Report

v. 1.0

Customer

Term Structure

Circuit Audit

Term Structure

29th August 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	29.08.23	A. Zveryanskaya	Initial Draft
0.2	29.08.23	A. Zveryanskaya	Minor revision
1.0	29.08.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Term Structure is a decentralized bond protocol powered by zkTrue-up, a platform that enables peer-to-peer lending and borrowing with fixed interest rates.

3 Project scope

We were asked to review:

- Original Circuts Code
- Code with Fixes

Files:

/			
	spec.circom	normal.circom	evacuation.circom
src/			
	request.circom	mechanism.circom	
src/type/			
	unit.circom	unit_set.circom	token.circom
	state.circom	sig.circom	req.circom
	prep_req.circom	order.circom	nullifier.circom
	fee.circom	channel.circom	bool.circom
	account.circom	_mod.circom	
src/gadgets/			
	tag_comparators.circom	merkle_tree_posei-don.circom	indexer.circom
	fp.circom	_mod.circom	
src/const/			
	req_type.circom	fmt.circom	bits.circom
	_mod.circom		

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 1 critical, 9 major, and a few less important issues. All identified Critical and Major issues have been fixed.

Issues	Info	Fixed
Severity: Critical	0	1
Major	0	9
Moderate	1	22
Minor	1	37

Fixed 69 out of 71 issues

6 Critical Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** fp.circom

Description This approach fails when ”in” is a multiple of 10^32, as it will produce exponent that doesn’t fit into 5 bits.

Client Comment *Indeed, this is an intentional design choice rather than an inadvertent flaw. Here’s our design rationale:*
To minimize gas costs for calldata during rollup, we’ve introduced a constraint on the precision of user-specified numbers using our pre-defined float format. With this format, we ensure that the size of the transformed number doesn’t exceed 5 bytes. This pre-efined float format is verified before users sign the transactions. Our Backend system validates these values for conversion to floating-point, rejecting any invalid inputs. Circuit also check if the values are convertable to floating-point values. If the value is a multiple of 10^32, it is not convertable to a floating-point value. This is our constraint rule.
We added a teampalte ’Fix2FloatCond()’ to check if a circuit with specific business logic is correct. For exmple, the ’Fix2FloatCond()’ checks if a transfer transaction fit the constraint rule associated with the transfer logic. Despoit transacitons don’t need to respect the constraint rule.

```
44 while(val_in % 10 == 0){
       val_in /= 10;
       val_exp++;
   }
```

7 Major Issues

CVF-48. FIXED

- **Category** Suboptimal
- **Source** account.circom

Description This implicitly performs range check on “acc.nonce” value that is probably already guaranteed to be in range.

Recommendation Consider either avoiding this additional check, or, in case the check is indeed necessary, making the check more explicit.

Client Comment *We removed the redundant validation check. We do a cehck within ’PreprocessedReq_Alloc()’ in the beginning of Normal().*

```
10 signal output arr[LenOfAccLeaf()] <== AccLeaf_Alloc()([addr, acc.
      ↪ nonce, acc.tokens]);
```

CVF-55. FIXED

- **Category** Suboptimal
- **Source** _mod.circom

Description The resulting function is not collision resistant for different ‘len‘: it is possible to craft two inputs of different lengths that produce the same hash.

Recommendation Consider doing a proper domain separation here or make sure that different-length inputs are never used.

Client Comment *We have implemented PoseidonArbitraryLen and PoseidonSpecificLen to address this issue. The bigint array and its length in PoseidonSpecificLen are hashed together, with the length value placed at the beginning. Please refer to the documentation for more details.*

```
64 template PoseidonV2(len){
```

CVF-57. FIXED

- **Category** Suboptimal
- **Source** request.circom

Description This logic relies on the fact that MaxFeeUnitsPerReq is 1.

Recommendation Consider either not relying on this fact or asserting it explicitly.

Client Comment *Added assertions at the beginning of this template.*

```
22 component fee_unit = FeeUnit();
   fee_unit.arr <== unit_set.feeUnits[0];
```

CVF-58. FIXED

- **Category** Suboptimal
- **Source** request.circom

Description This logic relies of the fact that MaxBondUnitsPerReq is 1.

Recommendation Consider either not relying on this fact or asserting it explicitly.

Client Comment *Added assertions at the beginning of this template.*

```
27 component bond_unit = BondUnit();
   bond_unit.arr <== unit_set.bondUnits[0];
```

CVF-59. FIXED

- **Category** Suboptimal
- **Source** request.circom

Description This logic relies on the fact that MaxOrderUnitsPerReq is 1.

Recommendation Consider either not relying on this fact or asserting it explicitly.

Client Comment *Added assertions at the beginning of this template.*

```
32 component order_unit = OrderUnit();
   order_unit.arr <== unit_set.orderUnits[0];
```

CVF-60. FIXED

- **Category** Suboptimal
- **Source** request.circom

Description This logic relies on the fact that MaxNullifierUnitsPerReq is 1.

Recommendation Consider either not relying on this fact of asserting it explicitly.

Client Comment *Added assertions at the beginning of this template.*

```
51 component nullifier_unit = NullifierUnit();
   nullifier_unit.arr <== unit_set.nullifierUnits[0];
```

CVF-63. FIXED

- **Category** Readability
- **Source** mechanism.circom

Recommendation Consider giving descriptive names to these ”temp” signals” to improve code readability.

Client Comment *Resolved*.

```
15 signal temp <== interest * days + one * (365 - days);
   signal temp2;
```

```
28 signal temp <== feeRate * days;
   signal temp2;
```

```
44 signal temp <== feeRate * days;
   signal temp2 <== absInterest * temp;
   signal temp3;
```

```
77 signal temp <== (days * (priceMQ - priceBQ) + 365 * priceBQ);
```

```
84 signal temp0 <== avl_BQ * days;
   signal temp1 <== temp0 * priceMQ;
   signal temp2 <== 365 * priceBQ;
   signal temp3 <== (365 - days) * priceBQ;
   signal temp4 <== temp3 * avl_BQ;
```

```
93 signal temp <== MQ * feeRate;
```

```
110 signal temp7 <== (1 - (is_market_order * (1 - makerSide)));
```

```
114 signal temp0, temp1, temp2, temp3;
```

CVF-64. FIXED

- **Category** Documentation
- **Source** mechanism.circom

Description The semantics of these templates is unclear.

Recommendation Consider documenting.

Client Comment *Added comments.*

73
```
template CalcNewBQ(){
```

80
```
template CalcSupMQ(){
```

CVF-65. FIXED

- **Category** Documentation
- **Source** mechanism.circom

Description Here "TagIsEqual()([lend_req.arg[1], borrow_req.arg[1]])" ensures that both orders have the same maturity time. This seems reasonable, but the specification doesn't have such check.

Client Comment *All behaviors in the Backend are now documented in the zkTrueUp circuit documentation, with circuit verification items highlighted*

159
```
isMatched <== And()(TagIsEqual()([lend_req.arg[1], borrow_req.arg
   ↪ [1]]), And()(TagIsEqual()([lend_req.tokenId, borrow_req.arg
   ↪ [4]]), TagGreaterEqThan(BitsRatio())([borrow_req.arg[3],
   ↪ lend_req.arg[3]])));
```

8 Moderate Issues

CVF-51. FIXED

- **Category** Suboptimal
- **Source** bool.circom

Recommendation While for two arguments these implementations are fine, in some case it could be necessary to compute logical AND or OR of many arguments. In such case, applying a chain of two-argument AND/OR templates could be suboptimal. More efficient approach would be: Calculate AND(x_1, x_2, ..., x_n) as x_1 + x_2 + ... + x_n == n Calculate OR(x_1, x_2, ..., x_n) as x_1 + x_2 + ... + x_n != 0 Consider implementing MultiAnd and MultiOr templates.

Client Comment *The recommendation has already been implemented in "./circuits/zkTrueUp/src/type/bool.circom".*

```
3 template And(){
```

```
11 template Or(){
```

CVF-52. FIXED

- **Category** Suboptimal
- **Source** nullifier.circom

Recommendation This could be optimized as: (digest - nullifierLeaf[0]) * ((digest - nullifierLeaf[1]) * ... != 0

Client Comment *Resolved.*

```
12 template NullifierLeaf_CheckCollision(){
```

CVF-53. FIXED

- **Category** Suboptimal
- **Source** order.circom

Recommendation Consider using boolean ”or” here instead of just sum. The fact that sum here is equivalent to ”or” is based on a business logic above that could change in the future. Such implicit relationships within code makes the code more error-prone.

Client Comment *Resolved*.

```
62 isFull <== isFullAsAuction + isFullAsSecondary;
```

CVF-54. FIXED

- **Category** Procedural
- **Source** merkle_tree_poseidon.circom

Recommendation Consider extracting these common formulas into a binary selector template.

Client Comment *Resolved*.

```
19 hash[0].inputs[0] <== (merkle_proof[0] - leaf_node) * n2B.out[0] +
      ↪ leaf_node;
20 hash[0].inputs[1] <== (leaf_node - merkle_proof[0]) * n2B.out[0] +
      ↪ merkle_proof[0];
```

```
23     hash[i].inputs[0] <== (merkle_proof[i] - hash[i - 1].out) * n2B.
          ↪ out[i] + hash[i - 1].out;
       hash[i].inputs[1] <== (hash[i - 1].out - merkle_proof[i]) * n2B.
          ↪ out[i] + merkle_proof[i];
```

CVF-56. FIXED

- **Category** Suboptimal
- **Source** fp.circom

Recommendation This array of signals could be demoted to an array of vars.

Client Comment *Resolved*.

```
9 signal temp[5];
```

CVF-62. FIXED

- **Category** Suboptimal
- **Source** mechanism.circom

Recommendation This signal is redundant. Just use ”debtAmt” instead.

Client Comment *Resolved*.

```
16 signal temp2;
```

```
29 signal temp2;
```

```
46 signal temp3;
```

CVF-79. FIXED

- **Category** Overflow/Underflow
- **Source** fee.circom

Description Overflow is possible here.

Recommendation Consider additionally checking that ”amount” fits into ”BitsUnsignedAmt” bits.

Client Comment *Resolved*.

```
7 _ <== Num2Bits(BitsUnsignedAmt())((feeLeaf[0] + amount) * enabled);
```

CVF-80. FIXED

- **Category** Overflow/Underflow
- **Source** token.circom

Description Overflow is possible here.

Recommendation Consider performing additional range check on ”amount”.

Client Comment *Resolved*.

```
7  signal output arr[LenOfTokenLeaf()] <== TokenLeaf_Alloc()([(
      ↪ tokenLeaf[0] + amount) * enabled, tokenLeaf[1] * enabled]);//!
      ↪ !tricky
```

```
25 signal output arr[LenOfTokenLeaf()] <== TokenLeaf_Alloc()([(
      ↪ tokenLeaf[0] + amount) * enabled, (tokenLeaf[1] - amount) *
      ↪ enabled]);//!!tricky
```

CVF-81. FIXED

- **Category** Overflow/Underflow
- **Source** token.circom

Description Underflow is possible here.

Recommendation Consider performing additional range check on ”amount”.

Client Comment *Resolved*.

```
13 signal output arr[LenOfTokenLeaf()] <== TokenLeaf_Alloc()([(
      ↪ tokenLeaf[0] - amount) * enabled, tokenLeaf[1] * enabled]);//!
      ↪ !tricky
```

```
19 signal output arr[LenOfTokenLeaf()] <== TokenLeaf_Alloc()([(
      ↪ tokenLeaf[0] - amount) * enabled, (tokenLeaf[1] + amount) *
      ↪ enabled]);//!!tricky
```

```
31 signal output arr[LenOfTokenLeaf()] <== TokenLeaf_Alloc()([tokenLeaf
      ↪ [0] * enabled, (tokenLeaf[1] - amount) * enabled]);//!!tricky
```

CVF-82. FIXED

- **Category** Suboptimal
- **Source** _mod.circom

Description This doesn't allow negative fees, but does allow fees exceeding unsigned amount bits.

Recommendation Consider adding "1 « BitsUnsignedAmt()".

Client Comment *Resolved.*

```
196 _ <== Num2Bits(BitsAmount())(arr[0]); //!!tricky
```

CVF-83. FIXED

- **Category** Overflow/Underflow
- **Source** req.circom

Description Underflow might be possible here.

Recommendation Consider explaining in the comment if the inputs are range checked

Client Comment *1. Used 'DayFrom()'. and added comments. 2. Modiied the programmatic defition of 'IntDivide()'.*

```
24 (days, _) <== IntDivide(BitsTime())((maturityTime - req_.arg[2]) *
     ↪ enabled, 86400);
```

CVF-84. FIXED

- **Category** Overflow/Underflow
- **Source** order.circom

Description Underflow is possible here.

Recommendation Consider additionally checking that "amt" fits into "BitsUnsignedAmt" bits.

Client Comment *Resolved.*

```
37 _ <== Num2Bits(BitsUnsignedAmt())((orderLeaf[i] - amt) * enabled);
```

CVF-85. FIXED

- **Category** Suboptimal
- **Source** _mod.circom

Description This can produce Poseidon instance whose number of inputs is greater than batch.

Recommendation Consider recursively calling the ”PoseidonV2” template here.

Client Comment *Resolved.*

```
79 out <== Poseidon(len - batch + 1)(t);
```

CVF-86. FIXED

- **Category** Documentation
- **Source** request.circom

Description This logic is complicated and hard to understand.

Recommendation Consider documenting it in details.

Client Comment *Added comments.*

```
47 ImplyEq()(enabled, ori_state.accRoot     , Mux(3)([new_state.accRoot
   ↪      , acc_unit[0].oriRoot[0], acc_unit[0].oriRoot[0]],
   ↪ acc_unit_switch));
ImplyEq()(enabled, acc_unit[0].newRoot[0], Mux(3)([acc_unit[0].
   ↪ newRoot[0], acc_unit[0].newRoot[0], acc_unit[1].oriRoot[0]],
   ↪ acc_unit_switch));
ImplyEq()(enabled, new_state.accRoot     , Mux(3)([ori_state.accRoot
   ↪      , acc_unit[0].newRoot[0], acc_unit[1].newRoot[0]],
   ↪ acc_unit_switch));
```

CVF-87. FIXED

- **Category** Suboptimal
- **Source** request.circom

Description This logic is complicated and hard to understand.

Recommendation Consider documenting it in details.

Client Comment *Added comments.*

64
```
ImplyEq()(enabled, acc_leaf[0][0].tokens   , Mux(4)([acc_leaf[0][1].
   ↪ tokens   , token_unit[0].oriRoot[0], token_unit[0].oriRoot[0],
   ↪  token_unit[0].oriRoot[0]], token_unit_switch));
ImplyEq()(enabled, token_unit[0].newRoot[0], Mux(4)([token_unit[0].
   ↪ newRoot[0], token_unit[0].newRoot[0], token_unit[1].oriRoot
   ↪ [0], token_unit[0].newRoot[0]], token_unit_switch));
ImplyEq()(enabled, acc_leaf[0][1].tokens   , Mux(4)([acc_leaf[0][0].
   ↪ tokens   , token_unit[0].newRoot[0], token_unit[1].newRoot[0],
   ↪  token_unit[0].newRoot[0]], token_unit_switch));
ImplyEq()(enabled, acc_leaf[1][0].tokens   , Mux(4)([acc_leaf[1][0].
   ↪ tokens   , acc_leaf[1][0].tokens   , acc_leaf[1][0].tokens   ,
   ↪  token_unit[1].oriRoot[0]], token_unit_switch));
ImplyEq()(enabled, acc_leaf[1][1].tokens   , Mux(4)([acc_leaf[1][1].
   ↪ tokens   , acc_leaf[1][1].tokens   , acc_leaf[1][1].tokens   ,
   ↪  token_unit[1].newRoot[0]], token_unit_switch));
```

CVF-88. FIXED

- **Category** Overflow/Underflow
- **Source** request.circom

Description Overflow is possible here.

Recommendation Consider implementing a proper range check.

Client Comment *1. Added and assertion: 'BitsAmount() + 1 <= ConstFieldBits()'; 2. Performed a range check on 'lock_amt'.*

291
```
var lockAmtIfLend = req.amount + lockFeeAmtIfLend;
```

294
```
signal lockAmtIf2ndBuy <== expectedSellAmtIf2ndBuy + lockFeeIf2ndBuy
   ↪ ;
```

CVF-89. FIXED

- **Category** Overflow/Underflow
- **Source** request.circom

Description Underflow is possible here.

Client Comment *1. Added the constraint: p_req.matchedTime[0] < currentTime. 2. The 'txId' incrementally increases as new transactions are added. In this context, 'ori_order1.txid' belongs to an existing partial order request, while 'conn.txId' relates to a new order request. As such, the value of 'conn.txId' must0 be larger than that of 'ori_order1.txid'. 3. Added comments, indicating channel_in.args[0] as 'oriCumAmt0' and 'channel_in.args[1]' as 'oriCumAmt1'. 4. The matching mechanism inherently ensures an increasing trend in the accumulated amount. Consequently, both 'cumAmt0' and 'cumAmt1' will consistently increase, and the new accumulated amount will surpass the previous accumulated amount.*

415
```
ImplyEq()(enabled, 1, TagLessThan(BitsTime())([currentTime - p_req.
   ↪ matchedTime[0], 86400]));
```

442
```
Chunkify(2, [FmtOpcode(), FmtTxOffset()])(And()(enabled, isAuction),
   ↪  p_req.chunks, [req.reqType, conn.txId - ori_order1.txId]);
Chunkify(3, [FmtOpcode(), FmtTxOffset(), FmtPacked()])(And()(enabled
   ↪ , Or()(isSecondaryLimit, isSecondaryMarket)), p_req.chunks, [
   ↪ req.reqType, conn.txId - ori_order1.txId, packedAmt1]);
```

475
```
var matched_amt0 = order.cumAmt0 - channel_in.args[0];
var matched_amt1 = order.cumAmt1 - channel_in.args[1];
```

484
```
ImplyEq()(enabled, 1, TagLessThan(BitsTime())([currentTime - p_req.
   ↪ matchedTime[0], 86400]));
```

572
```
var matched_amt0 = order.cumAmt0 - channel_in.args[0];
var matched_amt1 = order.cumAmt1 - channel_in.args[1];
```

579
```
ImplyEq()(enabled, 1, TagLessThan(BitsTime())([currentTime - p_req.
   ↪ matchedTime[0], 86400]));
```

CVF-90. FIXED

- **Category** Overflow/Underflow
- **Source** request.circom

Description Overflow is possible here.

Recommendation Consider doing proper range checking.

Client Comment *1. Assertions were incorporated into each indicated template: BitsAmount() + 1 <= ConstFieldBits(). 2. A range check is already performed during the computation of TokenLeaf and OrderLeaf.*

428
```
signal newNewOrder1[LenOfOrderLeaf()] <== OrderLeaf_DeductLockedAmt
   ↪ ()(new_order1.arr, enabled, feeFromLocked + matched_amt0);
```

438
```
ImplyEqArr(LenOfTokenLeaf())(enabled, TokenLeaf_Unlock()(
   ↪ TokenLeaf_Deduct()(conn.tokenLeaf[1][0], enabled, matched_amt0
   ↪  + feeFromLocked), enabled, refund), conn.tokenLeaf[1][1]);
```

CVF-91. FIXED

- **Category** Overflow/Underflow
- **Source** request.circom

Description Overflow is possible here.

Recommendation Consider doing proper range checking.

Client Comment *1. Assertions were incorporated into each indicated template: 'BitsAmount() + 1 <= ConstFieldBits()'. 2. A range check is already performed during the computation of TokenLeaf and Chunkify.*

505
```
ImplyEqArr(LenOfTokenLeaf())(enabled, TokenLeaf_Unlock()(
   ↪ TokenLeaf_Deduct()(conn.tokenLeaf[1][0], enabled, matched_amt0
   ↪  + feeFromLocked), enabled, refund), conn.tokenLeaf[1][1]);
```

783
```
Chunkify(4, [FmtOpcode(), FmtAccId(), FmtTokenId(), FmtStateAmount()
   ↪ ])(enabled, p_req.chunks, [req.reqType, conn.accLeafId[0],
   ↪ conn.tokenLeafId[0], token.avl_amt + token.locked_amt]);
```

CVF-92. FIXED

- **Category** Overflow/Underflow
- **Source** mechanism.circom

Description Overflow is possible here.

Recommendation Consider range checking the arguments or clearly stating in a comment that the caller must guarantee the arguments to be in certain ranges and specify these ranges.

Client Comment *1. Added assertions for constants in each indicated template to prevent from overflow. 2. Added comments in "./circuits/zkTrueUp/src/mechanism.circom".*

```
15 signal temp <== interest * days + one * (365 - days);
```

```
17 (temp2, _) <== IntDivide(BitsAmount())(principal * temp, one * 365);
```

```
28 signal temp <== feeRate * days;
```

```
30 (temp2, _) <== IntDivide(BitsAmount())(matchedLendingAmt * temp, one
   ↪  * 365);
```

```
43 signal absInterest <== ((one - matchedInterest) - (matchedInterest -
   ↪  one)) * slt + (matchedInterest - one);
signal temp <== feeRate * days;
signal temp2 <== absInterest * temp;
```

```
47 (temp3, _) <== IntDivide(BitsAmount())(matchedBorrowingAmt * temp2,
   ↪ one * one * 365);
```

```
67 newCumLendingAmt <== oriCumLendingAmt + matchedAmt;
newCumTslAmt <== oriCumTslAmt + matchedTslAmt;
newCumBorrowingAmt <== oriCumBorrowingAmt + matchedAmt;
70 (remainingCollateralAmt, _) <== IntDivide(BitsAmount())(
   ↪ CollateralAmt * (BorrowingAmt - newCumBorrowingAmt), (
   ↪ BorrowingAmt - 1) * enabled + 1);
```

```
77 signal temp <== (days * (priceMQ - priceBQ) + 365 * priceBQ);
(newBQ, _) <== IntDivide(BitsAmount())((365 * MQ * priceBQ), (temp -
   ↪  1) * enabled + 1);
```

(84, 93, 95, 110, 115, 136, 141, 164, 182, 187, 207, 211, 220)

CVF-93. FIXED

- **Category** Overflow/Underflow
- **Source** mechanism.circom

Description Underflow is possible here.

Recommendation Consider either implementing some underflow protection or clearly stating that the caller must guarantee certain constraints regarding the argument values and specify these constraints.

Client Comment *1. Incorporated conditions related to interest rates when placing an order to prevent underflow. 2. Added comments in "./circuits/zkTrueUp/src/mechanism.circom" for better understanding.*

```
63 signal matchedAmt <== Min(BitsAmount())([LendingAmt -
      ↪ oriCumLendingAmt, BorrowingAmt - oriCumBorrowingAmt]);
```

```
70 (remainingCollateralAmt, _) <== IntDivide(BitsAmount())(
      ↪ CollateralAmt * (BorrowingAmt - newCumBorrowingAmt), (
      ↪ BorrowingAmt - 1) * enabled + 1);
   newCumCollateralAmt <== CollateralAmt - remainingCollateralAmt;
```

```
77 signal temp <== (days * (priceMQ - priceBQ) + 365 * priceBQ);
   (newBQ, _) <== IntDivide(BitsAmount())((365 * MQ * priceBQ), (temp -
      ↪  1) * enabled + 1);
```

```
87 signal temp3 <== (365 - days) * priceBQ;
```

```
89 (supMQ, _) <== IntDivide(BitsAmount())(temp1 + temp4, (temp2 - 1) *
      ↪ enabled + 1);
```

```
108 signal remainTakerSellAmt <== takerSellAmt - oriCumTakerSellAmt;
    signal remainMakerSellAmt <== makerSellAmt - oriCumMakerSellAmt;
110 signal temp7 <== (1 - (is_market_order * (1 - makerSide)));
    signal remainTakerBuyAmt <== (takerBuyAmt - oriCumTakerBuyAmt) *
       ↪ temp7;
    signal remainMakerBuyAmt <== makerBuyAmt - oriCumMakerBuyAmt;
```

(115, 203, 214)

CVF-94. FIXED

- **Category** Documentation
- **Source** mechanism.circom

Description These templates are very complicated and their logic is not documented.

Recommendation Consider adding comments with detailed explanation of what happens here. Without such explanation it is very hard to read the code.

Client Comment *1. All backend activities are now documented in the zkTrue-up circuit documentation, with items requiring circuit verification highlighted. 2. Added comments.*

```
97  template SecondMechanism(){
```

```
146 template AuctionInteract(){
```

```
167 template SecondaryInteract(){
```

```
190 template CalcFee(){
```

CVF-95. INFO

- **Category** Unclear behavior
- **Source** mechanism.circom

Description The expiredTime parameter of the request is never used, is it ok?

Client Comment *In the begining of 'DoReqInteract' function's legality check, this has been done.*

```
146 template AuctionInteract(){
```

9 Minor Issues

CVF-247. FIXED

- **Category** Procedural
- **Source** account.circom

Description These “tricky” comments are confusing.

Recommendation Consider removing them.

Client Comment *Resolved*.

```
14 signal output arr[LenOfAccLeaf()] <== [accLeaf[0], accLeaf[1] + 1,
      ↪ accLeaf[2]];//!!tricky
```

```
18 signal output arr[LenOfAccLeaf()] <== [accLeaf[0], accLeaf[1], 0];
      ↪ //!!tricky
```

```
23 ImplyEq()(enabled, accLeaf[0], 0);//!!tricky
```

```
29 ImplyEq()(enabled, accLeaf[1], nonce);//!!tricky
```

CVF-248. FIXED

- **Category** Procedural
- **Source** fee.circom

Description This ”tricky” comment is confusing.

Recommendation Consider removing it.

Client Comment *Resolved*.

```
8 signal output arr[LenOfFeeLeaf()] <== [feeLeaf[0] + amount];//!!
     ↪ tricky
```

CVF-249. FIXED

- **Category** Procedural
- **Source** prep_req.circom

Description This ”tricky” comment is confusing.

Recommendation Consider removing it.

Client Comment *Resolved*.

```
5 signal output reqType <== preprocessedReq[0]; //!!tricky
```

CVF-250. FIXED

- **Category** Bad naming
- **Source** _mod.circom

Description The argument name ”offset” is confusing, as actually it is the slice length.

Recommendation Consider renaming.

Client Comment *Resolved*.

```
18 template Slice(len, start, offset){
```

CVF-251. FIXED

- **Category** Procedural
- **Source** _mod.circom

Description These ”tricky” comments are confusing.

