.rs

Impact The malicious node will not be detected and blocked in the following tasks.

Suggestion Revise the error handling logic for the function `aggregate()` and implement the `cheater-detection` mechanism to detect malicious nodes in `SignWithGroupcommitment` and `SignWithAdaptorPoint` modes.

Feedback from the project The `cheater-detection` feature has not been enabled yet. This feature will be implemented in the future.

Clarification from BlockSec The project should ensure that all participants in this p2p network are honest since the `cheater-detection` mechanism is not implemented yet.

2.1.11 Potential loss of funds due to disabled rune relaying flag

Severity Low

Status Confirmed

Introduced by `Version 1`

Description When scanning `Bitcoin` blocks for deposits in the bridge module, the relayer attempts to identify `Rune` related deposits by checking whether any output script starts with the byte pattern `[OP_RETURN, OP_PUSHNUM_13]`. This pattern is used to mark `Rune` deposits. If it is `Rune` related deposits and `Rune` relay is disabled in the configuration, the relayer skips the entire transaction, meaning the `Bitcoin` deposit itself is also ignored and never synced to the `Side Chain`. This behavior can result in permanent loss of `Bitcoin` and `Rune` assets, despite the funds being correctly sent to the vault address.

```
361       if bitcoin_utils::is_runes_deposit(tx) {
362           if !relayer.config().relay_runes {
363               debug!("Skip the tx due to runes relaying not enabled");
364               return true;
365           }
```

Listing 2.18: src/apps/relayer/bridge.rs

Impact Users may irreversibly lose both `Bitcoin` and `Runes`.

Suggestion Relay `Bitcoin` deposits regardless of the `Rune` relay configuration. `Rune` metadata parsing can be conditionally skipped, but the `Bitcoin` deposit should always be processed and synced.

Feedback from the project The `BTC` vault output in the `Runes` deposit is as the protocol fee by our design. Thus, the simultaneous deposits for `BTC` and `Runes` are not supported. Meanwhile, the `BTC` deposits can be submitted to the chain by anyone if no relayer is active.

2.1.12 Insufficient error handling logic after sending `MsgSubmitSignatures`

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the file `bridge.rs` and `lending.rs`, the function `SignAdaptor::on_complete()` will send constructed `MsgSubmitSignatures` to `Cosmos` and mark the task status as completed. However, the `shuttler` client only logs errors without resubmission when transactions fail. This creates potential inconsistency in task results among participants.

```
258       let any = Any::from_msg(&msg)?;
259       ctx.tx_sender.send(any)?;
260
261       task.submitted = true;
262       // task.memo = to_base64(&psbt_bytes);
263       task.status = Status::Complete;
264       ctx.task_store.save(&task.id, &task);
265
```

```
266         anyhow::Ok(())
```

Listing 2.19: src/apps/bridge.rs

```
172     let (tx_sender, tx_receiver) = std::sync::mpsc::channel::<Any>();
173     let conf2 = conf.clone();
174     let identifier2 = identifier.clone();
175     spawn(async move {
176         while let Ok(message) = tx_receiver.recv() {
177             metrics::counter!("transaction_total").increment(1);
178             match send_cosmos_transaction(&identifier2, &conf2, message).await {
179                 Ok(resp) => {
180                     if let Some(inner) = resp.into_inner().tx_response {
181                         debug!("Submited {}, {}, {}", inner.txhash, inner.code, inner.raw_log);
182                         metrics::counter!("transaction_success").increment(1);
183                     };
184                 },
185                 Err(e) => {
186                     error!("Submit error: {:?}", e);
187                     metrics::counter!("transaction_failure").increment(1);
188                 },
189             };
190         }
191     });
```

Listing 2.20: src/apps/shuttler.rs

```
263         let cosm_msg = MsgSubmitSignatures {
264             id: task.id.replace("lending-", "").parse()?,
265             sender: ctx.conf.relayer_bitcoin_address(),
266             signatures ,
267         };
268         let any = Any::from_msg(&cosm_msg)?;
269         if let Err(e) = ctx.tx_sender.send(any) {
270             tracing::error!("{:?}", e)
271         }
```

Listing 2.21: src/apps/lending.rs

Impact Potential loss of critical messages, causing discrepancies in task state and consensus among network participants.

Suggestion Implement according logic for failed transaction submissions to ensure message delivery reliability.

Feedback from the project The feature which can handle missed signing requests has been implemented.

