Security Audit Report

InterBTC Parachain Modules and Vault Client: Protocol Design and Source Code

2021/09/09

Last revised 2021/09/28

Authors: Josef Widder, Cezara Dragoi, Shon Feder

Contents

Audit overview

The Project

During our 2021/Q3 InterBTC audit, Interlay engaged Informal Systems to conduct a security audit over the documentation and the current state of the implementation of the new InterBTC fee model, and several crates that were out of scope of the audit in 2021/Q2.

InterBTC is a bridge between bitcoin and Polkadot. The link is achieved by BTC-Relay, which, according to its specification,

> acts a Bitcoin SPV/light client on Polkadot, storing only Bitcoin block headers and allowing users to verify transaction inclusion proofs. Further, it is able to handle forks and follows the chain with the most accumulated Proof-of-Work.

An economic fee model is included to align incentives of the actors with correct protocol execution.

Scope of this report

- Protocol completeness and compliance between code and specification
- Error handling and state validity
- Code organization and ease of review

The previous audit, in Q2 of 2021, focused on the main parts of the protocol and on auditing the source code of the `btc-relay`, `bitcoin`, and `vault-registry` crates. The current audit focused on the protocol aspects of the fee/sla and nomination protocols, as well as an audit of source code in the following crates:

crate	loc
vault (client)	4370
issue	1525
redeem	1868
replace	1467
refund	744
fee	733
sla	720
nomination	1088
TOTAL	12515

We conducted an audit, bounded in time to 3 person weeks. The audit was conducted by:

- Josef Widder (Principal Scientist)
- Shon Feder (Senior Software Engineer) and
- Cezara Dragoi (Principal Scientist).

This report covers the above tasks that were conducted between July, 12, 2021 and August, 17, 2021 by Informal Systems (the extended duration owing to overlap with vacation season).

Aims of audit

- Protocol completeness and compliance between code and specification
- Error handling and state validity
- Code organization and ease of review

Conducted work

Starting July 12, the Informal Systems team conducted an audit of the documentation and the code. For the protocol part, we reviewed the documentation of the fee model. Our team started with reviewing the papers that explain the underlying protocol as well as the documentation written by Interlay.

For the code review, Cezara Dragoi focussed on the off-chain software (vault-client) and Shon Feder focused on the onchain parts, that is, the remaining crates from the list above.

We set up a shared Github repository, wherein we documented the progress of the audit and collaborated with the Interlay team to record and refine our findings. We also used a shared Discord channel to exchange documents with preliminary findings, which we discussed during online meetings. In this summary document, we have distilled the central findings into uniquely identified findings, and reported specification-related suggestions.

Timeline

- 07/12/2021: Start
- 08/17/2021: Call to discuss findings of this audit
- 08/17/2021: End of audit
- 08/18/2021: Start of writing this report as PRs on a private Informal GitHub repo, shared with Interlay
- 09/08/2021: Complete draft of report fixed for final review by Interlay

Remark. Compared to the last report we had less meetings because many questions could be addressed in the shared Discord channel. Also we shared a Github repository with Interlay so that Interlay could observe our notes and progress throughout the audit. As a result there was no formal submission of intermediate reports.

Conclusions

Fee model at protocol level

The protocol/specification work in the initial scoping document was to consider the Fee/SLA and Nomination protocols. The Fee/SLA part changed significantly during the audit. In particular, the SLA was removed from the protocol. At this point in the design/development progress, we think that removal of the SLA was a good decision: During the Q2 audit, it was not entirely clear what operations should have what impact on the SLA of an individual agent. Also, the relevant calls into the SLA crate where spread over many protocols, and the intuition behind it was not always clear. The new fee model based on the "stake" (that is, the backed interBTC) is simple and appears robust. The protocol might still be refined with the reintroduction of SLAs at later stages of the project.

Overall, we found the documentation of the fee model and the economic incentives very clear. But we provide some suggestions for clarifying the documentation and to make economic implications more explicit.

Vault client

In the course of this audit, we also reviewed the off-chain part of the bridge, that is, the vault client. In contrast to the on-chain software, there was no specification for the vault client. Thus part of the audit consisted in reverse-engineering the design. We had a meeting with Interlay to align our understanding.

We found one medium- and one high-severity problem, detailed in the findings. These highlight the need for a system-level specification of the functionality, which would be helpful for evaluating the correctness of behavior and a prerequisite for more formal verification of the bridge. We propose a way to approach this in IF-INTERLAY2-SPEC.

On-chain crates

Regarding the crates that implement on-chain functionality, we found the code well organized, well documented, and faithful to the specification. In the previous audit we highlighted a number of issues regarding code quality, data representation, and code organization. We observed virtually none of these issues in the artifacts under review in this phase. We attribute this difference in quality to three principle factors:

- The Interlay team's rewrite of the interbtc specifications to adopt a pre- and post-condition style made the specs much clearer and more accurate.
- The parts of the code under review at this phase were more recent additions to the code base. We suspect the improved clarity in the code reflects maturation of the team's development practices.
- The on-chain code in scope for this review was generally less critical and less complex: most of the functionality only involves piping data and performing validation checks.

Nonetheless, we identified some divergence between the specification and the implementation. This is not unexpected for a project of this complexity which is still under active development. These discrepancies are detailed in the relevant findings and have all been addressed at the time this report was finalized. We also reported recommendation-level findings on 3 minor matters regarding clarity and condition of the code.

Further Increasing Confidence

The scope of this audit was limited to manual code review and manual analysis and reconstruction of the protocols. To further increase confidence in the protocol and the implementation, we recommend following up with more rigorous formal measures, including automated model checking and model-based adversarial testing. Our experience shows that incorporating test suites driven by TLA+ models that can lead the implementation into suspected edge cases and error scenarios enables discovery of issues that are unlikely to be identified through manual review.

It is our understanding that the Interlay team intends to pursue such measures to further improve the confidence in their system.

Audit Dashboard

Target Summary

- **Type**: Specification and Implementation
- **Platform**: Rust
- **Artifacts**
  - interbtc-clients/vault @ `0.8.0`
  - interbtc/crates/issue @ `0.8.3`
  - interbtc/crates/redeem @ `0.8.3`
  - interbtc/crates/replace @ `0.8.3`
  - interbtc/crates/refund @ `0.8.3`
  - interbtc/crates/fee @ `0.8.3`
  - interbtc/crates/sla @ `0.8.3`
  - interbtc/crates/nomination @`0.8.3`
  - interbtc-spec @ `5.2.1`
  - interbtc-spec @ `5.4.0`

Engagement Summary

- **Dates**: 07/12/2021 - 08/17/2021
- **Method**: Manual review
- **Employees Engaged**: 3
- **Time Spent**: 3 person-weeks

Coverage

Fee model at protocol level

Informal Systems manually reviewed the following artifacts:

- fees @ `5.4.0`
- economic incentives @ `5.4.0`
- staking @ `5.4.0`
- reward @ `5.4.0`
- nomination @ `5.4.0`
- Scalable Reward Distribution on Ethereum Blockchain (for reference, external documentation)
- Scalable Reward Distribution with Changing Stake Sizes (for reference, external documentation)

The review resulted in the comments on the specification..

Vault client

We reviewed

- interbtc-clients/vault @ `0.8.0`

resulting in the following findings and recommendations:

- IF-INTERLAY2-EXPIRATION
- IF-INTERLAY2-VAULT-SPEC
- IF-INTERLAY2-SPEC

Several other potential findings were identified but cleared with the help of the Interlay team. In more detail, we have reverse-engineered the code at commit 2caf0a1 and derived a more abstract specification by over-approximating the behavior of some components. However, in the absence of an actual specification of the vault client, to distinguish a bug from an expected behavior we had to interact with the development team. Reasoning about the `issue_set` was particularly intricate: this set is not kept synchronized between the vault and the Parachain. It turned out that this is not important for safety, because the cancellation task on the vault uses a different image of the `issue_set`, which is built from the Parachain, and not from the local `issue_set` maintained by the vault.