Recommendation Consider removing them.

Client Comment *Resolved*.

```
71  _ <== Num2Bits(BitsEpoch())(arr[6]); //!!tricky
    _ <== Num2Bits(BitsEpoch())(arr[7]); //!!tricky
```

```
196 _ <== Num2Bits(BitsAmount())(arr[0]); //!!tricky
```

CVF-252. FIXED

- **Category** Documentation
- **Source** req.circom

Recommendation Consider specifying in a comment what does ”req_.arg[2]” mean.

Client Comment *Resolved*.

```
15 ImplyEq()(enabled, 1, TagLessThan(BitsTime())([currentTime, req_.arg
      ↪ [2]]));
```

CVF-253. FIXED

- **Category** Bad naming
- **Source** order.circom

Recommendation The name should start with a lower case letter.

Client Comment *Resolved*.

```
5 signal input CumAmt0;
```

CVF-254. FIXED

- **Category** Documentation
- **Source** order.circom

Recommendation Consider adding a comment explaining what is ”req.arg[5]”.

Client Comment *Resolved*.

```
57 signal isFull1 <== TagIsEqual()([req.arg[5], order.cumAmt1]);
```

CVF-255. FIXED

- **Category** Documentation
- **Source** order.circom

Recommendation Consider adding a comment explaining what is ”req.arg[8]”.

Client Comment *Resolved.*

```
60 signal isFullIfSecondary <== (isFull0 - isFull1) * req.arg[8] +
      ↪ isFull1;
```

CVF-256. FIXED

- **Category** Suboptimal
- **Source** _mod.circom

Recommendation Consider extracting this common formula into a binary selector template.

Client Comment *Resolved.*

```
27 signal output out <== (in[0] - in[1]) * slt + in[1];
```

```
32 signal output out <== (in[0] - in[1]) * (1 - slt) + in[1];
```

```
38 in_0 === (in_1 - in_0) * enabled + in_0;
```

CVF-257. FIXED

- **Category** Bad datatype
- **Source** _mod.circom

Recommendation The value 253 should be a named constant.

Client Comment *Added the named constant in ”./circuits/zkTrueUp/src/const/_mod.circom”.*

```
56 signal n2b[253 - bits_divisor] <== Num2Bits(253 - bits_divisor)(
      ↪ quotient);
```

CVF-258. FIXED

- **Category** Bad datatype
- **Source** _mod.circom

Recommendation The value ”27” should be a template parameter.

Client Comment *Removed ’FixPtMul()’.*

```
62 (out, _) <== IntDivide(27)(in_0 * in_1, 10 ** 8);
```

CVF-259. FIXED

- **Category** Suboptimal
- **Source** req_type.circom

Description Moving this constant to the end of the list would make the constant values to go in sequence, and would make it harder to not update this constant when adding a new request type.

Client Comment *Resolved.*

```
3 function ReqTypeCount()                    { return 26; }
```

CVF-260. FIXED

- **Category** Documentation
- **Source** _mod.circom

Recommendation Consider adding comments with request types after the values. Currently it is very hard to tell what value corresponds to what request type.

Client Comment *Resolved.*

```
8  return [1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
   ↪ 1, 1, 0, 1, 1, 1];
```

```
11 return [0, 3, 2, 2, 2, 2, 3, 3, 1, 1, 4, 3, 1, 1, 1, 2, 1, 1, 1, 1,
   ↪ 1, 1, 2, 2, 2, 1];
```

```
14 return [0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   ↪ 0, 1, 0, 1, 1, 0];
```

CVF-261. FIXED

- **Category** Documentation
- **Source** request.circom

Description This comment sounds odd.

Recommendation Consider rephrasing.

Client Comment *Replaced this comment with assertion.*

```
10 //This template already has a set of const values defined. If the
     ↪ constants is changed, this template maybe report an error.
```

CVF-262. FIXED

- **Category** Documentation
- **Source** request.circom

Description These templates are very complicated and hard to read.

Recommendation Consider documenting them line by line.

Client Comment *1. All behaviors in the Backend are now documented in the zkTrueUp circuit documentation, with circuit verification items highlighted. 2. Added comments.*

```
115 template DoReqUnknown(){
133 template DoReqRegister(){
158 template DoReqDeposit(){
182 template DoReqTransfer(){
216 template DoReqWithdraw(){
244 template DoReqForcedWithdraw(){
270 template DoReqPlaceOrder(){
327 template DoReqStart(){
449 template DoReqEnd(){
515 template DoReqSecondMarketOrder(){
549 template DoReqSecondMarketEnd(){
599 template DoReqCancel(){
632 template DoReqSetAdminTsAddr(){
653 template DoReqIncreaseEpoch(){
678 template DoReqCreateBondToken(){
703 template DoReqRedeem(){
739 template DoReqWithdrawFee(){
763 template DoReqEvacuation(){
```

CVF-263. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment the meaning of each non-zero argument value here.

Client Comment *Resolved*.

```
142 component conn = Conn(0, 0, 0, 1, 0, 0, 0, 0);
167 component conn = Conn(0, 0, 0, 1, 0, 1, 0, 0);
191 component conn = Conn(0, 0, 0, 2, 0, 3, 0, 0);
225 component conn = Conn(0, 0, 0, 1, 0, 1, 0, 0);
253 component conn = Conn(0, 0, 0, 1, 0, 1, 0, 0);
279 component conn = Conn(0, 0, 1, 1, 1, 1, 0, 0);
336 component conn = Conn(0, 0, 1, 0, 0, 0, 0, 0);
368 component conn = Conn(1, 1, 1, 1, 0, 2, 0, 0);
458 component conn = Conn(1, 1, 1, 1, 0, 2, 0, 0);
524 component conn = Conn(0, 0, 0, 1, 1, 0, 0, 0);
558 component conn = Conn(1, 1, 0, 1, 0, 2, 0, 0);
608 component conn = Conn(0, 0, 1, 1, 0, 1, 0, 0);
641 component conn = Conn(0, 0, 0, 0, 0, 0, 0, 1);
662 component conn = Conn(0, 0, 0, 0, 0, 0, 1, 0);
687 component conn = Conn(0, 1, 0, 0, 0, 0, 0, 0);
712 component conn = Conn(0, 1, 0, 1, 0, 2, 0, 0);
748 component conn = Conn(1, 0, 0, 0, 0, 0, 0, 0);
772 component conn = Conn(0, 0, 0, 1, 0, 1, 0, 0);
```

CVF-264. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment, what ”req.arg[0]” is.