2.1.13 Insufficient check on `packets.sender` in the function `received_round2_packages()`

Severity Low

Status Confirmed

Introduced by Version 1

Description The function received_round2_packages() only checks if the sender has previously submitted a package, without verifying whether the sender is part of the expected participant set. As a result, a non-participant node can send a Round2 package after the actual Round2 process has completed. Since filtering occurs after insertion, the received set may incorrectly satisfy the completion condition, causing the protocol to re-enter the refresh finalization logic.

```
336     let mut received = ctx.db_round2.get(task_id).unwrap_or(BTreeMap::new());
337     if received.contains_key(&packets.sender) {
338         // already received this sender's round2 package
339         warn!("duplicated round2 package from {:?}: {}", mem_store::get_moniker(&packets.sender
                ), task_id);
340         return;
341     }
342     received.insert(packets.sender, round2_packages);
343     ctx.db_round2.save(&task_id, &received);
344
345     debug!("Received round2 packets: {} {:?}", task_id, received.keys().map(|k| mem_store::
            get_participant_moniker(k)).collect::<Vec<_>>());
```

Listing 2.22: src/protocols/refresh.rs

```
366     received.retain(|id, _| refresh_input.new_participants.contains(id));
367
368     if refresh_input.new_participants.len() == received.len() {
369
370         info!("#{} round2 completed", task_id);
```

Listing 2.23: src/protocols/refresh.rs

Impact This vulnerability allows non-participant nodes to repeatedly trigger the Round2 completion process, leading to unnecessary computation and the broadcasting of invalid Cosmos messages.

Suggestion Revise the logic accordingly.

Feedback from the project This is designed for efficiency and performance, because there are latencies between task creation of each node. If we reject these packages, we will also reject messages from expected participants that arrived earlier than local task creation.

2.1.14 Potential invalid task inserted into task list

Severity Low

Status Fixed in Version 2

Introduced by Version 1

Description In the sign module, the function new_task() reads events from the Side Chain to generate sign tasks. For each signer, it retrieves the corresponding participants and checks their liveness. If the number of live participants is zero, the task is supposed to be skipped. Otherwise, a sign_input is constructed and pushed into the inputs array.

However, the function does not check the length of the `inputs` array before creating a task. As a result, even when there are no live `participants` for signer, an empty inputs array may still result in a task being created and executed in later stages, which is incorrect.

```
202                         s.split(",").zip(h.split(",")).for_each(|(signer, sig_hash)| {
203                             let participants = mem_store::count_task_participants(ctx, &signer.
                                    to_string());
204                             if participants.len() > 0 {
205                                 let input = Input::new_with_message_mode(signer.to_string(),
                                        from_base64(sig_hash).unwrap(), participants, SignMode::
                                        SignWithTweak);
206                                 inputs.push(input);
207                             }
208                         });
209                         tasks.push( Task::new_signing(id.to_string(), "", inputs));
```

Listing 2.24: src/apps/bridge.rs

Impact The function `new_task()` may generate invalid tasks, leading to unnecessary resource consumption.

Suggestion Add a check to ensure that if the `inputs` array is empty, the corresponding sign task is not created.

2.1.15 Lack of removing deprecated key shares in function `received_round2_packages()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `refresh.rs` module, the function `received_round2_packages()` is responsible for receiving and processing the second round of data packets during the `DKG` refresh process. Within this function, after all participants' second-round data packets have been received and processed, the function `frost::keys::refresh::refresh_dkg_shares()` is invoked to compute and generate new key shares.

However, after the new key shares are generated, the current implementation lacks a clear mechanism to ensure that these no longer valid secret shares are permanently deleted from persistent storage, which is incorrect.

```
414             match frost::keys::refresh::refresh_dkg_shares(round2_secret_package, &
                    ith_round1_packages, round2_packages, old_key.pub_key, old_key.priv_key) {
415                 Ok((priv_key, pub_key)) => {
416                     keys.push((priv_key, pub_key));
417                 },
418                 Err(e) => {
419                     error!("Failed to compute threshold key: {} {:?}", task_id, e);
420                     metrics::counter!("refresh_failure").increment(1);
421                 }
422             };
```

Listing 2.25: src/protocols/refresh.rs

Impact If an attacker gains access to the storage medium and recovers these outdated key shares, they could potentially reconstruct the secret key once a sufficient number is collected, thereby compromising forward secrecy.