On-chain crates

We reviewed the following InterBTC parachain crates, alongside the specification at the listed tags:

- interbtc/crates/issue @ `0.8.3`
- interbtc/crates/redeem @ `0.8.3`
- interbtc/crates/replace @ `0.8.3`
- interbtc/crates/refund @ `0.8.3`
- interbtc/crates/fee @ `0.8.3`
- interbtc/crates/sla @ `0.8.3`
- interbtc/crates/nomination @ `0.8.3`
- interbtc-spec @ `5.2.1`
- interbtc-spec @ `5.4.0`

Yielding the following findings and recommendations:

- IF-INTERLAY2-CMP.md
- IF-INTERLAY2-FEE
- IF-INTERLAY2-ISSUE
- IF-INTERLAY2-MINTING

- IF-INTERLAY2-NOMINATION
- IF-INTERLAY2-REDEEM
- IF-INTERLAY2-REFUND
- IF-INTERLAY2-REPLACE
- IF-INTERLAY2-STORAGE

See the overview of the On-chain crates and the listed items for details.

Recommendations

This section aggregates all the recommendations made during the audit. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Middle term

- We suggest to add documentation and specification of the vault client. In the distributed setting, operations take several steps executed by different entities, e.g., by the Parachain, by the bitcoin chain or by the vault. The specification should highlight the result (e.g., success/failure and the expected side effects) for all partial executions of these steps. Such a specification (1) would have helped in determining without the help of the Interlay team the relation between the issue sets manipulated by the vault during either issuing or canceling requests and the issue set manipulated by the Parachain and (2) it is a prerequisite for formal verification. See IF-INTERLAY2-VAULT-SPEC and IF-INTERLAY2-SPEC.

To ensure atomicity of the cancel request: Clarify the specification of canceling a request. In the implementation, call the RPC only after making sure the parachain will accept the cancellation, as per IF-INTERLAY2-EXPIRATION.

Long term

- Clarify economics with respect to scale (total amount of scale) in the fee model (more details here).
- Clarify economics and incentives to submit block headers (more details here).
- Make the structure in the fee model documentation more coherent and complete. We give more detailed recommendations here.

Specification comments on the economic model

In this section we comment on the specification and the documentation. As such, the suggestions in this sections do not constitute findings that need to be fixed in short-term, but should be understood as suggestions how the documentation can be further improved in the long-run, in order to help users and adopters of the technology to get a quick understanding of the underlying concepts and the inherent involved economic risk.

Interlay reported that several of the points have been addressed before this report has been published. Those are marked with - **fix:** (link to PR)

Involved artifacts

- fees
- economic incentives
- staking
- reward
- nomination
- Scalable Reward Distribution on Ethereum Blockchain (for reference, external documentation)
- Scalable Reward Distribution with Changing Stake Sizes (for reference, external documentation)

Overview

The protocol/specification work in the initial scoping document was to consider the Fee/SLA and Nomination protocols. In particular the Fee/SLA part changed significantly during the audit. In particular the SLA was removed from the protocol.

In this report we provide feedback on the fee model of version 5.4.0 where the fee distribution is based on the following two papers - Scalable Reward Distribution on Ethereum Blockchain - Scalable Reward Distribution with Changing Stake Sizes

These papers are not peer-reviewed. We could not follow all details of the math there but the proposed solution seems to address the problem adequately.

As we did some preliminary review of the specifications during the first week of the audit, we also include here some comments about version 5.2.1 that might be relevant in the future in case SLA returns.

Comments

The new fee model

General comments

At this point on the design/development progress, we think that removing of SLA was a good decision. At the stage of the Q2 audit, it was not so clear what operations should have what impact of the SLA of an individual agent. Also the relevant calls into the SLA crate where spread over many protocols, and the intuition behind was not always clear. The new fee model based on the "stake", that is, the backed interBTC, is simple and appears robust. It might still be refined with SLA at later stages of the project.

We find the documentation of the fee model and the economic incentives very clear.

Some comments that might deserve more attention:

- "The reward is influenced by the total of all stakes.". Is there a theoretical limit when the total stake is so high that it is not profitable anymore to join, participate, or add collateral? Is there an assumption that the higher the stake the higher will be the frequency will be on cross-chain transaction that will pay fees?

- The overview in the staking spec discusses that slashing is handled in a dual way to fees (cf. `SlashPerToken` and `SlashTally`). It seems that a discussion on these concepts should also appear in the fee documentation as currently the fee documentation seems to serve as a central point for all economical matters.
- Similarly, the discussion about vaults opting out of nomination and possibly opting in again before all nominators have claimed their rewards would deserve more precise discussion.

Detailed comments to help clarifying the documentation

- It seems that this description does not capture collateral
  - **fix:** https://github.com/interlay/interbtc-spec/pull/63.
- The figure is a bit unclear about the semantics of the arrows. In particular those that change color. During the meeting it was discussed that those are implemented using a substrate mechanism that allows to activate/deactivate funds. It would be great to clarify that.
- Given that the relayer functionality is now part of the vault, it might make sense to reflect that in the description
- The term primary fee is a bit unclear. Is there secondary? (Primary and secondary income is used in the incentives. Do these terms correspond?)
- The usage of the term "stake" in this paragraph is to match it with the papers. This could be clarified as there is possible ambiguity with the term collateral: If a vault backs 100 interBTC it needs to be over collateralized, say to the value of 120 BTC. If there is a problem, it might be slashed these 120. So once may say that the vault has 120 "at stake", while in the context of payout the "stake" used is 100. Perhaps move the equation already up to this discussion.
- typo: should be "requested"
  - **fix:** https://github.com/interlay/interbtc-spec/pull/64
- The comment about self-redeem seems important for operators, but looks a bit out of context here. Is this point discussed somewhere else?
- The example could be clarified. In order to better explain the equation, instead of saying the vault has 50% stake, say the total stake is 200 and the vault's stake is 100.
  - **fix:** https://github.com/interlay/interbtc-spec/pull/66
- The parenthesis "i.e., the actually locked BTC" is a bit imprecise. It would only be precise of the issued interBTC are always equal to the locked ones, which is not the case in the middle of issue/redeem operations.
  - **fix:** https://github.com/interlay/interbtc-spec/pull/67
- The section on "fee payouts" would become clearer if it started with the three sentences following "The payouts are based on the pull-based. . . "
- At this point it would be good to discuss how the grieving payout works. That is
  - transfer done on `cancel`
  - what happens on `issue`, `redeem`, etc.
- Is the transfer done by the redeem function? More general, in the sections
  - Premium Redeem Fee
  - Punishment Fees
  - Theft Fee it might make sense to add a second bullet point (after the Fee bullet) called "Payout Event" and discuss on what function call the payout actually happens.
- The section on Arbitrage, strictly speaking, is not about fees. The question is, whether it would make sense to rename the whole document to "Economic model" to capture the more broader content.
- We suggest some additions for clarification, namely that
  - issuedTokens are actually held by users

  - lockedCollateral was previously held by a vault

- We would suggest to write that "Anyone can exchange their interBTC against COL" instead of "can now burn..."
- "might not work as intended" is an unfortunate wording. Perhaps clarify using "as expected" and mention exchange rate fluctuations if appropriate.
- it is not clear how the reward_tally can be negative.
- Perhaps "slashed collateral" should be replaced with "slashed grieving collateral" here and here for clarity. Also since the grieving collateral in "replace" is paid from vault to vault, it may also appear in the vault's "internal costs".
- perhaps it should also be mentioned here that the functionality of the relayer is now considered part of the vault.
- The user may be referred to as "they" rather than "he".
  - **fix:** https://github.com/interlay/interbtc-spec/pull/65

Comments on spec

- It would be good to discuss a bit which crates (functions) call into Reward
  - **fix:** https://github.com/interlay/interbtc-spec/pull/39

Comments on the older version

- Compared to the previous version of the incentives document, the new one provides a better-structured and more complete overview of the risks/economics of the different roles. The section on "Processes" has been removed. It appears that the relevant discussion anyway appear in other places of the documentation.

Nomination

Overall the discussion is quite clear. However, the location within the specification part might not be optimal. We suggest that more high-level discussions about nomination also should be put into the economics section.

Detailed comments to help clarifying the documentation

- The three points under this one do not seem to be protocol steps:
  - line 49 seems to be a safety property
  - line 50 seems to be somewhat a safety property (the question is until when it is locked?)
  - line 51 is part of a correctness argument it seems.
- We suggest to make the distinction according to cases (a) under-collateralization and (b) theft:
- It is not so clear how the value is determined here. Is it in a specific currency at a given time?
- It is unclear what is the intention of "100% *" in this formula. Also perhaps add "effective collateralization = 100%..." in the formula.
- the discussion about premium redeem seems a bit out of context here.
- as above, it might make sense to rename the section Propostional Slashing, e.g., to "under-collateralization failure".
- "if you modify my example..." discussion is not so clear.

Issue discussed in collaboration with Interlay

The economic model faces the free rider problem: There are no explicit incentives to submit block headers. This issue should be addressed in future versions of the protocol.

Findings

ID	Title	Type	Severity	Status
IF-INTERLAY2-EXPIRATION	Possible disagreement on expiration status from request cancellation	Implementation	High	Resolved
IF-INTERLAY2-MINTING	Vault not banned precondition not enforced on minting tokens	Implementation	Medium	Resolved
IF-INTERLAY2-FEE	Discrepancies between implementation and spec in `fee` crate	Documentation	Low	Resolved
IF-INTERLAY2-ISSUE	Discrepancies between implementation and spec in `issue` crate	Documentation	Low	Resolved
IF-INTERLAY2-NOMINATION	Discrepancies between implementation and spec in `nomination` crate	Documentation	Low	Resolved
IF-INTERLAY2-REDEEM	Discrepancies between implementation and spec in `redeem` crate	Documentation	Low	Resolved
IF-INTERLAY2-REFUND	Discrepancies between implementation and spec in `refund` crate	Documentation	Low	Resolved
IF-INTERLAY2-REPLACE	Discrepancies between implementation and spec in `replace` crate	Documentation	Low	Resolved
IF-INTERLAY2-VAULT-SPEC	Vault client is undocumented and unspecified	Documentation	Low	In Progress
IF-INTERLAY2-CMP	Use `cmp` for clearer case analysis of inequalities	Implementation	Rec	Unresolved
IF-INTERLAY-SPEC	Specification of Concurrent Behaviors	Documentation	Rec	In Progress
IF-INTERLAY2-STORAGE	Redundant lookups in Substrate storage	Implementation	Rec	Resolved

Severity Categories

Severity	Description
High	The issue is an exploitable security vulnerability
Medium	The issue is a security vulnerability that may not be directly exploitable or may require certain complex conditions in order to be exploited
Low	The issue is objective in nature, but the security risk is relatively small or does not represent security vulnerability
Rec	No security vulnerability or immanent risk is identified, but an improvement is recommended

IF-INTERLAY2-CMP: Use `cmp` for clearer case analysis of inequalities

Severity	Recommendation
Type	Implementation
Difficulty	Easy
Status	Unresolved

Involved artifacts

- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/issue/src/lib.rs#L373-L430

Description

The logic in issue/src/lib.rs#L373-L430 is relatively intricate. There are three cases to be handled, corresponding to the possible outcomes of the inequality, but the current code structure requires careful analysis to uncover this trichotomy.

Problem Scenarios

When all three possible outcomes of an inequality check must be accounted for, then use of the standard operators requires writing and reading three different tests (`a > b`, `a == b`, `a < b`) and imposes the accompanying complication of chained `if/else` statements. In general, code that is harder to read and reason about is more likely to lead to errors during maintenance or development.

Recommendation

Instead, we can make such trichotomies readily apparent to the reader via cmp:

```
match amount_transferred.cmp(&expected_total_amount) {
    // release griefing collateral
    Ordering::Equal => ext::collateral::release_collateral::<T>(&requester,
    ↪  issue.griefing_collateral)?,
    Ordering::Less => {
        // handle less case,
    },
    Ordering::Greater => {
        ext::collateral::release_collateral::<T>(&requester, issue.griefing_collateral)?,
        // handle greater case
    }
}
```

In the event that use of `cmp` is not feasible due to the need to maintain `no_std` compatibility but the team finds this kind of readability improvement sufficiently compelling, they might consider use of no_std_compat.

IF-INTERLAY2-EXPIRATION: Possible disagreement on expiration status from request cancellation

Severity	High
Type	Implementation
Difficulty	Medium
Status	Resolved by interbtc-clients@73bb0

Involved artifacts

The vault client implementation: Canceling requests.

- https://github.com/interlay/interbtc-clients/blob/2caf0a1079a69a6299dbc416dd48b3467345c8a4/vault/src/system.rs#L454
- https://github.com/interlay/interbtc-clients/blob/2caf0a1079a69a6299dbc416dd48b3467345c8a4/vault/src/system.rs#L461
- https://github.com/interlay/interbtc-clients/blob/2caf0a1079a69a6299dbc416dd48b3467345c8a4/vault/src/cancellation.rs#L191
- https://github.com/interlay/interbtc-clients/blob/2caf0a1079a69a6299dbc416dd48b3467345c8a4/vault/src/cancellation.rs#L205

Description

The function `cancel_request(&self.parachain_rpc, request.id)` is called when a request is expired (where the expiration is determined by the function `drain_expired`). `cancel_request` does an RPC, `cancel_issue`, `cancel_replace`, etc., on the parachain, without waiting for the parachain to reach the same `bitcoin_height` as the vault.

Problem Scenarios

Because of the delay induced by relaying bitcoin blocks from the vault (relayer) to the parachain, a request might be expired from the vault's perspective, but **not** from the parachain client perspective, hence the `cancel_request` could fail on the parachain.

As implemented, the cancellation takes effect when, from the vault's perspective, it has expired, even though, from the parachain's perspective, it may not be expired.

Recommendation

The specification of canceling a request should ensure that a request is expired iff it is expired on both the vault and the parachain.

In more detail:

- Specification: Clarify the specification of canceling a request.
- Implementation: Call the RPC only after making sure the parachain will accept the cancellation.

IF-INTERLAY2-FEE: Discrepancies between implementation and spec in `fee` crate

Severity	Low
Type	Documentation
Difficulty	Easy
Status	Resolved by various

Involved artifacts

- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/fee/src/lib.rs
- https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/fee.rst

Description

The following discrepancies between the specification and the implementation of the `fee` crate were found:

`fn distribute_rewards`

Discrepancy in maintainer fund increase

- **fix**: interbtc-spec#25
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/fee.rst#L127

The specification calls for the maintainer fund to be increased by the amount multiplied by the maintainer rewards:

```
127         The Maintainer fund MUST increase by ``amount * MaintainerRewards``.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/fee/src/lib.rs#L303-L305

However, the implementation does not reflect this multiplication:

```
303         let maintainer_rewards = amount.saturating_sub(vault_rewards);
304         let maintainer_account_id = Self::maintainer_account_id();
305         ext::treasury::transfer::<T>(&Self::fee_pool_account_id(), &maintainer_account_id,
            ↪  maintainer_rewards)?;
```

`fn withdraw_rewards`

Specified signature missing `vault_id` argument

- **fix:** interbtc-spec#53
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/fee.rst#L139-L145

```
*Function Signature*

``withdrawRewards(account)``

*Parameters*

* ``account``: the account withdrawing ``interBTC`` rewards.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/fee/src/lib.rs#L263-L270

```
    /// Withdraw all rewards from the `origin` account in the `vault_id` staking pool.
    ///
    /// # Arguments
    ///
    /// * `origin` - signing account
    #[pallet::weight(<T as Config>::WeightInfo::withdraw_rewards())]
    #[transactional]
    pub fn withdraw_rewards(origin: OriginFor<T>, vault_id: T::AccountId) ->
    ↪  DispatchResultWithPostInfo {
```

Implementation preconditions unspecified

- **fix:** interbtc-spec#53
- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/fee/src/lib.rs#L271-L272

The implementation entails preconditions that (1) the parachain must be running and (2) that the request must be signed by the nominator who requested the withdrawal:

```
        ext::security::ensure_parachain_status_not_shutdown::<T>()?;
        let nominator_id = ensure_signed(origin)?;
```

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/fee.rst#L151-L153

However, these are conditions not included in the specified preconditions:

```
*Preconditions*

* The ``account`` MUST have available rewards for ``interBTC``.
```

Note that these two preconditions show up in most other function specifications, so it is worth including for consistency.

Problem Scenarios

Such discrepancies can contribute to miscommunication, divergent understandings, inability to reason effectively about the system, and even clashing logic in different parts of the code base. Insofar as the spec is meant to guide alternative implementations, such discrepancy can contribute to incorrect implementations.

Recommendation

Correct the spec or the implementation, as appropriate, to bring them into alignment.

IF-INTERLAY2-ISSUE: Discrepancies between implementation and spec in `issue` crate

Severity	Low
Type	Documentation
Difficulty	Easy
Status	Resolved by various

Involved artifacts

- **parachain issue spec**: https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/issue.rst
- **parachain issue implementation**: https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/issue/src/lib.rs

Description

The following discrepancies between the specification and the implementation of the `issue` crate were found:

Name of `IssueRequest` struct

- **fix:** https://github.com/interlay/interbtc-spec/pull/48

The struct is called `Issue` in the spec but `IssueRequest` in the implementation.

Discrepancies with spec for `request_issue` function

The vault is not banned

- **fix:** https://github.com/interlay/interbtc-spec/pull/56

This precondition occurs in the implementation

```
278         // Check that the vault is currently not banned
279         ext::vault_registry::ensure_not_banned::<T>(&vault_id)?;
```

https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/issue/src/lib.rs#L277-L278

but is not listed in the specified preconditions.

`amount_requested >= dust_value`

- **fix:** https://github.com/interlay/interbtc-spec/pull/22

This precondition occurs in the implementation

```
290         // only continue if the payment is above the dust value
291         ensure!(
292             amount_requested >= Self::issue_btc_dust_value(),
293             Error::<T>::AmountBelowDustAmount
294         );
```

https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/issue/src/lib.rs#L290-L294

but is not listed in the specified preconditions.

Discrepancies with spec for `execute_issue` function

`IssueAmountChange` event that is only emitted on some conditions is specified as unconditional

- **fix:** https://github.com/interlay/interbtc-spec/pull/56
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/issue.rst#L178

  The spec lists the `IssueAmountChange` event as if it were unconditionally emitted

```
*Events*

* :ref:`executeIssueEvent`
* :ref:`issueAmountChangeEvent`
```

In the implementation, this event is only emitted if either:

- **`amount_transferred < expected_total_amount`:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/issue/src/lib.rs#L401
- **or `amount_transferred > expected_total_amount && vault is not liquiated && increase of issued tokens succeeds`:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/issue/src/lib.rs#L416

To illustrate the logic, here is the structure of the conditional, with irrelevant sections of the code omitted:

```
        // check for unexpected bitcoin amounts, and update the issue struct
        if amount_transferred < expected_total_amount {
            // [snip]

            Self::update_issue_amount(&issue_id, &mut issue, amount_transferred,
            ↪  slashed_collateral)?;
        } else {
            // release griefing collateral
            ext::collateral::release_collateral::<T>(&requester, issue.griefing_collateral)?;

            if amount_transferred > expected_total_amount
                && !ext::vault_registry::is_vault_liquidated::<T>(&issue.vault)?
            {
                // [snip]

                match ext::vault_registry::try_increase_to_be_issued_tokens::<T>(&issue.vault,
                ↪  surplus_btc) {
                    Ok(_) => {
                        // Current vault can handle the surplus; update the issue request
                        Self::update_issue_amount(&issue_id, &mut issue, amount_transferred,
                        ↪  0u32.into())?;
                    }
                    Err(_) => {
                        // [snip]
                    }
                }
            }
        };
```

Note that when `amount_transferred = expected_total_amount`, the event will **not** be emitted.

The spec should explicitly note that this event is only conditional.

Specs gives postcondition of `issue.status` set to `Completed` as conditional

- **fix:** https://github.com/interlay/interbtc-spec/pull/28
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/issue.rst#L193-L200

  The spec only gives the postcondition for the `issue.status` as a conditional outcome depending on the amount transferred:

```
* If the amount transferred IS less than the ``issue.amount + issue.fee``:

    * The ``executor`` MUST be the account that made the issue request.
    * The Vault's ``toBeIssuedTokens`` MUST decrease by the deficit (``issue.amount - amountTransferred``).
    * The ``issue.fee`` MUST be updated to the amount transferred multiplied by the :ref:`issueFee`.
    * The ``issue.amount`` MUST be set to the amount transferred minus the updated ``issue.fee``.
    * The Vault's free balance MUST increase by the ``griefingCollateral``.
    * The ``issue.status`` MUST be set to ``Completed``.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/issue/src/lib.rs#L461

  However, in the implementation, this status is set under any condition that ends with a successful dispatch result.

Discrepancies with spec for `cancel_issue` function

postcondition `decrease_to_be_issued_tokens` unspecified

- **fix:** https://github.com/interlay/interbtc-spec/pull/35
- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/issue/src/lib.rs#L490-L496

  The implementation entails the post condition that the vault's "to be issued" tokens are decreased:

```
// Decrease to-be-redeemed tokens:
let full_amount = issue
    .amount
    .checked_add(&issue.fee)
    .ok_or(Error::<T>::ArithmeticOverflow)?;

ext::vault_registry::decrease_to_be_issued_tokens::<T>(&issue.vault, full_amount)?;
```

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/issue.rst#L249-L259

  But this is not included in the specified postconditions:

```
*Postconditions*

* If the vault IS liquidated:

    * The requester's free balance MUST increase by the ``griefingCollateral``.
```

```
* If the Vault IS NOT liquidated:

    * The vault's free balance MUST increase by the ``griefingCollateral``.

* The issue status MUST be set to ``Cancelled``.
```

Problem Scenarios

Such discrepancies can contribute to miscommunication, divergent understandings, inability to reason effectively about the system, and even clashing logic in different parts of the code base. Insofar as the spec is meant to guide alternative implementations, such discrepancy can contribute to incorrect implementations.

Recommendation

Correct the spec or the implementation, as appropriate, to bring them into alignment.

IF-INTERLAY2-MINTING: Vault not banned precondition not enforced on minting tokens

Severity	Medium
Type	Implementation
Difficulty	Easy
Status	Resolved by interbtc#227

Involved artifacts

- https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L361
- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L594

Description

A specified precondition of the `mint_tokens_for_reimbursed_redeem` function is that the vault not be banned

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L361

  The follow precondition is specified

```
361 * The vault MUST NOT be banned.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L594

  But this was not enforced in the body of the function, nor in its subroutines.

Problem Scenarios

A banned vault could mint tokens to reimburse a redeem request.

Recommendation

A call to `ensure_not_banned` (https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/ext.rs#L125-L127) should be added to the routine.

IF-INTERLAY2-NOMINATION: Discrepancies between implementation and spec in `nomination` crate

Severity	Low
Type	Documentation
Difficulty	Easy
Status	Resolved by interbtc-spec#52 et al.

Involved artifacts

- https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/nomination.rst
- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/nomination/src/lib.rs

Description

The following discrepancies between the specification and the implementation of the `nomination` crate were found:

Specified `Nominator` struct is not implemented

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/nomination.rst#L175

  A `Nominator` struct is specified, but no corresponding struct was found in the implementation.

`fn set_nomination_enabled`

unclear specification of precondition allowing function to be called by "a passed governance referendum"

- **fix:** https://github.com/interlay/interbtc-spec/pull/59/commits/b0e2cb4a9c41a0971b22da055daf6d2a3de736ff
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/nomination.rst#L230

The specified precondition allows the function to be called by either the "root" or from "a passed governance referendum":

```
230 The calling account MUST be root or the function MUST be called from a passed governance
    ↪ referendum.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/nomination/src/lib.rs#L146-L150

  However, only the root account is actually able to invoke the function:

```
146 pub fn set_nomination_enabled(origin: OriginFor<T>, enabled: bool) ->
    ↪ DispatchResultWithPostInfo {
147             ensure_root(origin)?;
148             <NominationEnabled<T>>::set(enabled);
```

```
            Ok(().into())
        }
```

This gave the appearance an implementation oversight. However, Greg Hill clarified that

> Root is the system-level caller (i.e. superuser), the democracy pallet for instance is able to dispatch calls from this origin. See: https://substrate.dev/docs/en/knowledgebase/runtime/origin

And the spec was amended to clarify.

fn opt_in_to_nomination

unspecified precondition: nomination must be enabled

- **fix:** https://github.com/interlay/interbtc-spec/pull/59/commits/b0e2cb4a9c41a0971b22da055daf6d2a3de736ff
- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/nomination/src/lib.rs#L265

```
ensure!(Self::is_nomination_enabled(), Error::<T>::VaultNominationDisabled);
```

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/nomination.rst#L259-L263

```
*Preconditions*

* The BTC Parachain status in the :ref:`security` component MUST be ``RUNNING:0``.
* A Vault with id ``vaultId`` MUST be registered.
* The Vault MUST NOT be opted in.
```

fn opt_out_of_nomination

unspecified precondition: vault's nominated collateral must be non-zero

- **fix:** https://github.com/interlay/interbtc-spec/pull/59/commits/b0e2cb4a9c41a0971b22da055daf6d2a3de736ff
- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/nomination/src/lib.rs#L281-L284

```
ensure!(
            Self::get_total_nominated_collateral(vault_id)?.is_zero(),
            Error::<T>::HasNominatedCollateral
        );
```

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/nomination.rst#L292-L297

```
*Preconditions*

* The BTC Parachain status in the :ref:`security` component MUST be ``RUNNING:0``.
* A Vault with id ``vaultId`` MUST be registered.
* A Vault with id ``vaultId`` MUST exist in the ``Vaults`` mapping.
```

Problem Scenarios

Such discrepancies can contribute to miscommunication, divergent understandings, inability to reason effectively about the system, and even clashing logic in different parts of the code base. Insofar as the spec is meant to guide

alternative implementations, such discrepancy can contribute to incorrect implementations.

Recommendation

Correct the spec or the implementation, as appropriate, to bring them into alignment.

IF-INTERLAY2-REDEEM: Discrepancies between implementation and spec in `redeem` crate

Severity	Low
Type	Documentation
Difficulty	Easy
Status	Resolved by various

Involved artifacts

- https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst
- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs

Description

The following discrepancies between the specification and the implementation of the `issue` crate were found.

All discrepancies were addressed, as per our recommendation.

`fn request_reedem`

RequestRedeem event in spec is missing fields

- **fix:** https://github.com/interlay/interbtc-spec/pull/24
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L150-L152

```
150 *Events*
151
152 * ``RequestRedeem(redeemId, redeemer, amount, vault, btcAddress)``
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L73-L82

```
73      RequestRedeem(
74          H256,          // redeem_id
75          T::AccountId,  // redeemer
76          Wrapped<T>,    // redeem_amount_wrapped
77          Wrapped<T>,    // fee_wrapped
78          Collateral<T>, // premium
79          T::AccountId,  // vault_id
80          BtcAddress,    // user btc_address
81          Wrapped<T>,    // transfer_fee_btc
82      ),
```

Discrepancy in inequality used to check `dust_value`

- **fix:** https://github.com/interlay/interbtc-spec/pull/57

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L163

  The precondition on the `burnedTokens` amount is specified as strictly *greater than* (`MUST be above`):

```
``burnedTokens`` minus the inclusion fee MUST be above the :ref:`RedeemBtcDustValue`,
↪  where the inclusion fee is the multiplication of :ref:`RedeemTransactionSize` and the
↪  fee rate estimate reported by the oracle.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L337-L341

  But the implementation checks for **greater than or equal** to

```
    ensure!(
        // this is the amount the vault will send (minus fee)
        user_to_be_received_btc >= dust_value,
        Error::<T>::AmountBelowDustAmount
    );
```

fn liquidation_redeem

Discrepancy between specified and emitted event

- **fix**: https://github.com/interlay/interbtc-spec/pull/24
- **spec**: https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L208-L210

  The spec gives the event for this functions as

```
*Events*

- ``RequestRedeem(redeemID, redeemer, redeemAmountWrapped, feeWrapped, premium, vaultID,
↪  userBtcAddress, transferFeeBtc)``
```

- **impl**: https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L411-L412

  But according to the implementation (and function name) this should `LiquidationRedeem`:

```
     // vault-registry emits `RedeemTokensLiquidation` with collateral amount
     Self::deposit_event(<Event<T>>::LiquidationRedeem(redeemer, amount_wrapped));
```

fn execute_redeem

Discrepancy in specified signature

- **fix:** https://github.com/interlay/interbtc-spec/pull/57
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L234-L243

  The specified signature calls for a `vault` argument, which should be the vault responsible for executing the redeem request:

```
*Function Signature*

``executeRedeem(vault, redeemId, merkleProof, rawTx)``

*Parameters*
```

```
* ``vault``: the vault responsible for executing this redeem request.
* ``redeemId``: the unique hash created during the ``requestRedeem`` function.
* ``merkleProof``: Merkle tree path (concatenated LE SHA256 hashes).
* ``rawTx``: Raw Bitcoin transaction including the transaction inputs and outputs.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L226-L249