Client Comment *Resolved*.

```
149 ImplyEq()(enabled, conn.accLeafId[0], req.arg[0]);
```

```
171 ImplyEq()(enabled, conn.accLeafId[0], req.arg[0]);
```

```
200 ImplyEq()(enabled, conn.accLeafId[1], req.arg[0]);
```

```
210 Chunkify(5, [FmtOpcode(), FmtAccId(), FmtTokenId(), FmtPacked(),
    ↪ FmtAccId()])(enabled, p_req.chunks, [req.reqType, req.accId,
    ↪ req.tokenId, packedAmt, req.arg[0]]);
```

```
259 ImplyEq()(enabled, conn.accLeafId[0], req.arg[0]);
```

```
778 ImplyEq()(enabled, conn.accLeafId[0], req.arg[0]);
```

CVF-265. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment, what ”req.arg[6]” is.

Client Comment *Resolved*.

```
150 ImplyEqArr(LenOfAccLeaf())(enabled, AccLeaf_Register()(conn.accLeaf
    ↪ [0][0], req.arg[6]), conn.accLeaf[0][1]);
```

```
152 Chunkify(3, [FmtOpcode(), FmtAccId(), FmtHashedPubKey()])(enabled,
    ↪ p_req.chunks, [req.reqType, conn.accLeafId[0], req.arg[6]]);
```

```
645 ImplyEq()(enabled, conn.adminTsAddr[1], req.arg[6]);
```

CVF-266. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment, what ”req.arg[8]” is.

Client Comment *Resolved*.

```
288 signal is2ndBuy <== And()(is2nd, Not()(Bool()(req.arg[8])));
    signal is2ndSell <== And()(is2nd, Bool()(req.arg[8]));
```

```
420 ImplyEq()(And()(enabled, Or()(isSecondaryLimit, isSecondaryMarket)),
      ↪  conn.feeLeafId[0], Mux(2)([ori_order1_req.tokenId,
      ↪ ori_order1_req.arg[4]], ori_order1_req.arg[8]));
```

```
423 ImplyEq()(And()(enabled, Or()(isSecondaryLimit, isSecondaryMarket)),
      ↪  conn.bondLeafId[0], Mux(2)([ori_order1_req.arg[4],
      ↪ ori_order1_req.tokenId], ori_order1_req.arg[8]));
```

```
489 ImplyEq()(And()(enabled, isSecondaryLimit), conn.feeLeafId[0], Mux
      ↪ (2)([order_req.tokenId, order_req.arg[4]], order_req.arg[8]));
```

```
492 ImplyEq()(And()(enabled, isSecondaryLimit), conn.bondLeafId[0], Mux
      ↪ (2)([order_req.arg[4], order_req.tokenId], order_req.arg[8]));
```

```
583 ImplyEq()(enabled, conn.feeLeafId[0], Mux(2)([order_req.tokenId,
      ↪ order_req.arg[4]], order_req.arg[8]));
    ImplyEq()(enabled, conn.bondLeafId[0], Mux(2)([order_req.arg[4],
      ↪ order_req.tokenId], order_req.arg[8]));
```

CVF-267. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment, what ”req.arg[1]” is.

Client Comment *Resolved*.

```
290 signal lockFeeAmtIfLend <== AuctionCalcLendFee()(req.fee0, req.
       ↪ amount, DaysFrom()(enabled, p_req.matchedTime[0], req.arg[1]))
       ↪ ;
```

```
292 signal expectedSellAmtIf2ndBuy <== CalcNewBQ()(enabled, req.arg[5],
       ↪ req.amount, req.arg[5], req.amount, Req_DaysFromExpired()(
       ↪ p_req.req, enabled, req.arg[1]));
    signal lockFeeIf2ndBuy <== SecondCalcFee()(req.arg[5], Max(BitsRatio
       ↪ ())([req.fee0, req.fee1]), DaysFrom()(enabled, p_req.
       ↪ matchedTime[0], req.arg[1]));
```

```
319 Chunkify(7, [FmtOpcode(), FmtAccId(), FmtTokenId(), FmtPacked(),
       ↪ FmtPacked(), FmtTime(), FmtTime()])(And()(enabled, isLend),
       ↪ p_req.chunks, [req.reqType, req.accId, req.tokenId,
       ↪ packedAmount0, packedFee0, req.arg[1], p_req.matchedTime[0]]);
```

```
422 ImplyEq()(And()(enabled, isAuction), bond.maturity, ori_order1_req.
       ↪ arg[1]);
```

```
491 ImplyEq()(And()(enabled, isAuction), bond.maturity, order_req.arg
       ↪ [1]);
```

```
621 ImplyEq()(And()(enabled, TagIsEqual()([req.reqType,
       ↪ ReqTypeNumUserCancel()])), order.txId, req.arg[1]);
```

```
695 ImplyEq()(enabled, bond.maturity, req.arg[1]);
```

CVF-268. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment, what ”req.arg[5]” is.

Client Comment *Resolved*.

```
292 signal expectedSellAmtIf2ndBuy <== CalcNewBQ()(enabled, req.arg[5],
       ↪ req.amount, req.arg[5], req.amount, Req_DaysFromExpired()(
       ↪ p_req.req, enabled, req.arg[1]));
    signal lockFeeIf2ndBuy <== SecondCalcFee()(req.arg[5], Max(BitsRatio
       ↪ ())([req.fee0, req.fee1]), DaysFrom()(enabled, p_req.
       ↪ matchedTime[0], req.arg[1]));
```

```
315 signal packedAmount1 <== Fix2Float()(req.arg[5]);
```

```
441 signal packedAmt1 <== Fix2Float()(ori_order1_req.arg[5]);
```

```
542 signal packedAmount1 <== Fix2Float()(req.arg[5]);
```

CVF-269. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment, what ”req.arg[7]” is.

Client Comment *Resolved*.

```
300 ImplyEq()(enabled, req.arg[7], Mux(2)([conn.epoch[0][0], conn.epoch
       ↪ [1][0]], p_req.nullifierTreeId[0]));
```

```
531 ImplyEq()(enabled, req.arg[7], Mux(2)([conn.epoch[0][0], conn.epoch
       ↪ [1][0]], p_req.nullifierTreeId[0]));
```

CVF-270. FIXED

- **Category** Bad datatype
- **Source** request.circom

Recommendation The value 86400 should be a named constant, i.e. a function returning a constant value.

Client Comment *Added the named constant in "./circuits/zkTrueUp/src/const/_mod.circom".*

```
303 ImplyEq()(enabled, 1, TagLessThan(BitsTime())([currentTime - p_req.
       ↪ matchedTime[0], 86400]));
```

```
415 ImplyEq()(enabled, 1, TagLessThan(BitsTime())([currentTime - p_req.
       ↪ matchedTime[0], 86400]));
```

```
484 ImplyEq()(enabled, 1, TagLessThan(BitsTime())([currentTime - p_req.
       ↪ matchedTime[0], 86400]));
```

```
579 ImplyEq()(enabled, 1, TagLessThan(BitsTime())([currentTime - p_req.
       ↪ matchedTime[0], 86400]));
```

CVF-271. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment, what "req.arg[2]" is.