Suggestion Revise the logic to ensure the removal of deprecated key shares.

2.2 Recommendation

2.2.1 Revise typos

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Several typos exist in type names and enum variants across the codebase. These issues reduce code readability and may lead to confusion or errors in usage and maintenance.

For example, the debug info logs the submission of a transaction. The log message should be `Submitted`.

```
181                         debug!("Submited {}, {}, {}", inner.txhash, inner.code, inner.raw_log);
```

Listing 2.26: src/apps/shuttler.rs

Besides, the label used in the `metircs::counter!` should be `received_messages`.

```
263                     metrics::counter!("recieved_messages", "sender"=> message.source.unwrap_or(
                            propagation_source).to_string()).increment(1);
```

Listing 2.27: src/apps/shuttler.rs

The variable `signaure` represents the signature of a message. It should be renamed to `signature`.

```
453         let signaure = ctx.node_key.sign(raw, None).to_vec();
```

Listing 2.28: src/protocols/sign.rs

The struct `SignMesage` should be `SignMessage`

```
19pub struct SignMesage {
```

Listing 2.29: src/protocols/sign.rs

The error message `umatched input` should be `unmatched input`.

```
172             _ => return Err(DKGError("umatched input".to_string()))
```

Listing 2.30: src/protocols/dkg.rs

The exception message `Counld not create database!` should be `Could not create database!`.

```
42          let db_relayer = sled::open(conf.get_database_with_name("relayer")).expect("Counld not
                create database!");
```

Listing 2.31: src/apps/relayer/mod.rs

The struct `KeygenHander` should be `KeygenHandler`.

```
75pub struct KeygenHander{}
```

Listing 2.32: src/apps/bridge.rs

The type `Round1SecetStore` and `Round2SecetStore` should be `Round1SecretStore` and `Round2SecretStore`.

```
217pub type Round1SecetStore = DefaultStore<String, Vec<frost_adaptor_signature::keys::dkg::round1::
       SecretPackage>>;
```

Listing 2.33: src/apps/core.rs

```
218pub type Round2SecetStore = DefaultStore<String, Vec<frost_adaptor_signature::keys::dkg::round2::
       SecretPackage>>;
```

Listing 2.34: src/apps/core.rs

The constant `BLOCK_TOLERENCE` should be `BLOCK_TOLERANCE`.

```
  7pub const BLOCK_TOLERENCE: u64 = 5;
```

Listing 2.35: src/config/keys.rs

Suggestion Revise the typos accordingly.

2.2.2 Redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `bridge.rs`, the function `send_withdraw_tx()` constructs a `MsgSubmitWithdrawTransaction` with `relayer.config().relayer_bitcoin_address().to_string()`. However, the function `relayer_bitcoin_address()` already returns a `String`, making the additional `to_string()` conversion redundant.

```
522         sender: relayer.config().relayer_bitcoin_address().to_string(),
```

Listing 2.36: src/apps/relayer/bridge.rs

```
306     pub fn relayer_bitcoin_address(&self) -> String {
307         let pubkey = self.relayer_bitcoin_pubkey();
308         Address::p2wpkh(&pubkey, self.bitcoin.network).to_string()
309     }
```

Listing 2.37: src/config/mod.rs

Suggestion Remove the redundant code.

2.2.3 Validate the existence of apps in the scheduled task

Status Fixed in `Version 2`

Introduced by `Version 1`

Description During the initialization of the `shuttler` client, scheduled tasks are created to handle missed signing requests through the functions `handle_missed_tss_signing_request()` and `handle_missed_bridge_signing_request()`. These functions forward the missed tasks to the corresponding applications. However, there is no check to verify the existence of these applications before attempting to send the tasks. This may lead to unnecessary execution or potential runtime errors where specific apps are not enabled or present.

```
255             _ = ticker.tick() => {
256                 self.handle_missed_tss_signing_request(&mut context).await;
257                 self.handle_missed_bridge_signing_request(&mut context).await;
258             }
```

Listing 2.38: src/apps/shuttler.rs

Suggestion Implement an existence check for the applications prior to invoking the corresponding functions.

2.3 Note

2.3.1 Ensure trusted participants in the `swarm` network

Introduced by `Version 1`

Description Since the `cheater-detection` mechanism has not been implemented in the `Shuttler` project, it is essential to ensure that all participants in the swarm network are trusted. The governors should validate the identity of all nodes and banish malicious nodes from the network.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS