Security Audit Report

InterBTC Parachain: Protocol Design and Source Code

2021/06/12
Last revised 2021/09/17

Authors: Josef Widder, Shon Feder

Contents

Audit overview

The Project

In April 2021, Web3 and Interlay engaged Informal Systems to conduct a security audit over the documentation and the current state of the implementation of *interBTC*: a trustless bridge from Bitcoin to Polkadot formerly known as *PolkaBTC*. The bridge protocol is based on *XCLAIM*. XCLAIM is designed to support issuing, transferring, and redeeming Cryptocurrency Backed Assets (CbAs). XCLAIM is intentionally generic in order to support a wide range of assets but requires that one side of the bridge allows to execute smart contracts (Polkadot) while the other side just needs to provide a history of transaction in the backing currency (bitcoin).

Scope of this report

The agreed-upon workplan consisted of the following tasks:

- Task 1. Deep dive of the XCLAIM protocol and its sub-protocols
  - on tag 3.1.0
- Task 2. Crates to audit (parachain only)
  - `bitcoin`: on commit e4cb057.
  - `btc-relay`: on commit e4cb057.
  - `vault-registry`: on tag 0.7.4

This report covers Task 1 and Task 2 that were conducted May 10 through June 7, 2021 by Informal Systems under the lead of Josef Widder, with the support of Shon Feder.

Conducted work

Starting May 10, the Informal Systems team conducted an audit of the existing documentation and the code. Interlay gave us a one-hour presentation with an overview over the protocol with focus on the scope of this audit. For the protocol deep dive, the team also reviewed the xclaim paper. Our team started with reviewing the paper to get an overview of the protocol design principles, and the "Security Analysis" parts of the protocol specs v3.1.0 in order to get an overview over the specifics of `XCLAIM(BTC,DOT)`. We then continued with the specific subprotocols (`redeem`, `replace`, `issue`, `refund`, etc.) within the protocol specs v3.1.0 with special focus on correctness of concurrent execution of these protocols.

For the code review, Interlay gave as two one-hour code walk-throughs to help us getting started. We then started the code audit with the `bitcoin` and the `btc-relay` crates, and held back with the `vault-registry` crate for a week as Interlay updated the code when we started the audit. We audited `vault-registry` in the last week of the audit period.

Over the shared Discord channel we shared documents with preliminary findings, which we discussed during online meetings. In this document, we distilled the central findings into numbered findings, and the less central issues into a section called "minor comments".

Timeline

- 05/10/2021: Start
- 05/10/2021: Interlay presentation (1 hour)
- 05/11/2021: code walkthrough (1 hour)
- 05/12/2021: code walkthrough (1 hour)
- 05/19/2021: submitted first intermediate report on the protocol deep dive
- 05/21/2021: meeting Informal/Interlay with discussion of first report
- 05/26/2021: submitted first intermediate report on code audit ('bitcoin', 'btc-relay')

- 05/26/2021: meeting Informal/Interlay with discussion of code report
- 05/28/2021: submitted second intermediate report on the protocol deep dive
- 05/28/2021: meeting Informal/Interlay with discussion of second protocol report + code report
- 06/02/2021: submitted third intermediate report on the protocol deep dive
- 06/07/2021: submitted second intermediate report on code audit ('vault-registry')
- 06/07/2021: meeting Informal/Interlay with discussion of intermediate reports
- 06/07/2021: End of audit
- 06/16/2021: submission of first draft of this report

Conclusions

We found that the `XCLAIM(BTC,DOT)` design and security model in general is well thought out and addresses the challenges in bridge design, given the limitation that smart contracts can only be run on one side of the bridge. Despite the general high quality in the protocol design, we found some details that need to be addressed. We highlighted potential security issues in IF-INTERLAY-THEFT that are the result of the code of several protocols differing from the specification, and in IF-INTERLAY-NO-BLOCK where on-chain safety is based on an off-chain liveness assumption (existence of a correct and timely relayer). We highlighted two issues that are related to making more explicit incentives and rational behavior, namely, IF-INTERLAY-LIQUIDATION and IF-INTERLAY-TIMEOUT . This should help users of the bridge to understand the inherent risk they are taking and what are the beneficial actions in dynamic scenarios (exchange rate fluctuations, being close to timeout expiration). Finally, we gave some recommendations in IF-INTERLAY-SPEC to clarify the high-level temporal properties and invariants maintained by the protocol.

Overall we found the code well organized, well documented, and faithful to the specification. Despite the general high quality of the implementation work, we found seven issues regarding code quality, data representation, code organization, and divergence from the specification. These are detailed in the relevant findings. We also found a number of minor imperfections, which we note in the minor comments.

With one exception, all the findings we identified during the audit have been resolved at the time this report was last updates. The sole exception is IF-INTERLAY-SPEC, which sets out recommendations towards a more exhaustive and formalized specification.

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

- **Name**: Selected Crates in the InterBTC Parachain
- **Code Version**:
  - `bitcoin`: on commit e4cb057.
  - `btc-relay`: on commit e4cb057.
  - `vault-registry`: on tag 0.7.4
- **Specification Version**: tag 3.1.0
- **Type**: Specification and Implementation
- **Platform**: Rust, using the Substrate framework

Engagement Summary

- **Dates**: 5/10/2021 to 6/15/2021
- **Method**: Manual review
- **Employees Engaged**: 2
- **Time Spent**: 3 person-weeks

Engagement Goals

Scope

- Deep dive of the XCLAIM protocol and its sub-protocols
  - on tag 3.1.0
- Crates to audit (parachain only)
  - `bitcoin`: on commit e4cb057.
  - `btc-relay`: on commit e4cb057.
  - `vault-registry`: on tag 0.7.4

Aims of audit

(From the scoping doc)

1. **Process/specification** :: are there any flaws in the specification of the different protocols?
2. **Implementation/specification mismatches** :: are there discrepancies between the specification of the InterBTC protocols and their implementation?
3. **Bitcoin implementation issues** :: are there any issues in terms of Bitcoin compatibility (e.g. parsing, fork handling etc.)?
4. **Implementation issues** :: are there issues in the implementation that may introduce failures?
5. **Testing issues** :: are there cases/states of the parachain or clients not covered as part of the tests?

Coverage

Informal Systems manually reviewed, the xclaim paper, the protocol specs v3.1.0 the code of the crate `bitcoin` on commit e4cb057, of the crate `btc-relay` on commit e4cb057, and the crate `vault-registry` on tag 0.7.4.