  However, on the public function exposed via RPC, there is only the general requirement that the request be signed by *someone* (any party is permissible):

```rust
    /// A Vault calls this function after receiving an RequestRedeem event with their public key.
    /// Before calling the function, the Vault transfers the specific amount of BTC to the BTC address
    /// given in the original redeem request. The Vault completes the redeem with this function.
    ///
    /// # Arguments
    ///
    /// * `origin` - anyone executing this redeem request
    /// * `redeem_id` - identifier of redeem request as output from request_redeem
    /// * `tx_id` - transaction hash
    /// * `tx_block_height` - block number of collateral chain
    /// * `merkle_proof` - raw bytes
    /// * `raw_tx` - raw bytes
    #[pallet::weight(<T as Config>::WeightInfo::execute_redeem())]
    #[transactional]
    pub fn execute_redeem(
        origin: OriginFor<T>,
        redeem_id: H256,
        merkle_proof: Vec<u8>,
        raw_tx: Vec<u8>,
    ) -> DispatchResultWithPostInfo {
        let _ = ensure_signed(origin)?;
        Self::_execute_redeem(redeem_id, merkle_proof, raw_tx)?;
        Ok(().into())
    }
```

In the private function, the `vault` is derived from the redeem request corresponding to the `redeem_id`:

```rust
    ext::vault_registry::redeem_tokens::<T>(&redeem.vault, burn_amount, redeem.premium, &redeem.redeemer)?;
```

https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L444

Discrepency in `burn` amount

- **fix:** https://github.com/interlay/interbtc-spec/pull/57
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L262

  According to the spec, the amount to be burned is derived via subtracting these two quantities:

```
``redeemRequest.amountBtc - redeemRequest.transferFeeBtc`` of the tokens in the redeemer's account MUST be burned.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L433-L438

  However in the implementation, we see the amount to be burned calculated by adding the quantities:

```
// burn amount (without parachain fee, but including transfer fee)
let burn_amount = redeem
    .amount_btc
    .checked_add(&redeem.transfer_fee_btc)
    .ok_or(Error::<T>::ArithmeticOverflow)?;
ext::treasury::burn::<T>(&redeem.redeemer, burn_amount)?;
```

fn cancel_redeem

Discrepancy in specified signature

- **fix:** https://github.com/interlay/interbtc-spec/pull/57
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L286-L288

  The specified signature gives two arguments:

```
*Function Signature*

``cancelRedeem(redeemId, reimburse)``
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L255-L266

  But the implementation requires three arguments, indicating the spec is missing the `origin` or `redeemer`:

```
    /// # Arguments
    ///
    /// * `origin` - sender of the transaction
    /// * `redeem_id` - identifier of redeem request as output from request_redeem
    /// * `reimburse` - specifying if the user wishes to be reimbursed in collateral
    /// and slash the Vault, or wishes to keep the tokens (and retry
    /// Redeem with another Vault)
    #[pallet::weight(if *reimburse { <T as Config>::WeightInfo::cancel_redeem_reimburse()
    ↪  } else { <T as Config>::WeightInfo::cancel_redeem_retry() })]
    #[transactional]
    pub fn cancel_redeem(origin: OriginFor<T>, redeem_id: H256, reimburse: bool) ->
    ↪  DispatchResultWithPostInfo {
        let redeemer = ensure_signed(origin)?;
        Self::_cancel_redeem(redeemer, redeem_id, reimburse)?;
```

Spec omits postcondition for when the user has requested retry and vault is liquidated

- **fix:** https://github.com/interlay/interbtc-spec/pull/41

It appears that the post-conditions for the case when the vault is liquidated and the user has requested a retry is not specified.

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L311

  The following postcondition is specified, on the condition that the vault is liquidated:

```
If the vault is liquidated, the redeemer MUST be transferred part of the vault's
↪  collateral: an amount of  ``vault.backingCollateral * ((amountIncludingParachainFee) /
↪  vault.to_be_redeemed_tokens)``.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L493-L497

  However, in the implementation, there is an additional qualification to the transfer in the case of a liquidated vault, triggered when the user has requested a retry, (i.e., when `reimburse == false`):

```
493     let slashing_destination = if reimburse {
494             CurrencySource::FreeBalance(redeemer.clone())
495         } else {
496             CurrencySource::LiquidationVault
497         };
498     ext::vault_registry::decrease_liquidated_collateral::<T>(&vault_id,
        ↪  confiscated_collateral)?;
499     ext::vault_registry::transfer_funds::<T>(
500         CurrencySource::ReservedBalance(vault_id.clone()),
501         slashing_destination,
502         confiscated_collateral,
503     )?;
```

  When `reimburse == false`, the `slashing_destination == CurrencySource::LiquidationVault`, and so the funds transfer goes to the Liquidation Vault instead of to the user.

  There is a section in the postconditions which addresses the condition that a user has requested replace, but it does do not account for the coincidence of the vault also being liquidated:

```
325 * If ``reimburse`` is false:
326     * All the user's tokens that were locked in :ref:`requestRedeem` MUST be unlocked,
    ↪  i.e. an amount of ``redeem.amount_btc + redeem.fee + redeem.transfer_fee_btc``.
327     * The vault's ``toBeRedeemedTokens`` MUST decrease by ``amountIncludingParachainFee``.
```

  https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L325-L327

- For context on the meaning of the `reimburse` flag, see: https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L293

```
293 * ``reimburse``: boolean flag, specifying if the user wishes to be reimbursed in DOT and
    ↪  slash the vault, or wishes to keep the interbtc (and retry to redeem with another
    ↪  Vault).
```

fn mint_tokens_for_reimbursed_redeem

unspecified precondition: `vault_id` must match vault id in redeem request

- **fix:** https://github.com/interlay/interbtc-spec/pull/59/commits/823143f7955edb3b1418cba87cc489d9a16e11bd
- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L606

  The implementation puts the following precondition on successful execution

```
606     ensure!(redeem.vault == vault_id, Error::<T>::UnauthorizedUser);
```

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L357-L361

  But this is precondition is not specified:

```
*Preconditions*

    * The BTC Parachain status in the :ref:`security` component MUST be set to
    ↪  ``RUNNING:0``.
    * A ``RedeemRequest`` MUST exist with an id equal to ``redeemId``.
    * ``redeem.status`` MUST be ``Reimbursed(false)``.
    * The vault MUST have sufficient collateral to remain above the
    ↪  :ref:`SecureCollateralThreshold` after issuing ``redeem.amount_btc +
    ↪  redeem.transfer_fee_btc`` tokens.
    * The vault MUST NOT be banned.
```

Precondition misplaced as postcondition

- **fix:** https://github.com/interlay/interbtc-spec/pull/41
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L364-L366

```
*Postconditions*

    * The function call MUST be signed by ``redeem.vault``, i.e. this function can only
↪  be called by the the vault.
```

Missing postcondition: redeem status is set to `Reimbursed(true)`

- **fix:** https://github.com/interlay/interbtc-spec/pull/57
- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L617

  The implementation entails that the redeem status will be changed to reimbursed if the function executes successfully:

```
        Self::set_redeem_status(redeem_id, RedeemRequestStatus::Reimbursed(true));
```

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L364-L368

  However, this is not reflected in the specified postconditions:

```
*Postconditions*

* The function call MUST be signed by ``redeem.vault``, i.e. this function can only be
↪  called by the the vault.
* :ref:`tryIncreaseToBeIssuedTokens` and :ref:`issueTokens` MUST be called, both with the
↪  vault and ``redeem.amount_btc + redeem.transfer_fee_btc`` as arguments.
* ``redeem.amount_btc + redeem.transfer_fee_btc`` tokens MUST be minted to the vault.
```

Events

Spec for `ExecuteRedeem` is missing `fee` and `transfer_fee` fields

- **fix:** https://github.com/interlay/interbtc-spec/pull/27
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/redeem.rst#L416-L423

  Four fields are specified:

```
ExecuteRedeem
-------------

Emit an event when a redeem request is successfully executed by a vault.

