HashCloak

Code Review and Security Assessment
For
Term Structure Labs

Delivery: September 25, 2023

Updated: October 12, 2023

Prepared For:
Term Structure Labs

Prepared by:
Onur Inanc Dogruyol | *HashCloak Inc.*
Soumen Jana | *HashCloak Inc.*

Table Of Contents

Executive Summary

Term Structure Labs engaged *HashCloak Inc.* for an audit of its *Term Structure* product, which is a decentralized bond protocol that enables peer-to-peer lending and borrowing with fixed interest rates. The audit was done from *August 28th, 2023* to *September 25th*, *2023*. The relevant codebases were:

- Smart contract repository: ts-contract-diamond, assessed at commit: 483d2cf65124bd3bc55142c61e488ba2e2d20cf8
- Circom repository: ts-circom-dd, assessed at commit: 956ef6b2383cc883e9e7797c2cff49f421561018

The scope of the audit was all files in the following folders:

- `decentral-portal/ts-circom-dd`
- `decentral-portal/ts-contract-diamond`

In response to Term Structure Labs' request, this Audit Report is organized into two distinct sections. In this section, we will comprehensively assess the Circom component, examining its codebase, functions, and security considerations. We will outline our findings, including potential vulnerabilities and recommendations for enhancements or mitigations.

Throughout the audit, we familiarized ourselves with the contracts in scope and sought to understand how the overall *Term Structure* protocol works. For the first 2 weeks of the audit, we familiarized ourselves with the smart contracts and circuits. During this time, we focused on finding simple bugs and issues within the codebases. In the subsequent two weeks of the audit, we focused on finding more complex bugs and issues. For the smart contract portions, we combined our manual analysis skills alongside automated, off-the-shelf tooling such as Slither. For the circuits, we mainly relied on manual analysis but used circumspect and circumscribe to help identify simpler issues within the circuits.

Overall, we found the issues range from high to informational:

Severity	Number of Findings	Severity	Number of Findings
Critical	0	Low	1
High	0	Informational	1
Medium	5		

Findings

TKS-C-1: Underconstraint division inside `IntDivide` circuit

Type: Medium
Files affected: src/gadgets/_mod.circom

Description: The divisor inside the `IntDivide` circuits is not constrained. It's important to constrain the divisor to ensure that it is non-zero. Since division cannot be expressed using a quadratic constraint, it is common to use the following pattern to ensure that the signal `quotient` is equal to the `dividend/divisor`

```
quotient <-- dividend/divisor
quotient * divisor + remainder === dividend
```

This code snippet forces the `quotient` to be equal to `dividend/divisor` during the witness generation and checks that `quotient * divisor + remainder === dividend` during proof verification. However, the statement `quotient <-- dividend/divisor` only makes sense when the `divisor` is non-zero.

Impact: `quotient * divisor + remainder === dividend` may be also true even when the `divisor` is zero. This leads to verifying a wrong proof when the divisor is zero. For this reason, it is vital to also constrain the `divisor` is non-zero during the proof verification.

Suggestion: Add the following constraints inside the `IntDivide` circuit:

```
Signal is_zero <== IsZero()(divisor);
is_zero === 0;
```

Status: We reviewed the commit hash 54542f27ff93eac9a45ee6e1305973291ecf9cbf and the team fixed the issue by adding a new signal **mask** and this also changes the `dividend_` signal to be equal to `dividend * mask`. Also, the assignment for `(quotient, remainder)` is changed. Hence, these changes provide the same functionality as we suggested.

TKS-C-2: Mismatching bit lengths inside the `OrderLeaf_Alloc`

Type: Medium
Files affected: src/type/_mod.circom

Description: The `OrderLeaf_Alloc` circuit in line 159 uses the `Num2Bits` circuit to check the bit lengths. The document says the `OrderLeaf` type has the `cumAmt0`, `cumAmt1`, and `lockedAmt` of length `UnsignedAmount` of bits. However, the `OrderLeaf_Alloc` circuit uses the `BitsAmount` function in lines 165, 166, and 167 instead of using the `BitsUnsignedAmt` function.
Impact: This might lead to arithmetic overflow when using `OrderLeaf_Alloc` circuit.

Suggestion:
Instead of using the following:

```
_ <== Num2Bits(BitsAmount()(order_leaf.cumAmt0);
_ <== Num2Bits(BitsAmount())(order_leaf.cumAmt1);
_ <== Num2Bits(BitsAmount())(order_leaf.lockedAmt);
```

Use:

```
_ <== Num2Bits(BitsUnsignedAmt())(order_leaf.cumAmt0);
_ <== Num2Bits(BitsUnsignedAmt())(order_leaf.cumAmt1);
_ <== Num2Bits(BitsUnsignedAmt())(order_leaf.lockedAmt);
```

Status: We reviewed the commit hash 54542f27ff93eac9a45ee6e1305973291ecf9cbf and the team fixed the issue by changing the lines as we suggested.

TKS-C-3: Missing bit length checks for the inputs of `TagGreaterThan` and `TagGreaterEqThan` in some circuits

Type: Medium
Files affected: request.circom,

Description:
In order for the constraints and the outputs to be accurate while using the `LessThan` circuit, input signals need to be constrained so that the maximum number of bits outside the `LessThan` circuit expects a specified number of bits.

Although `Num2Bits` circuits are used inside the `LessThan`, it is used `in[0] + (1 << n) - in[1]` instead of on the inputs of the `LessThan` circuit themselves. Therefore, the `Num2Bits` constraints should be added for the inputs of the `LessThan` function.

`Num2Bits` needs to be used for the inputs of the tag comparators such as `TagGreaterThan`, and `TagGreaterEqThan`.

In the codebase, the following have the same issue inside the request.circom:

1. Before using the following code in line 671 inside the `DoReqPlaceOrder` circuit, `Num2Bits` circuits need to be used for each input for `TagGreaterThan`
    - `ImplyEq()(is2nd`, `1`, `TagGreaterThan(BitsRatio()` + `BitsTime())([MQ * daysFromMatchedIfEnabled` + `365 * BQ`, `BQ` * `daysFromMatchedIfEnabled]))`;

2. Before using the following code in line 1477 inside the `DoReqCreateTSBToken` circuit, `Num2Bits` circuit needs to be used for the first input.
    - `ImplyEq()(enabled`, `1`, `TagGreaterThan(BitsTime())([p_req.matchedTime[0]` + `86400` * `upper_lim_of_days`, `tSBToken.maturity]))`;

Impact: `TagGreaterThan` and `TagGreaterEqThan` circuits output 1 if `in[0] < in[1]`, and 0 otherwise. If the `in[0]` is used as a small number and `in[1]` is used as a number more than n bits mistakenly, this would cause an underflow.

Suggestion:
Add the following constraints inside the `DoReqPlaceOrder` circuit before `TagGreaterThan` is used in line 671 which is specified in the description part.

```
_ <== Num2Bits(BitsUnsignedAmt() + BitsTime())(MQ *
daysFromMatchedIfEnabled + 365 * BQ);
_ <== Num2Bits(BitsUnsignedAmt() + BitsTime())(MQ *
daysFromMatchedIfEnabled + 365 * BBQ * daysFromMatchedIfEnabled);
```

Add the following constraint inside the `DoReqCreateTSBToken` circuit before `TagGreaterThan` `is` used in line 1477 which is specified in the description part.

```
_ <== Num2Bits(BitsTime() + 1)(p_req.matchedTime[0] + 86400 *
upper_lim_of_days);
```

Status: We reviewed the commit hash 54542f27ff93eac9a45ee6e1305973291ecf9cbf and the team fixed the issue. In line, 671, our suggestion was replacing `BitsUnsignedAmt() + BitsTime() with BitsRatio() + BitsTime() inside the Num2Bits()`. The *Term Structure* team changed it with `BitsUnsingedAmt() + BitsTime() + 1`, which is the correct version. They fixed the other parts of the finding as intended using the suggestion part.

TKS-C-4: Missing bit length check of currentTime may lead to arithmetic overflow/underflow

Type: Medium
Files affected: request.circom, normal.circom,

Description:
This finding is similar to that of TKS-4. `Num2Bits` needs to be used for the inputs of the tag comparators such as `TagLessThan`, `TagLessEqThan`, `TagGreaterThan`, and `TagGreaterEqThan`.
`currentTime` is used in various places inside some of the tag comparators. However, the `Num2Bits` circuit is not used before these comparators.

Impact: If it is not used, there might be arithmetic overflow/underflow due to mismatching bit lengths.

Suggestion: Add the following constraint inside the `DoRequest` circuit in normal.circom.

```
_ <== Num2Bits(BitsTime())(currentTime);
```

Status: We reviewed the commit hash 54542f27ff93eac9a45ee6e1305973291ecf9cbf. The changes are not included in this commit since those checks are guaranteed not to overflow/underflow of signals as we discussed with the team.

TKS-C-5: Missing bit length checks for the inputs of TagLessThan and TagLessEqThan in some circuits

Type: Medium
Files affected: mechanism.circom, src/gadgets/_mod.circom, req.circom, request.circom, normal.circom

Description:
In order for the constraints and the outputs to be accurate while using the `LessThan` circuit, input signals need to be constrained so that the maximum number of bits outside the `LessThan` circuit expects a specified number of bits.

Although `Num2Bits` circuits are used inside the `LessThan`, it is used `in[0] + (1 << n) - in[1]` instead of on the inputs of the `LessThan` circuit themselves. Therefore, the `Num2Bits` constraints should be added for the inputs of the `LessThan` function.

Impact: `TagLessThan` and `TagLessEqThan` circuits output 1 if `in[0] < in[1]`, and 0 otherwise. If the `in[0]` is used as a small number and `in[1]` is used as a number more than n bits mistakenly, this would cause an underflow.
Suggestion:
Add the following constraint to the circuit before line 100 inside the `AuctionCalcFee` inside mechanism.circom

```
_ <== Num2Bits(BitsEpoch())(matchedPIR);
```

Add the following constraints to the circuit before line 337 inside the `SecondMechanism` circuit inside the mechanism.circom

```
_ <== Num2Bits(BitsAmount())(remainTakerSellAmt);
_ <== Num2Bits(BitsAmount())(matchedMakerBuyAmtExpected);
```

Add the following constraints inside the `Min` and `Max` circuit just before using the `TagLessThan` circuit inside the src/gadgets/_mod.circom. Use `Num2Bits` circuit for the inputs of `TagLessThan`. Therefore, the circuits will become the following:

```
template Min(bits){
      signal input in[2];
      _ <== Num2Bits(bits)(in[0]);
      _ <== Num2Bits(bits)(in[1]);
      signal slt <== TagLessThan(bits)(in);
      signal output out <== Mux(2)([in[1], in[0]], slt);
}

template Max(bits){
      signal input in[2];
      _ <== Num2Bits(bits)(in[0]);
      _ <== Num2Bits(bits)(in[1]);
      signal slt <== TagLessThan(bits)(in);
      signal output out <== Mux(2)([in[0], in[1]], slt);
}
```

Add the following constraints inside the `Req_CheckExpiration` circuit just before using the `TagLessThan` circuit inside the req.circom. Therefore, the circuit will become the following:

```
template Req_CheckExpiration(){
      signal input req[LenOfReq()];
      signal input {bool} enabled;
      signal input currentTime;

      component req_ = Req();
      req_.arr <== req;

      _ <== Num2Bits(BitsTime())(currentTime);
      ImplyEq()(enabled, 1, TagLessThan(BitsTime())([currentTime *
enabled, req_.arg[2] * enabled]));
}
```

Add the following constraint inside the `DoReqInteract`, `DoReqEnd`, `DoReqSecondMarketEnd` circuits in the request.circom before using the `TagLessEqThan` circuit:

- Use the following code block just before line 1128 inside the `DoReqEnd` template
- Use the following code block just before line 980 inside the `DoReqInteract` template
- Use the following code block just before line 1224 inside the `DoReqSecondMarketEnd` template:

  ```
  _ <== Num2Bits(BitsAmount())(feeFromTarget);
  _ <== Num2Bits(BitsAmount())(matched_amt1);
  ```

Status: We reviewed the commit hash 54542f27ff93eac9a45ee6e1305973291ecf9cbf. The changes are not included in this commit since those checks are guaranteed not to overflow/underflow of signals as we discussed with the team.

TKS-C-6: Missing bit length check for remainder inside `IntDivide` circuit

Type: Low
Files affected: src/gadgets/_mod.circom
Description: The remainder inside the `IntDivide` circuits is not constrained. This circuit has a missing bit length check on the output remainder. Two important constraints are ensuring the quotient and the remainder have the proper number of bits. There is a bit length check on the quotient, however, there is no check for the remainder.

In order to ensure that the remainder doesn't contain too many bits and proceed to cause unexpected behavior, a bit length constraint must be added to the remainder.

Impact: Without a bit length constraint on the remainder, the output of the `IntDivide` circuit is not guaranteed to be in the expected number of bits. Therefore, anyone who uses this circuit is not guaranteed to have the remainder be accurate and as expected.

Suggestion: Add the following constraints inside the `IntDivide` circuit.

```
_ <== Num2Bits(bits_divisor)(remainder);
```

Status: We reviewed the commit hash 54542f27ff93eac9a45ee6e1305973291ecf9cbf and the team fixed the issue by adding the same line as our suggestion.

TKS-C-7: Unused intermediate signals that only occur in a single constraint

Type: Informational
Files affected: src/request.circom

Description:
Since intermediate signals are not available outside the template, using an intermediate signal in a signal constraint might be a sign that the implementation has an issue. To be explained more clearly, using an intermediate signal in a single constraint indicates that either that signal is constrained in the single place unnecessarily or it needs to be used in another place and it is forgotten to constraint. In this case, these are unused.

In line 1039, `packedAmt1` is an intermediate signal that occurs only in a single constraint.

Another intermediate signal that is constrained in a single place is `isSecondaryMarket`**.** In line 1095, `isSecondaryMarket` is constrained as a boolean value. However, this signal is not used in another place inside the End Request.

Impact: This reduces the readability of the code.

Suggestion: Delete `packedAmt1` and `isSecondaryMarket` as they are unused.

Status: We reviewed the commit hash 54542f27ff93eac9a45ee6e1305973291ecf9cbf and the team fixed the issue by deleting the signals as we suggested.