Manual review resulted in the folowing findings:

- Reviewing the paper lead to finding unclear incentives IF-INTERLAY-LIQUIDATION, and unclear high-level properties and invariants as noted in IF-INTERLAY-SPEC.
- Reviewing the code and the specification we identified potential attacks in IF-INTERLAY-THEFT, IF-INTERLAY-NO-BLOCK as well as potential races in IF-INTERLAY-TIMEOUT.
- Reviewing the specification we found that the interaction between issue and refund are somewhat unclear, as reported in IF-INTERLAY-INTERACTION.
- Comparing specifications against the implementation, and reviewing the source code in detail yielded the various findings in IF-INTERLAY-ADTS, IF-INTERLAY-SUBJECTIVE, IF-INTERLAY-PARSING, IF-INTERLAY-NAMING, IF-INTERLAY-STORAGE, and IF-INTERLAY-WITNESS. Details of each are to be found in the relevant sections.
- From these activities, we also collected an extensive list of extensive minor comments. These remarks do not address major security or code quality risks, but aim to indicate minor defects or suggest helpful improvements.

Recommendations

This section summarizes key 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.

Short term

- *Align implementation with specification to eliminate attack vector (IF-INTERLAY-THEFT).* The implementation introduced an attack that was not possible in the specification
- Adding some shielding against timing attacks on-chain (IF-INTERLAY-NO-BLOCK). The assumption of a correct relayer is not checked in the `btc-relay`.
- Place links to the specific requirements from the spec in the docstrings of implementation units. This will help act against the other natural tendency of implementation and specification to diverge. See details in the minor comments.

Long term

- Clarify incentives and rational behavior (IF-INTERLAY-LIQUIDATION and IF-INTERLAY-TIMEOUT). Due to the dynamics of the system, it is not always clear what a specific agent should do in a specific environment to act rationally.
- Clarify invariants and temporal properties (IF-INTERLAY-SPEC). E.g., formalizing what is the relation of amounts BTC and amounts InterBTC over time is delicate.
- Look for opportunities to encode the business logic into the logic expressible in Rust's algebraic data types. Maximizing this brings a number of wins. See IF-INTERLAY-ADTS.
- Use the "parse, don't validate" approach where applicable to push validation of incoming data to the edge of the program. See IF-INTERLAY-PARSING.
- Make a principle of collocating storage updates as much as possible. This reduces the chance of errors introduced during development, and will likely lead to more performant code. See IF-INTERLAY-STORAGE

Minor comments

NOTE: At the time the report was last updated, the Interlay team has reported having made changes to address all of the following comments.

Fee / SLA

The specification of SLA could be made more complete along the following points:

- SLA and fee distribution is not so clear in the current documentation. It would be great to make these issues more explicit in specification to understand incentives.
- this seems to be the only place where SLA is updated in `redeem`, that is, there is no increase on success, like in issue.
  It would be great to have a central list want actions lead to increase/decrease of SLA.
- How precisely are SLAs and fees correlated. Is this subject to parameterization? Are there any constraints?

The flow graphic gives a great overview but is a bit unspecific what events lead to the depicted transfers.

Vault nomination

- "Operators are assumed to be trusted by their nominators not to steal Bitcoin backed by nominated collateral." The incentives of nomination are unclear, and could be clarified.
- "Operator and Nominator collateral cannot be withdrawn directly. Rather, withdrawals are subject to an unbonding period." It seems that there is a relation between `unbonding period` and the required time to `replace`: If a vault is close to a threshold it needs to replace. Otherwise, by unbonding it might run the risk to get under a threshold and even be liquidated.

Vault-registry

- functions not specified in specification
  - `accept_new_issues`
  - `report_undercollateralized_vault`
- What is the reason for the punishment delay? In particular, as the fees are spread over all participants, the consequence of not being able to act seem unclear.

Documentation improvements

The documentation provided by the specification and the doc strings on public objects is generally robust. The following minor suggestions were made towards further improvement.

Document the reference implementation and specs in the README of the `bitcoin` crate

Links to the reference implementations and documents used to guide the implementation for the `bitcoin` crate are only to be found scattered in various places in the source code. In keeping with the helpful documentation of specs and sources of truth we encounter in the README's of other crates in this project, ensure that the reference implementation any specs or documents used to guide development for the `bitcoin` crate are recorded prominently in the README.

Authoritative sources referenced include:

- the recapitulation in the btc-relay spec
- the btc-relay specs
- bitcoin core impl
- BIT

Fix Broken links

- The link to https://bitcoin.org/en/transactions-guide#term-null-data at https://github.com/interlay/interbtc-spec/blob/3.1.0/btcrelay-spec/docs/source/intro/accepted-format.rst#L14 gives a 404.
- The link to https://bitcoin.org/en/operating-modes-guide#simplified-payment-verification-spv at https://github.com/interlay/interbtc-spec/blob/3.1.0/btcrelay-spec/docs/source/intro/at-a-glance.rst#L8 gives a 404.

Code quality improvements

Code quality was generally deemed to be robust, especially given the volume and velocity of development. The following minor suggestions were offered to remove warts and further improve the quality.

Avoid use of magic numbers

While most constant values are properly protected and documented in named constants, there were a number of magic numbers in the source code.

Instances include

- https://github.com/interlay/interbtc/blob/e4cb057c2cb5c69c53d87deecce7627922332c1d/crates/bitcoin/src/script.rs#L50-L64
- https://github.com/interlay/interbtc/blob/e4cb057c2cb5c69c53d87deecce7627922332c1d/crates/bitcoin/src/parser.rs#L322
- https://github.com/interlay/interbtc/blob/e4cb057c2cb5c69c53d87deecce7627922332c1d/crates/btc-relay/src/lib.rs#L1202

The recommendation here aims to avoid the well-known problems with unnamed numerical constant.

Avoid redundant and scattered computations and validations

In addition to the more significant cases such as those identified in IF-INTERLAY-PARSING and IF-INTERLAY-STORAGE, we identified a more minor pattern of performing redundant computations when the information could have been passed between functions easily, sometimes requiring unnecessary reads via Substrate, and introducing the risk of logic that should be identical diverging.

One example is the duplicate computation of `current_block_height`. in the `_store_block_header` function.

Immediately after ensuring that the parachain is not shut down, `_store_block_header` applies `verify_block_header` to the `raw_block_header` to derive the `basic_block_header`:

https://github.com/interlay/interbtc/blob/e4cb057c2cb5c69c53d87deecce7627922332c1d/crates/btc-relay/src/lib.rs#L574

```
574 let basic_block_header = Self::verify_block_header(&raw_block_header)?;
```

The next five statements derive the `block_header_hash`, the `prev_header`, `prev_blockchain`, `prev_block_height`, and `current_block_height` on the basis of the `basic_block_header`: https://github.com/interlay/interbtc/blob/e4cb057c2cb5c69c53d87deecce7627922332c1d/crates/btc-relay/src/lib.rs#L577-L589

```
577         let prev_header =
          ↪  Self::get_block_header_from_hash(basic_block_header.hash_prev_block)?;
578
579         // get the block chain of the previous header
```

```
        let prev_blockchain = Self::get_block_chain_from_id(prev_header.chain_ref)?;

        // Update the current block header
        // check if the prev block is the highest block in the chain
        // load the previous block header block height
        let prev_block_height = prev_header.block_height;

        // update the current block header with height and chain ref
        // Set the height of the block header
        let current_block_height = prev_block_height + 1;
```

However, almost all of this is already computed in `verify_block_header` itself:

```
        let block_header_hash = raw_block_header.hash();

        // Check that the block header is not yet stored in BTC-Relay
        ensure!(
        !Self::block_header_exists(block_header_hash),
        Error::<T>::DuplicateBlock
        );

        // Check that the referenced previous block header exists in BTC-Relay
        let prev_block_header =
        ↪  Self::get_block_header_from_hash(basic_block_header.hash_prev_block)?;
        // Check that the PoW hash satisfies the target set in the block header
        ensure!(
        block_header_hash.as_u256() < basic_block_header.target,
        Error::<T>::LowDiff
        );

        // Check that the diff. target is indeed correctly set in the block header, i.e., check
        ↪  for re-target.
        let block_height = prev_block_header.block_height + 1;
```

In addition to following the recommendation of IF-INTERLAY-PARSING to parsing of the `raw_block_header` to the edge of the program, we recommend making the verification function take the arguments `BlockHeader, block_hash, prev_header, current_height`, to a `Result<(), DispatchError>`. This will reduce duplicate logic, minimizing the chance of errors being introduced letter in development.

Discrepancies with specification

There were numerous minor discrepancies between the specification (or, where relevant, reference implementation), and the implementation. Very few critical discrepancies were identified, and those are reported in the findings. But many minor discrepancies were identified, which is to be expected, as the implementation and specification evolve over time.

Nonetheless, we have found it is critical to keep the implementation and spec synchronized to the best of our ability. Subtle differences, such as slightly different naming of parameters or variables, can induce substantial disparities between the theory intended by the author of the specification and the understanding of the developer of the implementation (even when this communication is between the same person at different times). As a result of the fact that programming can be understood as theory building, the implications of such miscommunication on the correctness and reliability of the implementation can end up being quite serious.

As a general measure to help protect against drift, we recommend annotating the implementation with links back to the specification as precisely and frequently as possible. At the maximum, this means including a link to the specification of each function and data structure included in the respective objects docstring.

This is probably not an exhaustive list of discrepancies.

bitcoin crate

There were numerous and subtle differences between the names for things in the specs, reference implementation, and associated documentation, and the names for things in the implementation of the `bitcoin` crate. This adds cognitive load for any reader trying to connect the two, and adds to the risk of divergence.

In all these cases, we recommended either bringing the implementation in line with the spec or clearly documenting the divergence (often simply through comments in the source code).

Undocumented points of divergence included:

- Use of derived the value `target` instead of the specified `nBits`, in the BlockHeader fields.
- Undocumented absence of fields `tx_in count`, `tx_out count`. These are obviated in Rust by access to the length of the vectors used to store in/out transactions, but this should be commented, since they are listed in the spec.
- The fields named `tx_in` and `tx_out` in the reference implementation and documentation are instead named `inputs` and `outputs` in the `Transaction` struct.
- Discrepancies between the specified BlockChain structure and the

  implementation included:

  - fields specified as type `U256`, are given type `u32` in the (tho `U256` is imported from `sp_core` in the same module).
  - The `chain` field is missing in the implementation
  - `no_data` and `invalid` are specified as beings lists (`Vec`) but implemented as sets (`BTreeSet`).
- The specified `previous_output` field isn't present in the implementation of the `TransactionInput` struct.
- Value of opcode `OpInvalidOpcode` does not match the reference implementation.

  In the reference implementation, `OP_INVALIDOPCODE = 0xff`, but in the implementation, it is left implicit:

```
    OpNop9 = 0xb8,
    OpNop10 = 0xb9,

    OpInvalidOpcode,
```

  which will default to `0xba`.

  Since the opcode is not used, this itself won't create any issues with the current code, but it seems the discrepancy should be corrected.

btc-relay crate

- The `RichBlockHeader` includes the unspecified fields `para_height` and `account_id`. These could just be an implementation detail, were it not that the implementation annotates `account_id` as "required for fault attribution":

```
pub struct RichBlockHeader<AccountId, BlockNumber> {
    pub block_hash: H256Le,
    pub block_header: BlockHeader,
    pub block_height: u32,
    pub chain_ref: u32,
    // required for fault attribution
    pub account_id: AccountId,
    pub para_height: BlockNumber,
}
```

During discussion, the team determined that `account_id` could be removed. Which was completed in this PR.

- Minor discrepancies in the `initialize` function between the specification and implementation. include:
  - The function signature is missing the required `relayer` parameter.
  - The spec names the parameter `blockHeaderBytes` but the implementation has `raw_block_header`.
- An unspecified error can arise from calling `initialize` due to block header parsing. Only the `AlreadyInitialized` error is specified, but the implementation can also result in an error from a failed call to `parse_block_header`.
- Unspecified preconditions of `initialize`:
  - `raw_block_header` must be well formed and able to be parsed successfully
  - `active_block_height` must be initialized, due to btc-relay/src/lib.rs#L519-L520

```
519 // register the current height to track stable parachain confirmations
520 let para_height = ext::security::active_block_number::<T>();
```

  - The "Warning" re: block headers submitted to `initialize` needing to be from the main chain in the spec should be a function preconditions.
- Discrepancies between `store_block_header` function between the spec and the implementation include:
  - The spec has `blockHeaderBytes` but the implementation has `raw_block_header`.
  - Missing preconditions:
    1. `raw_block_header` is valid (if we are following the example of preconditions from `initialize`)
    2. Previous block (as indicated by the hash in the header of the current block) must already have been stored
       In order to compute the height of the current block, (which is required for both fork detection and storing the new block), the previous block must already be stored, otherwise we will error out in the call to `get_block_header_from_hash`:
       https://github.com/interlay/interbtc/blob/e4cb057c2cb5c69c53d87deecce7627922332c1d/crates/btc-relay/src/lib.rs#L977-L985

```
977 /// Get a block header from its hash
978 fn get_block_header_from_hash(
979     block_hash: H256Le,
980 ) -> Result<RichBlockHeader<T::AccountId, T::BlockNumber>, DispatchError> {
981     if BlockHeaders::<T>::contains_key(block_hash) {
982         return Ok(BlockHeaders::<T>::get(block_hash));
983     }
984     Err(Error::<T>::BlockNotFound.into())
985 }
```

  This requirement is implicit in the specification of the function, but to clarify for future implementations, the spec should make this explicit.

`vault-registry` crate

As with the other crates considered, the `vault-registry` implementation diverged from the specification in numerous small ways.

Disparity in structs fields

The spec of the `Vault` struct's `bannedUntil` field does not mention the optionality used in the implementation. To maintain alignment, the spec should either reflect this optionality, or the implementation should be annotated with a comment indicating the reason for the discrepancy.

Underspecification of event emissions

The events emitted by most functions are underspecified. As an example, consider the specification of the events emitted by `registerVault`.

The spec only indicates that `register_vault` emits a single event:

```
*Events*

* ``RegisterVault(Vault, collateral)``: emit an event stating that a new vault (``vault``) was registered and provide information on the Vault's collateral (``collateral``).
```

But the call to `lock` collateral during vault registration

```
ext::collateral::lock::<T>(vault_id, amount)?;
```

ends up emitting at least two additional events:

(2) `Event::Lock` in the lock function itself

```
    /// Lock an `amount` of currency. Note: this removes it from the
    /// free balance and adds it to the locked supply.
    ///
    /// # Arguments
    ///
    /// * `account` - the account to operate on
    /// * `amount` - the amount to lock
    pub fn lock(account: &T::AccountId, amount: BalanceOf<T, I>) -> DispatchResult {
        T::Currency::reserve(account, amount).map_err(|_| Error::<T, I>::InsufficientFreeBalance)?;

        // update total locked balance
        Self::increase_total_locked(amount)?;

        Self::deposit_event(Event::Lock(account.clone(), amount));
        Ok(())
    }
```

(1) `Event::Reserved` in the call to the `T::Currency::reserve` in the lock function, which, executes the following `Substrate` function:

```
        /// Move `value` from the free balance from `who` to their reserved balance.
        ///
        /// Is a no-op if value to be reserved is zero.
        fn reserve(who: &T::AccountId, value: Self::Balance) -> DispatchResult {
                if value.is_zero() { return Ok(()) }

                Self::try_mutate_account(who, |account, _| -> DispatchResult {
                        account.free = account.free.checked_sub(&value).ok_or(Error::<T, I>::InsufficientBalance)?;
                        account.reserved = account.reserved.checked_add(&value).ok_or(ArithmeticError::Overflow)?;
                        Self::ensure_can_withdraw(&who, value.clone(), WithdrawReasons::RESERVE, account.free)
                })?;

                Self::deposit_event(Event::Reserved(who.clone(), value));
                Ok(())
        }
```

As we discussed in one of our meetings, at minimum, the spec should explicitly inform the reader that the specified events emitted due to a successful function call are only a subset of the possible events. Otherwise, someone trying to reason about the system from the spec is liable to have a false sense of confidence.

Discrepancies in preconditions

For example, the spec is missing at least the following preconditions for `liquidateVault`:

- Vault is registered : https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/vault-registry/src/lib.rs#L1131
- Vault is active : https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/vault-registry/src/lib.rs#L1131

This preconditions are so widely required, it is probably a good idea to just state them as preconditions of the entire applicable class of functions.

Discrepancies in postconditions

E.g., in the spec for `liquidateVault` the following post-condition is not specified:

Vault status is set to `VaultStatus::Liquidated`, ensured by vault-registry/src/types.rs#L525

```
521         // Update vault: clear to_be_issued & issued_tokens, but don't touch to_be_redeemed
522         let _ = self.update(|v| {
523             v.to_be_issued_tokens = Zero::zero();
524             v.issued_tokens = Zero::zero();
525             v.status = status;
526             Ok(())
527         });
```

Additionally, the return value of this function is not specified in the signature, if this is not an implementation detail, may want to specify that it returns the collateral amount.

Findings

ID	Title	Type	Severity	Status
IF-INTERLAY-THEFT	Theft by redeeming (replacing) too much	Protocol	High	Resolved
IF-INTERLAY-NO-BLOCK	Scenario of "no block being recently submitted" (all relayers offline) not handled gracefully	Protocol	High	Resolved
IF-INTERLAY-LIQUIDATION	Liquidation event incentives unclear	Protocol	Medium	Resolved
IF-INTERLAY-TIMEOUT	Timeouts (and races) on sender chain	Protocol	High	Resolved
IF-INTERLAY-ADTS	Under-utilization of algebraic data types leads to confusing and error prone code	Practice	Low	Resolved
IF-INTERLAY-SUBJECTIVE	"Subjective initialization" condition assuming block_height is the correct height for the raw_block_header in relay initialization not specified	Protocol	Low	Resolved
IF-INTERLAY-PARSING	raw_block_header parsing occurs at multiple locations, but should be moved to the edge of the program	Implementation	Low	Resolved
IF-INTERLAY-NAMING	Documentation and variable naming of check_and_do_reorg function is misleading	Implementation	Low	Resolved
IF-INTERLAY-STORAGE	Storage updates of Vault struct and cached values are not co-located	Implementation	Low	Resolved
IF-INTERLAY-WITNESS	Missing check for illegal encoded witness in transaction parsing	Implementation	Low	Resolved
IF-INTERLAY-INTERACTION	Interaction between the `issue` and `refund` protocols	Protocol	Rec	Resolved
IF-INTERLAY-SPEC	Specification of Concurrent Behaviors	Specification	Rec	Unresolved

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-INTERLAY-ADTS: Under-utilization of algebraic data types leads to confusing and error prone code

Severity	Low
Type	Practice
Difficulty	Easy
Status	Resolved by interbtc@d3059, interbtc@98900

Involved artifacts

- bitcoin/src/script.rs#L40-L64
- bitcoin/src/address.rs#L34-L51
- bitcoin/src/types.rs#L315-L323
- bitcoin/src/parser.rst#L322-L326

Description

We identified several cases where inapt types were used to encode data, resulting in unclear and error prone code.

An illustrative example regards the `block_height` and `locktime` fields of the `Transaction` struct annotated with `FIXME` comments in bitcoin/src/types.rs#L315-L323

```
315 /// Bitcoin transaction
316 #[derive(PartialEq, Debug, Clone)]
317 pub struct Transaction {
318     pub version: i32,
319     pub inputs: Vec<TransactionInput>,
320     pub outputs: Vec<TransactionOutput>,
321     pub block_height: Option<u32>, //FIXME: why is this optional?
322     pub locktime: Option<u32>,     //FIXME: why is this optional?
323 }
```

The logic corresponding to this typing entails that either, both, or neither of the two fields could have values. However, this is an overly permissive encoding of what should actually a mutually exclusive choice between two alternatives, as shown by the parsing logic:

```
322     let (locktime, block_height) = if locktime_or_blockheight < 500_000_000 {
323         (None, Some(locktime_or_blockheight))
324     } else {
325         (Some(locktime_or_blockheight), None)
326     };
```

Since these are mutually exclusive values, and there is no possibility for both to `None` the correct representation for this would be a disjoint sum (canonically an `Either` type, but a custom enum would work as well).

The impact of the ill fitting encoding also shows up in the following unclear, method of accessing the mutually wrapped values:

```
199 // only block_height or locktime should ever be Some
200 if let Some(b) = self.block_height.or(self.locktime) {
201     formatter.format(b)
202 }
```

This is hard to read, thus the clarifying comment. More importantly, it is not faithful to the underlying logic: reasoning locally about this code, one would naturally assume that `b` may or may not end up being formatted. But, as we know from the way values of this type are constructed, this is an infallible conditional, and `b` will always end up being formatted.

These problems are resolved by instead representing the alternative with the proper type:

```
let b = match self.block_or_time {
    Foo::BlockHeight(b) => b,
    Foo:LockTime(b) => b
};
formatter.format(b);
```

This encoding obviates the need for comments, and gives a more faithful encoding of the logic at the type level.

Another salient example regards the predicate methods on the `Script` struct, used to determine the kind of script represented by the wrapped bytes. We can see from the usage of this predicate that is also representing mutually exclusive alternatives:

```
    pub fn from_script_pub_key(script: &Script) -> Result<Self, Error> {
        if script.is_p2pkh() {
            // 0x76 (OP_DUP) - 0xa9 (OP_HASH160) - 0x14 (20 bytes len) - <20 bytes pubkey hash>
            ↪  - 0x88 (OP_EQUALVERIFY)
            // - 0xac (OP_CHECKSIG)
            Ok(Self::P2PKH(H160::from_slice(&script.as_bytes()[3..23])))
        } else if script.is_p2sh() {
            // 0xa9 (OP_HASH160) - 0x14 (20 bytes hash) - <20 bytes script hash> - 0x87
            ↪  (OP_EQUAL)
            Ok(Self::P2SH(H160::from_slice(&script.as_bytes()[2..22])))
        } else if script.is_p2wpkh_v0() {
            // 0x00 0x14 (20 bytes len) - <20 bytes hash>
            Ok(Self::P2WPKHv0(H160::from_slice(&script.as_bytes()[2..])))
        } else if script.is_p2wsh_v0() {
            // 0x00 0x20 (32 bytes len) - <32 bytes hash>
            Ok(Self::P2WSHv0(H256::from_slice(&script.as_bytes()[2..])))
        } else {
            Err(Error::InvalidBtcAddress)
        }
    }
```

Problem Scenarios

Failing to make effective use of algebraic types to encode the logic of the underlying domain increases the cost of maintenance by requiring that logic to be implemented manually, and it leaves unnecessary opportunities for future development to introduce errors due to, e.g.,

- `Script` values of the wrong kind being used in certain contexts, if the developer doesn't remember to invoke the predicate first.
- Case analysis of a `Script`'s kind might be incomplete.
- Logic errors introduced by developers reasoning locally about the code.

Recommendation

Look for opportunities to encode the business logic into the logic expressible in Rust's algebraic data types. By expressing logical relationships like mutually exclusive predicates and values, necessarily conjoined values, or implications, in the type level, we leverage the type checker to automate the enforcement of logical invariants through exhaustiveness checking and data structures that are correct by construction. (I.e., always keep an eye out for opportunities to leverage Curry-Howard .)

IF-INTERLAY-INTERACTION: Interaction between the `issue` and `refund` protocols

Severity	Recommendation
Type	Protocol
Difficulty	Easy
Status	Resolved by interbtc-spec#68

Involved artifacts

- Specification

Description

As indicated in IF-INTERLAY-THEFT, there is some flexibility in the design that allows a difference between

- the amount of BTC submitted via `request-issue`, and
- the actual amount transferred from a user to a vault on the bitcoin network.

During a meeting in the course of the audit, it was discussed that this was done to allow more usability.

One aspect of this area is that there is a `refund` protocol, that allows to request the return of over-payments. As a result, there is substantial complexity in the different choices and the combination of the `issue` and `refund` protocols.

Problem Scenarios

The usability of the bridge might be hindered as the choice of actions in these protocols may not be clear to the user.

Recommendation

We suggest to clarify the interaction between the `issue` and `refund` protocols. More generally, the (distributed) control flow of the `issue` protocol could be documented more explicitly.

It would be great to have a protocol flow (decision tree) from the viewpoint of the user. For `issue` it could start with the following points:

1. issue request-issue (`x`) on Polkadot
2. transfer `y` to vault on Bitcoin (expected `x = y`)
    - if transaction never occurs on Bitcoin, then the vault can call cancel issue after timeout
      -> "TO COMPLETE: outcome"
3. otherwise, that is, if transaction appears on Bitcoin
    - user may execute-issue (proof of `y` transaction) on Polkadot with the following possible outcomes depending on the value of `y`
        - `x = y`:
            - execute-issue was in-time -> `success`.
              -> user has `x` InterBTC
            - execute-issue was too late
                - cancel-issue had been called before
                  -> "TO COMPLETE: outcome"

- cancel-issue had not been called before
  -> "TO COMPLETE: outcome"
- `x < y`: "TO COMPLETE: resulting flow"
- `x > y`: "TO COMPLETE: resulting flow"
  - vault has enough collateral
  - vault does not have enough collateral
  - user may try to refund ... "TO COMPLETE: possible outcomes etc."

IF-INTERLAY-LIQUIDATION: Liquidation event incentives unclear

Severity	Medium
Type	Protocol
Difficulty	Hard
Status	Resolved by interbtc-spec@64120a via interbtc-spec#40

Involved artifacts

- XCLAIM SP paper
- Specification
- Liquidation Documentation

Description

Liquidation is the solution to the problem of "Redeemability". Intuitively, it means that a user that exchanges BTC for InterBTC should get its value back. A formal understanding of this property is not immediate. However, in the solution it is enforced

- either by successful completion of `redeem`, or
- by slashing a vault that does not comply with `redeem`, or
- by a liquidation event. The latter might lead to somewhat surprising results.

In particular in the case of a liquidation event, the incentives of the different agents are not so clear. Similarly, the solution is based on thresholds where it is not so clear what is the rationale for the concrete values for these thresholds.

Problem Scenarios

Liquidation event. Assume BTC suddenly rises relative to DOT:

- In order to not fall under the liquidation thresholds, a vault needs to quickly add collateral
- If it fails to do so quickly enough, it might be liquidated
- The users will be reimbursed DOTs at the current rate
- If BTC continues to rise,
  - the vault still has its locked BTC and will have made a profit from not acting quickly enough
  - the users will have lost their expensive BTC for relatively cheap DOTs
- The paper mentions in Section III.C an assumption on Delta_min(epsilon), but neither the assumption itself nor how it is used is clear
- The paper also claims that this "is necessary to prevent users from financial loss". What the precise meaning of this claim is, is not so obvious in case of exchange rate fluctuations.

This may provide a potentials attack vector whereby dishonest vaults could deliberately profit off of exchange rate fluctuations (deliberately not participate in redeem protocol, not update collateral, etc.).

Recommendation

Thresholds

- There are thresholds for security, redemption, and liquidation (e.g., 200%, 120%, and 110%) in the systems. It was discuss in the meetings during that audit that the actual values are derived from comparable systems. However, we suggest to analyze how these thresholds hold up against historical data about exchange rate fluctuations between BTC and DOT. E.g., a central question is “how often a typical vault might have run into liquidation events in the last year due to rapid fluctuations?”. Further, the rational used to determine these numbers should be documented.