*Event Signature*

``ExecuteRedeem(redeemer, redeemId, amountinterbtc, vault)``
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L85-L86

  But the implementation has six fields:

```
        // [redeem_id, redeemer, amount_wrapped, fee_wrapped, vault, transfer_fee_btc]
        ExecuteRedeem(H256, T::AccountId, Wrapped<T>, Wrapped<T>, T::AccountId,
↪  Wrapped<T>),
```

Problem Scenarios

Such discrepancies can contribute to miscommunication, divergent understandings, inability to reason effectively about the system, and even clashing logic in different parts of the code base. Insofar as the spec is meant to guide alternative implementations, such discrepancy can contribute to incorrect implementations.

Recommendation

Correct the spec or the implementation, as appropriate, to bring them into alignment.

IF-INTERLAY2-REFUND: Discrepancies between implementation and spec in `refund` crate

Severity	Low
Type	Documentation
Difficulty	Easy
Status	Resolved by interbtc-spec#54

Involved artifacts

- https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/refund.rst
- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/refund/src/lib.rs

Description

The following discrepancies between the specification and the implementation of the `refund` crate were found:

`fn execute_refund`

Specified precondition that vault not be banned is unenforced

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/refund.rst#L72

The spec states the precondition that the vault not be banned:

```
72 * ``vault.isBanned()`` MUST return ``false``.
```

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/refund/src/lib.rs#L227-L233

However this is not enforced in the implementation:

```
227         // mint issued tokens corresponding to the fee. Note that this can fail
228         ext::vault_registry::try_increase_to_be_issued_tokens::<T>(&request.vault,
            ↪  request.fee)?;
229         ext::vault_registry::issue_tokens::<T>(&request.vault, request.fee)?;
230         ext::treasury::mint::<T>(&request.vault, request.fee)?;
231
232         // reward vault for this refund by increasing its SLA
233         ext::sla::event_update_vault_sla::<T>(&request.vault, ext::sla::Action::Refund)?;
```

Problem Scenarios

Such discrepancies can contribute to miscommunication, divergent understandings, inability to reason effectively about the system, and even clashing logic in different parts of the code base. Insofar as the spec is meant to guide alternative implementations, such discrepancy can contribute to incorrect implementations.

Recommendation

Correct the spec or the implementation, as appropriate, to bring them into alignment.

IF-INTERLAY2-REPLACE: Discrepancies between implementation and spec in `replace` crate

Severity	Low
Type	Documentation
Difficulty	Easy
Status	Resolved by various

Involved artifacts

- https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/replace.rst
- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/replace/src/lib.rs

Description

The following discrepancies between the specification and the implementation of the `replace` crate were found:

Outdated characterization of the fee model as a optional

- **fix:** https://github.com/interlay/interbtc-spec/pull/37
- **spec**: https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/replace.rst#L55-L61

  The spec presents the fee-related features as if they were conditional depending on the optional inclusion of the fee model. However, it is my understanding that the fee model is not considered an optional component any longer.

```
Fee Model
---------

The following additions are added if the fee model is integrated.

- If a replace request is cancelled, the griefing collateral is transferred to the *newVault*.
- If a replace request is executed, the griefing collateral is transferred to the *oldVault*.
```

`fn accept_replace`

unspecified precondition: ensure new vault is not banned

- **fix:** https://github.com/interlay/interbtc-spec/pull/55
- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/replace/src/lib.rs#L384-L385

The implementation requires that the `new_vault_id` not be banned:

```
        // Check that new vault is not currently banned
        ext::vault_registry::ensure_not_banned::<T>(&new_vault_id)?;
```

- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/replace.rst#L222

This is not reflected in the specified preconditions.

naming discrepancy between `consumedTokens` vs. `redeemable_tokens`

- **fix:** https://github.com/interlay/interbtc-spec/pull/55

Spec uses `consumedTokens` but implementation uses `redeemable_tokens`. This is a significant enough difference in terms and connotation that it could lead to confusion.

- **impl:** https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/replace/src/lib.rs#L420
- **spec:** https://github.com/interlay/interbtc-spec/blob/75fd3e2b9106c00cf33a111f133c6e2ab8c477d8/interbtc-spec/docs/source/spec/replace.rst#L247

Problem Scenarios

Such discrepancies can contribute to miscommunication, divergent understandings, inability to reason effectively about the system, and even clashing logic in different parts of the code base. Insofar as the spec is meant to guide alternative implementations, such discrepancy can contribute to incorrect implementations.

Recommendation

Correct the spec or the implementation, as appropriate, to bring them into alignment.

IF-INTERLAY2-SPEC: Specification of Concurrent Behaviors

Severity	Recommendation
Type	Protocol
Difficulty	Hard
Status	Unresolved

Involved artifacts

Description

In general, the possibility of determining whether a system is operating correctly is limited by the extent to which its expected behavior has been specified. As such, specification is a condition of possibility for determining whether or not something is correct. It is all the more important to specify behavior when dealing with concurrent systems, since the interaction of concurrent behaviors are notoriously difficult to reason about and often defy intuition. That said, the completeness and exactness of specification is a matter of degree: some aspects of a system are too innocuous to warrant specification, others are too little understood to enable it, while most aspects are worthy of general description, but not critical enough to call for rigorous specification. Each team must make their own cost/benefit analysis when deciding how extensively to describe their systems' expected behaviors and how intensively to specify those properties.

This finding collects recommendations regarding aspects of the system which are unspecified or underspecified.

Atomicity of operations:

This recommendation concerns the vault client implementation.

At the high level there are five operations performed by INTER-BTC bridge, that we refer to as high-level operations in the following:

- issue request,
- execute request,
- redeem request,
- refund request,
- replace request.

The specification of these operations should be sequentially defined, as an operation on one or more sequential objects, i.e., an account or a set. The distributed/concurrent implementation of these operations, as the specification defines it, is a sequence of steps performed by the parachain and the vault. These steps modify the bitcoin state, the parachain, but also the vault(s) state.

The specification does not emphasize the atomicity requirements over the implementation, more precisely over the reads and writes to the different stores (belonging to the implementation of the same high level operation). It does not mention which sequence of reads and writes lead to a successful execution of the operation and which lead to failure.

For example, redeem is defined by

1. on parachain:

- checks on the parachain for sufficient funds
- blocking of those funds, as if the redeem will succeed, and
- emit an event for the vault.

2. on vault

- the vault builds and sends the funds transfer transaction to the bitcoin chain, waits for it to be stable (i.e., enough confirmations) and
- calls `execute_redeem` on the parachain

3. the parachain finishes the redeem.

Which of these writes are allowed to fail without requiring a failure of concurrent ones on the other chain? Which writes, if successfully accomplished, must entail that all subsequent (concurrent) writes (maybe to a different chain) must succeed? For example, it looks like if a transaction is committed on the bitcoin chain the corresponding high-level operation must succeed.

More generally, the specification should highlight the atomicity constraints imposed on the implementation. Which steps define one atomic operation (the high-level one), and can be interleaved with steps of other (high-level) operations without changing the semantics or the outcome?

Problem Scenarios

For the redeem operation, the most difficult part of the atomicity proof is w.r.t. the vaults behaviour. Could pay (on the bitcoin) fail due to other operations? The answer seems to be no, in the absence of theft, because all operations are sequentialized by the parachain. So although the minus operation is not commutative in general for positive accounts, since all decrements to an account are sequentialised by the parachain and sent to the bitcoin only if it is possible to execute them, i.e., there are enough funds, all minus operations on the bitcoin chain become commutative. In the absence of theft, once the parachain checks passed (and an event to the vault is emitted) the high-level redeem operation seems to have taken place and it will complete successfully, even if the there are still a few writes do be done on bitcoin.

What happens in case of a theft, what is the result of the high-level redeem operation and when does it become effect-full in the case of a theft?

Recommendation

- Writing the spec of the vault client, highlighting the writes and reads that are part of the same atomic operation, is very useful for detecting concurrency bugs manually, and it is a required input of any formal methods tool.
- We recommend enriching the existing specification with diagrams like the one we have attached for replace, where the distributed steps implementing some high-level operation stand out. We recommend defining the expected result of the high-level operations starting from such diagrams, emphasizing the result of executing only a prefix of the steps defining the operation due to various failures, e.g., what happens if `execute_replace` fails or it is never called? Does the theft detection impact the result of the replace? If so, how?

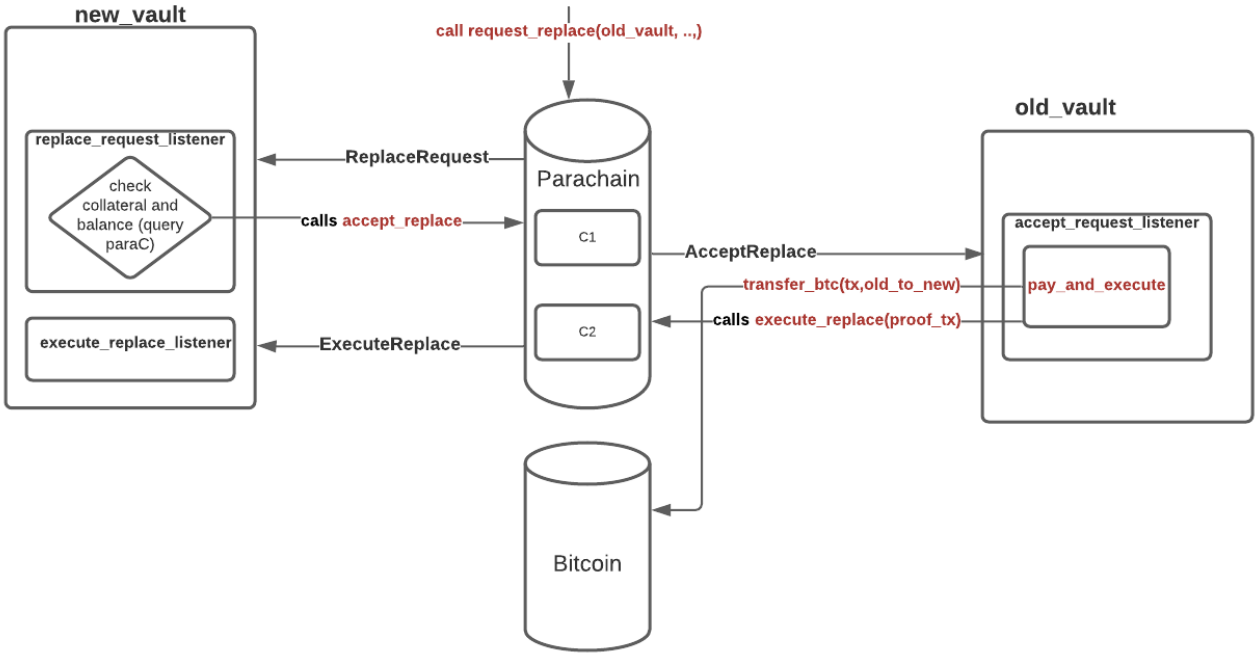

Figure 1: replace partial definition

IF-INTERLAY2-STORAGE: Redundant lookups in Substrate storage

Severity	Recommendation
Type	Implementation
Difficulty	Easy
Status	Resolved by interbtc@890080 and interbtc@38af23

Involved artifacts

- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/vault-registry/src/lib.rs

Description

We found one routine in which the same data is fetched from the Substrate storage at least 6 different times on any invocation.

In the `_request_redeem` function, the `Vault` storage is accessed through the following subroutines:

1. In the check to ensure that vault is not banned

- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/vault-registry/src/lib.rs#L1248
- via https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L333

2. In the check for available funds

- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/vault-registry/src/lib.rs#L1248
- via https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L344

3. When decreasing "to be released" tokens

- https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/vault-registry/src/lib.rs#L802
- via https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/redeem/src/lib.rs#L363

4. In each of these last checks, there is redundant double checking, first accessing to check if the vault exists, and then using a `get` to fetch the value:

   https://github.com/interlay/interbtc/blob/39565ae0a5402a00dc1f222ea322a13303cab9fb/crates/vault-registry/src/lib.rs#L562-L566

```
562 pub fn get_vault_from_id(vault_id: &T::AccountId) -> Result<DefaultVault<T>,
    ↪  DispatchError> {
563     ensure!(Self::vault_exists(&vault_id), Error::<T>::VaultNotFound);
564     let vault = Vaults::<T>::get(vault_id);
565     Ok(vault)
566 }
```

Problem Scenarios

This could impact performance.

Recommendation

Consider replacing the redundant checks in `get_vault_from_id` with use of Substrate's try_get, i.e.,

```
pub fn get_vault_from_id(vault_id: &T::AccountId) -> Result<DefaultVault<T>, DispatchError> {
    Vaults::<T>::try_get(vault_id)? // + map_err to DispatError
}
```

Additionally, we suggest the team consider the benefits of a design alteration that would allow fetching the `Vault` once, and then performing all the needed checks on the data once it's loaded into memory. This will likely improve performance, maintainability, and ease of reasoning about the code.

IF-INTERLAY2-VAULT-SPEC: Vault client is undocumented and unspecified

Severity	Low
Type	Documentation
Difficulty	Hard
Status	In Progress (see interbtc-clients#175)

Involved artifacts

- https://github.com/interlay/interbtc-clients/tree/2caf0a1079a69a6299dbc416dd48b3467345c8a4/vault

Description

There is robust documentation and clear specification of the parachain, but the vault client is undocumented and unspecified. While there are elements of the parachain specification that speak to the vault client's properties and operation, these are insufficient to understand the behavior of the latter.

A clear specification of the correct behavior of the vault client is important, because the system in which the vault client is involved is spread over three entities (the parachain, bitcoin, and the vault client). The result of the client interfacing operations depends on the result of partial executions on each of these entities. Without a specification that emphasises the results of client operations in case the execution on one of the participants fails, as suggested in Recommendation on Atomicity Violation Analysis of Vault Client, the evaluation of the correctness of the concurrency aspects in the implementation has no firm grounding.

Problem Scenarios

The most obvious problem we should expect is misunderstanding and confusion, whether on the part of engineers developing or maintaining the system or external readers reviewing it.

One example is the `issue_set` which is maintained by the parachain, but which the vault needs to know enough about that it can execute or cancel issues. Just reading the code, one could think that the vault maintains a faithful copy of the issue set which is stored on the parachain, however this is not the approach taken in the implementation: for request execution, it uses what seems to be a subset of the issue set on the parachain. Since this vault-local set is not used for cancellation, it is not expected to cause problems: the cancellation the vault "reads" the issue set from the parachain. However, the intended relationship between these sets is unclear, and had to be abducted by careful study of the code and discussion with the engineering team.

The implications of this lack of specification may be important also for the user (which can be a contract or AMM), that might want to understand in a timely manner when to cancel and resend a request.

Recommendation

Draft documentation and a specification of the vault client. This should include material like the exemplary material we've reviewed for the parachain, but we strongly recommend going further to include the kinds of material detailed in the Recommendation on Atomicity Violation Analysis of Vault Client