Client Comment *Resolved.*

```
321 Chunkify(10, [FmtOpcode(), FmtAccId(), FmtTokenId(), FmtPacked(),
       ↪ FmtPacked(), FmtPacked(), FmtTokenId(), FmtPacked(), FmtTime()
       ↪ , FmtTime()])(And()(enabled, is2nd), p_req.chunks, [req.
       ↪ reqType, req.accId, req.tokenId, packedAmount0, packedFee0,
       ↪ packedFee1, req.arg[4], packedAmount1, req.arg[2], p_req.
       ↪ matchedTime[0]]);
```

```
544 Chunkify(8, [FmtOpcode(), FmtAccId(), FmtTokenId(), FmtPacked(),
       ↪ FmtPacked(), FmtTokenId(), FmtPacked(), FmtTime()])(enabled,
       ↪ p_req.chunks, [req.reqType, req.accId, req.tokenId,
       ↪ packedAmount0, packedFee0, req.arg[4], packedAmount1, req.arg
       ↪ [2]]);
```

CVF-272. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment, what ”req.arg[4]” is.

```
321 Chunkify(10, [FmtOpcode(), FmtAccId(), FmtTokenId(), FmtPacked(),
    ↪ FmtPacked(), FmtPacked(), FmtTokenId(), FmtPacked(), FmtTime()
    ↪ , FmtTime()])(And()(enabled, is2nd), p_req.chunks, [req.
    ↪ reqType, req.accId, req.tokenId, packedAmount0, packedFee0,
    ↪ packedFee1, req.arg[4], packedAmount1, req.arg[2], p_req.
    ↪ matchedTime[0]]);

420 ImplyEq()(And()(enabled, Or()(isSecondaryLimit, isSecondaryMarket)),
    ↪  conn.feeLeafId[0], Mux(2)([ori_order1_req.tokenId,
    ↪ ori_order1_req.arg[4]], ori_order1_req.arg[8]));

423 ImplyEq()(And()(enabled, Or()(isSecondaryLimit, isSecondaryMarket)),
    ↪  conn.bondLeafId[0], Mux(2)([ori_order1_req.arg[4],
    ↪ ori_order1_req.tokenId], ori_order1_req.arg[8]));

425 ImplyEq()(And()(enabled, Or()(isSecondaryLimit, isSecondaryMarket)),
    ↪  conn.tokenLeafId[0], ori_order1_req.arg[4]);

488 ImplyEq()(And()(enabled, isAuction), conn.feeLeafId[0], order_req.
    ↪ arg[4]);
    ImplyEq()(And()(enabled, isSecondaryLimit), conn.feeLeafId[0], Mux
    ↪ (2)([order_req.tokenId, order_req.arg[4]], order_req.arg[8]));
490 ImplyEq()(And()(enabled, isAuction), bond.baseTokenId, order_req.arg
    ↪ [4]);

492 ImplyEq()(And()(enabled, isSecondaryLimit), conn.bondLeafId[0], Mux
    ↪ (2)([order_req.arg[4], order_req.tokenId], order_req.arg[8]));
    ImplyEq()(enabled, conn.tokenLeafId[0], order_req.arg[4]);

544 Chunkify(8, [FmtOpcode(), FmtAccId(), FmtTokenId(), FmtPacked(),
    ↪ FmtPacked(), FmtTokenId(), FmtPacked(), FmtTime()])(enabled,
    ↪ p_req.chunks, [req.reqType, req.accId, req.tokenId,
    ↪ packedAmount0, packedFee0, req.arg[4], packedAmount1, req.arg
    ↪ [2]]);

583 ImplyEq()(enabled, conn.feeLeafId[0], Mux(2)([order_req.tokenId,
    ↪ order_req.arg[4]], order_req.arg[8]));
    ImplyEq()(enabled, conn.bondLeafId[0], Mux(2)([order_req.arg[4],
    ↪ order_req.tokenId], order_req.arg[8]));
    ImplyEq()(enabled, conn.tokenLeafId[0], order_req.arg[4]);
```

(694)

CVF-273. FIXED

- **Category** Documentation
- **Source** request.circom

Recommendation Consider explaining in a comment, what ”req.arg[3]” is.

Client Comment *Resolved*.

```
352 signal packedInterest <== Fix2Float()(req.arg[3]);
```

```
357 channelOut <== Channel_New()(conn.orderLeaf[0][0], [order.cumAmt0,
    ↪ order.cumAmt1, req.arg[3] * isAuctionStart, 0, 0]);
```

```
408 ImplyEq()(And()(enabled, isAuction), 1, TagGreaterEqThan(BitsRatio()
    ↪ )([ori_order1_req.arg[3], channel_in.args[3]]));
```

```
445 signal channelOutIfAuction[LenOfChannel()] <== Channel_New()(
    ↪ newBorrow, [channel_in.args[0], channel_in.args[1], channel_in
    ↪ .args[2], ori_order1_req.arg[3], 0]);
```

CVF-274. FIXED

- **Category** Suboptimal
- **Source** request.circom

Description The expression ”And()(enabled, isAuction)” is calculated several times.

Recommendation Consider calculating once and reusing.

Client Comment *Resolved*.

```
406 ImplyEq()(And()(enabled, isAuction), ori_order0_req.reqType,
    ↪ ReqTypeNumAuctionBorrow());
    ImplyEq()(And()(enabled, isAuction), ori_order1_req.reqType,
    ↪ ReqTypeNumAuctionLend());
    ImplyEq()(And()(enabled, isAuction), 1, TagGreaterEqThan(BitsRatio()
    ↪ )([ori_order1_req.arg[3], channel_in.args[3]]));
```

CVF-275. FIXED

- **Category** Suboptimal
- **Source** request.circom

Description The expression ”And()(enabled, isSecondaryLimit)” is calculated several times.

Recommendation Consider calculating once and reusing.

Client Comment *Resolved*.

```
409 ImplyEq()(And()(enabled, isSecondaryLimit), ori_order0_req.reqType,
       ↪ ReqTypeNumSecondLimitOrder());
410 ImplyEq()(And()(enabled, isSecondaryLimit), ori_order1_req.reqType,
       ↪ ReqTypeNumSecondLimitOrder());
```

CVF-276. FIXED

- **Category** Suboptimal
- **Source** request.circom

Description The expression ”And()(enabled, isSecondaryMarket)” is calculated several times.

Recommendation Consider calculating once and reusing.