Section V.D of the paper presents the solution based on multiple thresholds. However, due to (a) exchange rate fluctuations and (b) issue requests, the precise problem that is solved by the solution is not clear. What are the precise assumption on the following?

- exchange rate fluctuations
- validator responsiveness (timeliness and its financial capacity to add collateral)
- user interference to mitigate her risk by redeeming

A vault might need to rapidly act to remain within the thresholds. Thus, exchange rate fluctuations impose

- a requirement for a vault to rapidly add collateral to avoid being liquidated (assuming it does not want to be liquidated)
- a requirement on the user to redeem in order to mitigate exchange rate loss due to liquidation.

These timing constraints need to be captured.

Realistic scenarios

The above issues only appear in extreme situations. Can the expectation of “non-extreme” situations be captured, e.g., “if the exchange rate changes by x in t time units, then properties are guaranteed”?

Incentives

We suggest to document different scenarios (e.g., rising/falling BTC value relative to DOT), and for each agent (e.g., vault, user, staked relayer) to make explicit the rational behavior. This would also help in argueing whether the protocols are indeed aligned with the incentives.

Reconsider Liquidation as Liveness concern

Liquidation on Theft is expensive (nearly as expensive as possible; 150% of BTC value). At the same time:

- Undercollateralization can only be achieved by
  - not locking collateral quickly enough
  - not redeeming -> being punished -> go beyond threshold -> liquidation
- It is unclear what precisely is achieved by liquidating
  - the vault must hold on to its BTC
  - users will receive DOT
    - value depends on when the users want to burn
    - race between users to burn or redeem depending in the exchange rate

Thus, while presented as safety concern, “Severe Undercollteralization” in fact can only be achieved by passive vaults. As a result, if a vault maintainer is offline for some time period, this may result in a slashing event. The consequences should be discussed/highlighted in the documentation.

IF-INTERLAY-NAMING: Documentation and variable naming of `check_and_do_reorg` function is misleading

Severity	Low
Type	Implementation
Difficulty	Medium
Status	Resolved by interbtc@a355e

Involved artifacts

- https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/btc-relay/src/lib.rs#L1371-L1378

Description

`check_and_do_reorg` is documented as follows:

https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/btc-relay/src/lib.rs#L1371-L1378

```
1371 /// Checks if a newly inserted fork results in an update to the sorted
1372 /// Chains mapping. This happens when the max height of the fork is greater
1373 /// than the max height of the previous element in the Chains mapping.
1374 ///
1375 /// # Arguments
1376 ///
1377 /// * `fork` - the blockchain element that may cause a reorg
1378 fn check_and_do_reorg(fork: &BlockChain) -> Result<(), DispatchError> {
```

but in the condition where the function is called, it is in the branch where `is_fork` is false:

https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/btc-relay/src/lib.rs#L623-L635

Problem Scenarios

This led to significant confusion while trying to read code (even for the developers). Such legibility issues increase the risk of error during development.

Recommendation

Dom suggested that changing `is_fork` to `is_new_fork` in the `initialize` function could help.

I also recommend moving the initial check inside `check_and_do_reorg` out of that function:

```
// Check if the ordering needs updating
// if the fork is the main chain, we don't need to update the ordering
if fork.chain_id == MAIN_CHAIN_ID {
    return Ok(());
}
```

Putting this conditional inside of the function with an early return that bypasses the function body effectively hides the control flow.

iiuc, what we really what to say is

```
if !(chain.chain_id == MAIN_CHAIN_ID) {
  do_reorg(chain)
}
```

but with the current structure, the reader can't see this control flow until they look inside `check_and_do_reorg`.

IF-INTERLAY-NO-BLOCK: Scenario of "no block being recently submitted" (all relayers offline) not handled gracefully

Severity	High
Type	Protocol
Difficulty	Hard
Status	Resolved by interbtc@6fb17

Involved artifacts

- security analysis
- Specification
- btc-relay/src/lib.rs

Description

As mentioned in the security analysis, the BTC-relay safety relies on a "a steady stream of Bitcoin block headers". In other words, the safety of the on-chain software relies on **safety and liveness** of *off-chain* activity.

> It should be noted that the security of the chainrelay does not directly derive from the PoW "objectivity property", as it involves transaction outside of the bitcoin network (that are not visible there). Hence it is unfeasible to rely on the usual incentive assumptions appealed to by SPVs.

BTC-Relay does not deal with time, that is, BTC-Relay does not have a failure mode in the case when no new headers are uploaded for extended durations.

Problem Scenarios

If no correct relayer is online, the BTC-relay may not be updated for an undetermined period of time. As a result the state of BTC-relay may be arbitrarily outdated. At this point, an adversarial relayer may submit an alternative bitcoin history and thus "prove" existence of non-existing bitcoin transactions.

Recommendation

Adding some shielding against timing attacks on-chain. To illustrate how other bridges deal with this kind of problem we sketch how IBC mitigates such an attack. It does so by

1. disabling the on-chain light client if no new header was uploaded for a specific time span (trusting period).
2. introducing a "packet delay". Translated to interBTC, this would mean that a transaction can only be verified against header h if there are sufficiently many headers on top of the h **AND** h has been uploaded some "quarantine period" ago (to give other relayers some time to fix the current view of the bitcoin chain at BTC-relay).

IF-INTERLAY-PARSING: `raw_block_header` parsing occurs at multiple locations, but should be moved to the edge of the program

Severity	Low
Type	Implementation
Difficulty	Easy
Status	Resolved by interbtc@347a4

Involved artifacts

- https://github.com/interlay/interbtc/blob/e4cb057/crates/btc-relay/src/lib.rs#L1177

Description

Currently, parsing the `RawBlockHeader` happens in multiple different places, e.g.,

- In `verify_block_header`: https://github.com/interlay/interbtc/blob/e4cb057/crates/btc-relay/src/lib.rs#L1177
- In `initialize`: https://github.com/interlay/interbtc/blob/e4cb057/crates/btc-relay/src/lib.rs#L516

Problem Scenarios

Anywhere raw data is handled inside a program there is a risk of corrupting, misparsing, or otherwise compromising its integrity. There is also a runtime cost of having to perform validation repeatedly.

Recommendation

Using the parse, don't validate approach described by Alexis King, these functions can be changed to take a `BlockHeader` instead of a `RawBlockHeader`, and the parsing of the raw header can be pushed into the edges of the program.

This moves one of the preconditions for the functions into static analysis (which can then be removed from the spec), simplifies the possible error handling needed for these complex functions, and ensures junk data is intercepted at the earliest possible point, thus reducing the chance for errors to be inserted later during maintenance and development.

Since the hash of the `raw_block_header` is also needed, I suggest adding a private field to the `BlockHeader` struct that stores the hash, and then a `hash` getter to retrieve this value. This follows, e.g., https://github.com/summa-tx/bitcoin-spv/blob/master/golang/btcspv/types.go#L20-L27

IF-INTERLAY-SPEC: Specification of Concurrent Behaviors

Severity	Recommendation
Type	Protocol
Difficulty	Hard
Status	Unresolved

Involved artifacts

- XCLAIM SP paper
- Specification
- vault registry specification

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.

Protocol Level - System goals (as discussed in the paper)

General remarks:

- The overall properties of the protocol as described in the paper are sometimes vague.
- Sometimes the properties are implicitly preconditioned by environment assumptions (e.g., timing, synchrony, smooth exchange rate fluctuations). These assumptions should be made explicit.

We give some more detailed comments below.

Auditability

While in general the goal is clear, the term "protocol failures" is not so clear. Under the assumption that the `chainrelay` is reliable, in principle it is enough to have access to I to track the complete history of the transactions.

Consistency

The property seems hard to formalize precisely. It should be something like: if $i(b)$ is issued at time t, then at time t, b tokens are locked and it holds that $|i(b)| = |b|$. Also it should be formalized that a token in the backing currency cannot be blocked for multiple issued tokens.

Redeemability

see IF-INTERLAY-THEFT.

Liveness

Liveness seems to be preconditioned by some timing assumptions. These should be clarified. Section V.B of the paper mentions several timing assumptions. It is not clear whether

- some components in the design are responsible for ensuring them, or
- they are entirely put on the environment.

It is also not clear whether these timing assumptions are relevant for safety or liveness (or both).

Some of these timing assumptions also are safety critical (see IF-INTERLAY-NO-BLOCK). This should be clarified.

Invariants

The vault registry manages different amounts of tokens in InterBTC and DOTS, e.g.,

- `toBeIssuedTokens`
- `issuedTokens`
- `toBeRedeemedTokens`
- `collateral`

To specify correct behavior, it would be great to define

- invariants, e.g.,
  - toBeRedeemedTokens <= issuedTokens
- transition invariants, e.g.,
  - for all functions different from `executeIssue` and [...] it holds that `issuedTokens' = issuedTokens`
  - for all functions different from [...], `toBeIssuedTokens' + issuedTokens' = toBeIssuedTokens + issuedTokens`

Towards this, there already exist quite insightful figures (e.g., this figure). They are very helpful in understanding the evolution of the protocol/amounts. However, the figures are not always complete/correct. For instance, `executeIssue` may move a different amount of tokens than the one announced in `requestIssue`, so the invariant is not obvious.

Global invariants between BTC and InterBTC

Intuitively, one would expect some global invariants somewhat in the spirit of

(I) `"BTC locked in vaults" = "issued InterBTC"`

or at least some stabilization property, e.g.,

(II) if there are no new requests (issue, redeem, etc) for 24 hours, then (I) holds.

Problem Scenarios

The InterBTC protocol is a collection of several protocols that each involve different transactions on the Bitcoin network and Polkadot. In production, many instantiations of these protocols will run in parallel. In order to convince oneself that the protocol is "correct" under concurrency, one would need explicit statements on the invariants, or the expected preconditions/postconditions under which protocols run. For the mentioned amounts, such formalizations are lacking in the specification.

Recommendation

A formalization of these invariants is a pre-requisite to reason more formally about the correctness of concurrent execution of the involved protocols.

IF-INTERLAY-STORAGE: Storage updates of Vault struct and cached values are not co-located

Severity	Low
Type	Implementation
Difficulty	Medium
Status	Resolved by interbtc@82424 via interbtc#165

Involved artifacts

- https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/vault-registry/src/lib.rs#L1173-L1177
- https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/vault-registry/src/ext.rs#L23
- https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/vault-registry/src/slash.rs#L149-L170

Description

When depositing funds into a vault, the storage is updated in the following places:

- https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/vault-registry/src/lib.rs#L1173-L1177
- https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/vault-registry/src/ext.rs#L23
- https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/currency/src/lib.rs#L167-L168

But then it seems that same calculations and data must also be represented in the `Vault` struct, via

https://github.com/interlay/interbtc/blob/7c02930d7f7f0b32693110b702bde74072436594/crates/vault-registry/src/slash.rs#L149-L170

```rust
pub trait TryDepositCollateral<
    Collateral: TryInto<u128> + CheckedAdd,
    SignedFixedPoint: FixedPointNumber,
    E: From<SlashingError>,
>: SlashingAccessors<Collateral, SignedFixedPoint, E>
{
    /// Called by the vault or nominator to deposit collateral.
    fn try_deposit_collateral(&mut self, amount: Collateral) -> Result<(), E> {
        checked_add_mut!(self, mut_collateral, &amount);
        checked_add_mut!(self, mut_total_collateral, &amount);
        checked_add_mut!(self, mut_backing_collateral, &amount);

        let amount_as_fixed = collateral_to_fixed::<Collateral, SignedFixedPoint>(amount)?;
        let slash_per_token = self.get_slash_per_token()?;
        let slash_per_token_mul_amount = slash_per_token
            .checked_mul(&amount_as_fixed)
            .ok_or(SlashingError::ArithmeticOverflow)?;
        checked_add_mut!(self, mut_slash_tally, &slash_per_token_mul_amount);

```

```
        Ok(())
    }
}
```

And then the Vault struct itself must updated in storage.

Problem Scenarios

Having the cached valued updated in totally different locations than the principle data structure presents lots of opportunity for subtle logic errors to be introduced during maintenance and development.

Recommendation

A more robust and maintainable design would be one of the following:

- collocate the storage updates and struct updates (each update could be a single call into a method on the `Vault` struct)
- fetch the struct values from the storage (of course, this comes with a performance penalty)
- derive all updates to the ancillary storage locations from the vault struct perhaps through a `store` method on the `Vault` struct (this was suggested by Greg in our meeting).

We recommend considering any other places where in memory structs are updated separately from the storage layer, and co-locating the logic of these updates as closely as possible.

IF-INTERLAY-SUBJECTIVE: "Subjective initialization" condition assuming `block_height` is the correct height for the `raw_block_header` in relay initialization not specified

Severity	Low
Type	Protocol
Difficulty	Easy
Status	Resolved by interbtc-spec@6a25a

Involved artifacts

- btc-relay/src/lib.rs

Description

When the `btc-relay` is initialized, a `block_header` and `block_height` are supplied:

https://github.com/interlay/interbtc/blob/e4cb057c2cb5c69c53d87deecce7627922332c1d/crates/btc-relay/src/lib.rs#L511

```
    pub fn initialize(relayer: T::AccountId, raw_block_header: RawBlockHeader, block_height:
    ↪  u32) -> DispatchResult {
```

The given `block_height` must correctly reflect the height of the `raw_block_header`, or else the fork height detection mechanism will be invalid, since detection relies on the `block_height`, which is calculated based on incrementing each successive blocks height based on that of it's ancestor.

We call this "subjective initialization" in tendermint.

Problem Scenarios

The incorrect block height could be given during initialization of the btc-relay, making all subsequent block height calculations incorrect.

Recommendation

The specification should make this assumption clear and it could detail any governance mechanisms, conventions, or external conditions that help ensure that the assumption holds.

IF-INTERLAY-THEFT: Theft by redeeming (replacing) too much

Severity	High
Type	Protocol
Difficulty	Easy
Status	Resolved by interbtc@67c5a

Involved artifacts

- Specification
- issue/src/lib.rs
- redeem/src/lib.rs
- replace/src/lib.rs

Description

The protocols `Issue`, `Redeem` and `Replace` contain the following steps on Polkadot:

- request to transfer a specific amount of BTC
- submit and verify a proof for the transfer (in order to effectuate the operation)

While the specification mostly indicates that the requested amounts and the transferred amounts should be equal, the implementation is more flexible. The implementation allows to redeem and replace larger amounts than initially specified.

As a result, it is unclear what correct behavior actually is intended. The specification states, e.g., for Issue, on `executeIssue`: "If the function completes successfully, the user receives the requested amount of InterBTC into his account." However, the implementation deviates from the specification. It allows different amounts to be issued than requested.

We also observed similar deviations for `redeem` and `replace`. For these, there are scenarios that are quite comparable to theft, e.g., when too many BTC are redeemed. In addition, it seems that over-redeeming is not recorded, and thus leads to deviation of the actually "locked BTC" and the InterBTC issued by the same vault.

Problem Scenarios

- In the case of `Redeem`, a `vault` and a `user` can collaborate: A vault may redeem more BTC than recorded in the Polkadot smart contract.
- In the case of `Replace`, two vaults can collaborate and replace more BTC than recorded in the Polkadot smart contract.

In more detail, the implementation allows to transfer too many BTC linked to `replace` and `redeem` requests. This breaks an invariant that roughly corresponds to

(I) `"BTC locked in vaults" = "issued InterBTC"`.

Further, reporting of theft also allows

- redeeming too much
- "replacing" too much

Intuitively, we understand these latter two points as behavior that may lead to similar situations as theft.

Moreover, it seems that the concept of "undercollateralization" is detected and checked in terms of InterBTC (rather than BTC locked with a vault). This is OK if (I) holds. However, if (I) is violated, then this becomes less clear.

Recommendation

- For `Redeem` and `Replace`: Implement checks for equality of the requested amount and the actually transferred amounts, in the execute as well as in the slashing conditions.
- For `Issue`, the checks do not seem safety relevant. Non-precise checks allow for more usability. We suggest to precisely document the cases; cf. IF-INTERLAY-SPEC.

Collaborative discussion

In the course of addressing this finding, the team at Interlay identified a related attack scenario where multiple transactions on the Bitcoin network could contain the same `OP_RETURN`. This constitutes a variant of double spending. The proposed solution was to introduce a new evidence type that consists of two distinct transactions with the same `OP_RETURN`, and thus punish this behavior. We recommend to implement this slashing condition.

IF-INTERLAY-TIMEOUT: Timeouts (and races) on sender chain

Severity	High
Type	Protocol
Difficulty	Hard
Status	Resolved by interbtc@c43d5 via interbtc#156

Involved artifacts

- XCLAIM SP paper
- Specification

Description

The protocol uses timeouts, mostly to de-risk agents. E.g., during `issue`, if a vault locks InterBTC, and the user fails to transfer (rather "fails to prove a transfer of") BTC within 24 hours, the locked InterBTC will be returned to the vault. Similarly, if `redeem` times out, the vault is slashed for not providing a proof of a BTC transfer.

Problem Scenarios

If the vault redeems BTC but fails to get the transaction on the issuing chain in-time (before Delta^I_redeem expires), then there is a **race** between the vault and the user (depending on whether `execute` is still allowed after the timeout expires). The user can end up with

- its initial BTC
- **AND** wrongly paid back InterBTC

Even if there is no race, if `cancel` succeeds although the BTC transaction took place, this seems to violate an (implicit) invariant that should relate the amount of locked BTC with the stored amounts of issued InterBTC in Polkadot.

Recommendation

Clarify use cases around timeouts

The inherent reason for the race lies in the problem of proving the absence of payment on the bitcoin chain (from vault back to the user). In the design, the timeout period needs to transpire on the Polkadot side.

In other designs, e.g., in IBC there is no such race: the comparable timeout would need to transpire **on the receiving side**, that is, translated to interBTC, the vault would need to get the transaction into bitcoin before a certain timeout height T is reached **on the bitcoin chain**. Then, the absence of the transaction could be proven by inspecting the bitcoin chain up to height T. On the other hand, the vault also would not try to get the transaction on the bitcoin chain once the height T is surpassed.

We suggest do document the involved risks, incentives, and potentially broken global invariants in adverse scenarios in the specification.

Similar to the race in `redeem`, there is a timeout with a race in issue. The function `executeIssue` may abort because of timeout (measured in terms of activeblockcount), even if there is a Merkle proof that the transfer has happened.

Time parameters in the paper

The paper mentions several involved times that should be clarified:

- Delta^I_redeem: enforced by the smart contract. What are the guarantees on time provided by the Polkadot blockchain?
- Delta_relay: seems to be an assumption of XLAIM. It is the sum of
  - Delta_B: from transaction broadcast to secure inclusion. This is out of control of the protocol. How is it estimated? Is it accounted for that a transaction may be put on a "wrong fork"?
  - Delta_submit: unclear what this precisely is. It seems that a relayer might be responsible to ensure it. Are relayers incentivized to do so, or is it also the responsibility of the user?
  - 2 Delta^I: unclear
- Delta_redeem > Delta^B + Delta_relay
- Where is the time it takes the user to prepare and submit a transaction to Bitcoin (after initiating the redeem process and the "Delta^I_redeem" timeout starts to run)?
- the mentioned timeout for batching is unclear

IF-INTERLAY-WITNESS: Missing check for illegal encoded witness in transaction parsing

Severity	Low
Type	Implementation
Difficulty	Easy
Status	Resolved by interbtc@44f00 via interbtc#160

Involved artifacts

- reference implementation
- bitcoin/src/parser.rs#L317
- bitcoin/src/types.rs#L272-L275

Description

In the reference implementation, a transaction with a set witness-flag must actually include witnesses in the transaction.

```
214         if (!tx.HasWitness()) {
215             /* It's illegal to encode witnesses when all witness stacks are empty. */
216             throw std::ios_base::failure("Superfluous witness record");
217         }
```

However, this was not checked in the parsing function:

```
314     if (flags & 1) != 0 && allow_witness {
315         flags ^= 1;
316         for input in &mut inputs {
317             input.with_witness(flags, parser.parse()?);
318         }
319     }
```

or the `with_witness` method it envokes:

```
272     pub fn with_witness(&mut self, flags: u8, witness: Vec<Vec<u8>>) {
273         self.flags = flags;
274         self.witness = witness;
275     }
```

Problem Scenarios

An illegally encoded transaction could be parsed successfully.

Recommendation

Add a check that makes the implementation faithful to the bitcoin core reference implementation.