Client Comment *Resolved*.

```
411 ImplyEq()(And()(enabled, isSecondaryMarket), ori_order0_req.reqType,
       ↪  ReqTypeNumSecondMarketOrder());
    ImplyEq()(And()(enabled, isSecondaryMarket), ori_order1_req.reqType,
       ↪  ReqTypeNumSecondLimitOrder());
```

CVF-277. FIXED

- **Category** Suboptimal
- **Source** mechanism.circom

Description In this comment, the constant name ”one” and its value ”10 ** 8” are used together.

Recommendation Consider using either the name of the value but not both.

Client Comment *Resolved*.

```
13 // debtAmt := principal * (interest * days + one * (365 - days)) /
       ↪ (365 * (10 ** 8))
```

CVF-278. FIXED

- **Category** Suboptimal
- **Source** mechanism.circom

Recommendation This formula could be simplified as: (2 *slt-one)* (one - matchedInterest)

Client Comment *The formula should be '(2 * slt - 1)* (one - matchedInterest)'. Resolved.*

```
43 signal absInterest <== ((one - matchedInterest) - (matchedInterest -
      ↪  one)) * slt + (matchedInterest - one);
```

CVF-279. FIXED

- **Category** Documentation
- **Source** spec.circom

Recommendation Consider explaining in a comment how this value was calculated.

Client Comment *Added comments.*

```
26 return
      ↪ 12657967895078469224578163781222964703015536623478048995515841
      ↪
04870723496262;
```

CVF-280. FIXED

- **Category** Suboptimal
- **Source** evacuation.circom

Description The formulas used to calculate bit indexes are cumbersome and error-prone.

Recommendation Consider using a running index counter instead. Also, consider extracting a template that copies a range of signals from one array into another.

Client Comment *1. Added new operations with arrays. 2. Redesigned template 'calcCommitment'.*

```
28 component sha256 = Sha256(256 * 4 + (BitsChunk() + 8) * (
      ↪ NumOfChunksForEvacuation()));

34     sha256.in[256 * 1 - 1 - i] <== n2b_oriStateRoot[i];
       sha256.in[256 * 2 - 1 - i] <== n2b_newStateRoot[i];
       sha256.in[256 * 3 - 1 - i] <== n2b_newTsRoot[i];
       sha256.in[256 * 4 - 1 - i] <== n2b_currentTime[i];

40     sha256.in[256 * 1 - 1 - i] <== 0;
       sha256.in[256 * 2 - 1 - i] <== 0;
       sha256.in[256 * 3 - 1 - i] <== 0;
       sha256.in[256 * 4 - 1 - i] <== 0;

46     sha256.in[(256 * 4) + (8 * (i + 1)) - 1 - 0] <== isCriticalChunk
          ↪ [i];

48         sha256.in[(256 * 4) + (8 * (i + 1)) - 1 - j] <== 0;

50         sha256.in[(256 * 4) + (8 * NumOfChunksForEvacuation()) + (
              ↪ BitsChunk() * (i + 1)) - 1 - j] <== n2b_chunks[i].out[
              ↪ j];

85     bits_chunks[i] <== bits_reqType[1 * 8 - i - 1];

87     bits_chunks[1 * 8 + i] <== bits_accId[4 * 8 - i - 1];

89     bits_chunks[5 * 8 + i] <== bits_tokenId[2 * 8 - i - 1];

91     bits_chunks[7 * 8 + i] <== bits_amount[16 * 8 - i - 1];

98         tmp[j] = bits_chunks[BitsChunk() * (i + 1) - j - 1];
```

CVF-281. FIXED

- **Category** Procedural
- **Source** evacuation.circom

Recommendation 253 and 254 are proof-system dependent constants and should be named.

Client Comment *Added the constants in "./circuits/zkTrueUp/src/const/_mod.circom".*

```
29 signal n2b_oriStateRoot[254] <== Num2Bits_strict()(oriStateRoot);
30 signal n2b_newStateRoot[254] <== Num2Bits_strict()(newStateRoot);
   signal n2b_newTsRoot[254] <== Num2Bits_strict()(newTsRoot);
   signal n2b_currentTime[254] <== Num2Bits_strict()(currentTime);
   for(var i = 0; i < 254; i++){
```

```
39 for(var i = 254; i < 256; i++){
```

```
52 component b2n_commitment = Bits2Num(253);
   for(var i = 0; i < 253; i++)
```

CVF-282. INFO

- **Category** Suboptimal
- **Source** evacuation.circom

Recommendation The signal "expectedStateRoot" is redundant. Just use "stateRoot" instead.

Client Comment *In accordance with Circom syntax, we must first declare a signal using '⇐' to store a value, then subsequently use '===' to verify the equality of two values.*

```
73 signal expectedStateRoot <== Poseidon(2)([tsRoot, accRoot]);
```

CVF-283. FIXED

- **Category** Procedural
- **Source** normal.circom

Recommendation Constants 254 and 253 are dependent on the proof system and should be named.

Client Comment *Added the constants in "./circuits/zkTrueUp/src/const/_mod.circom".*

```
89 signal n2b_oriStateRoot[254] <== Num2Bits_strict()(oriStateRoot);
90 signal n2b_newStateRoot[254] <== Num2Bits_strict()(newStateRoot);
   signal n2b_newTsRoot[254] <== Num2Bits_strict()(newTsRoot);
   signal n2b_currentTime[254] <== Num2Bits_strict()(currentTime);
   for(var i = 0; i < 254; i++){
```

```
99 for(var i = 254; i < 256; i++){
```

```
113 for(var i = 0; i < 253; i++)
```

CVF-284. FIXED

- **Category** Suboptimal
- **Source** normal.circom

Description The formulas used to calculate bit indexes are cumbersome and error-prone.

Recommendation Consider using a running index counter instead. Also, consider extracting a template that copies a range of signals from one array into another.

Client Comment *1. Added new operations with arrays. 2. Redesigned template 'calcCommitment'.*

```
94  sha256.in[256 * 1 - 1 - i] <== n2b_oriStateRoot[i];
    sha256.in[256 * 2 - 1 - i] <== n2b_newStateRoot[i];
    sha256.in[256 * 3 - 1 - i] <== n2b_newTsRoot[i];
    sha256.in[256 * 4 - 1 - i] <== n2b_currentTime[i];

100 sha256.in[256 * 1 - 1 - i] <== 0;
    sha256.in[256 * 2 - 1 - i] <== 0;
    sha256.in[256 * 3 - 1 - i] <== 0;
    sha256.in[256 * 4 - 1 - i] <== 0;

106 sha256.in[(256 * 4) + (8 * (i + 1)) - 1 - 0] <== isCriticalChunk[i];

108     sha256.in[(256 * 4) + (8 * (i + 1)) - 1 - j] <== 0;

110     sha256.in[(256 * 4) + (8 * NumOfChunks()) + (BitsChunk() * (i +
        ↪ 1)) - 1 - j] <== n2b_chunks[i].out[j];
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting