

Report
v. 6.0

Customer
zkLink



Circuits and contracts audit zkLink Protocol

9th August 2023

Report prepared by
ABDK
Consulting

Contents

- 1 Changelog** **7**

- 2 Introduction** **8**

- 3 Project scope** **9**

- 4 Methodology** **11**

- 5 Our findings** **12**

- 6 Critical Issues** **13**
 - CVF-1. FIXED 13
 - CVF-2. FIXED 13
 - CVF-3. FIXED 14
 - CVF-4. FIXED 14
 - CVF-5. FIXED 14
 - CVF-7. FIXED 15
 - CVF-8. FIXED 15
 - CVF-9. FIXED 15
 - CVF-10. FIXED 16
 - CVF-11. FIXED 16
 - CVF-12. FIXED 16
 - CVF-13. FIXED 17

- 7 Major Issues** **18**
 - CVF-6. FIXED 18
 - CVF-17. FIXED 18
 - CVF-24. FIXED 19
 - CVF-27. FIXED 19
 - CVF-28. FIXED 20
 - CVF-29. FIXED 20
 - CVF-30. FIXED 21
 - CVF-32. FIXED 21
 - CVF-33. FIXED 21
 - CVF-36. FIXED 22
 - CVF-40. FIXED 22
 - CVF-41. FIXED 22
 - CVF-42. FIXED 23
 - CVF-43. FIXED 23
 - CVF-44. FIXED 24
 - CVF-46. FIXED 24
 - CVF-47. FIXED 25
 - CVF-48. FIXED 25
 - CVF-54. FIXED 25

CVF-55. FIXED	26
CVF-57. FIXED	26
CVF-59. FIXED	26
CVF-60. FIXED	27
8 Moderate Issues	28
CVF-14. INFO	28
CVF-15. INFO	28
CVF-18. INFO	29
CVF-20. INFO	29
CVF-22. INFO	30
CVF-23. INFO	30
CVF-25. INFO	30
CVF-26. INFO	31
CVF-37. INFO	31
CVF-61. INFO	32
CVF-62. INFO	32
CVF-63. INFO	33
CVF-64. INFO	33
CVF-65. INFO	34
CVF-66. INFO	34
CVF-67. INFO	35
CVF-68. INFO	35
CVF-69. INFO	36
CVF-70. INFO	36
CVF-71. FIXED	36
CVF-72. FIXED	37
CVF-73. INFO	38
CVF-74. FIXED	39
CVF-75. FIXED	39
CVF-76. INFO	40
CVF-77. INFO	40
CVF-78. INFO	41
9 Minor Issues	42
CVF-79. FIXED	42
CVF-80. INFO	42
CVF-81. FIXED	42
CVF-82. FIXED	43
CVF-83. INFO	43
CVF-84. INFO	43
CVF-85. INFO	44
CVF-86. FIXED	44
CVF-87. INFO	44
CVF-88. INFO	45
CVF-89. INFO	45
CVF-90. FIXED	46

CVF-91. FIXED	46
CVF-92. FIXED	47
CVF-93. INFO	48
CVF-94. FIXED	48
CVF-95. FIXED	48
CVF-96. INFO	49
CVF-97. FIXED	49
CVF-98. FIXED	49
CVF-99. INFO	50
CVF-100. INFO	50
CVF-101. INFO	50
CVF-102. FIXED	51
CVF-103. FIXED	51
CVF-104. FIXED	51
CVF-105. FIXED	52
CVF-106. INFO	52
CVF-107. FIXED	52
CVF-108. FIXED	52
CVF-109. FIXED	53
CVF-110. FIXED	53
CVF-111. INFO	53
CVF-112. FIXED	53
CVF-113. FIXED	54
CVF-114. FIXED	54
CVF-115. FIXED	54
CVF-116. FIXED	55
CVF-117. INFO	55
CVF-118. FIXED	55
CVF-119. INFO	56
CVF-120. FIXED	56
CVF-121. FIXED	57
CVF-122. INFO	57
CVF-123. FIXED	58
CVF-124. FIXED	58
CVF-125. FIXED	58
CVF-126. FIXED	59
CVF-127. FIXED	59
CVF-128. INFO	59
CVF-129. INFO	59
CVF-130. INFO	60
CVF-131. INFO	60
CVF-132. INFO	60
CVF-133. FIXED	60
CVF-134. INFO	61
CVF-135. INFO	61
CVF-136. INFO	61

CVF-137. INFO	62
CVF-138. FIXED	62
CVF-139. INFO	63
CVF-140. INFO	64
CVF-141. FIXED	64
CVF-142. FIXED	64
CVF-143. INFO	65
CVF-144. INFO	65
CVF-145. INFO	66
CVF-146. FIXED	67
CVF-147. INFO	68
CVF-148. INFO	68
CVF-149. INFO	68
CVF-150. FIXED	69
CVF-151. INFO	69
CVF-152. FIXED	70
CVF-153. FIXED	70
CVF-154. FIXED	70
CVF-155. INFO	71
CVF-156. FIXED	72
CVF-157. FIXED	73
CVF-158. FIXED	73
CVF-159. INFO	74
CVF-160. INFO	74
CVF-161. FIXED	75
CVF-162. INFO	75
CVF-163. INFO	75
CVF-164. FIXED	76
CVF-165. FIXED	76
CVF-166. FIXED	77
CVF-167. FIXED	77
CVF-168. FIXED	77
CVF-169. FIXED	78
CVF-170. FIXED	78
CVF-171. FIXED	78
CVF-172. INFO	79
CVF-173. INFO	79
CVF-174. INFO	79
CVF-175. INFO	80
CVF-176. INFO	80
CVF-177. INFO	80
CVF-178. INFO	81
CVF-179. INFO	81
CVF-180. FIXED	82
CVF-181. FIXED	82
CVF-182. FIXED	82

CVF-183. FIXED 82
CVF-184. FIXED 83
CVF-185. FIXED 83
CVF-186. INFO 83
CVF-187. FIXED 83

1 Changelog

#	Date	Author	Description
0.1	30.01.23	A. Zveryanskaya	Initial Draft
0.2	30.01.23	A. Zveryanskaya	Minor revision
1.0	31.01.23	A. Zveryanskaya	Release
1.1	07.02.23	A. Zveryanskaya	Issues classification added
1.2	07.02.23	A. Zveryanskaya	Syntax highlighted
2.0	07.02.23	A. Zveryanskaya	Release
2.1	21.04.23	A. Zveryanskaya	Original Repository link fixed
3.0	21.04.23	A. Zveryanskaya	Release
3.1	03.07.23	A. Zveryanskaya	Solidity issues added
3.2	03.07.23	A. Zveryanskaya	All repository links updated
3.3	03.07.23	A. Zveryanskaya	List of Solidity files added
4.0	04.07.23	A. Zveryanskaya	Release
4.1	08.08.23	A. Zveryanskaya	CVF-6 downgraded
4.2	08.08.23	A. Zveryanskaya	CVF-3, 5, 11, 12 updated
4.3	08.08.23	A. Zveryanskaya	CVF-14, 15, 18, 20, 22, 23, 25, 26, 37 downgraded
4.4	08.08.23	A. Zveryanskaya	CVF-16, 19, 31, 34, 38, 39, 45, 49, 50, 51,52,53,56, 58 removed
4.5	08.08.23	A. Zveryanskaya	CVF-27, 28, 29, 30, 32, 36, 57 fixed
5.0	08.08.23	A. Zveryanskaya	Release
5.0	09.08.23	A. Zveryanskaya	CVF-21, 35 removed
6.0	09.08.23	A. Zveryanskaya	Release

2 Introduction

All modifications to this document are prohibited. Violators will be prosecuted to the full extent of the U.S. law.

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.

zkLink is a trading-focused multi-chain L2 network with unified liquidity secured by ZK-Rollups.

3 Project scope

We were asked to review several repositories:

- [Original Rust Code](#)
- [Original Solidity Code](#)

And corresponding fixes:

- [Rust Fixes](#)
- [Solidity Fixes](#)

Rust Files:

/		
account.rs	allocated_structures.rs	circuit.rs
element.rs	exit_circuit.rs	operation.rs
serialization.rs	signature.rs	utils.rs
witness/		
change_pubkey_offchain.rs	withdraw.rs	deposit.rs
forced_Exit.rs	full_exit.rs	noop.rs
order_matching.rs	transfer.rs	transfer_to_new.rs
utils.rs	mod.rs	
op_circuit/		
order_matching.rs	deposit.rs	forced_exit.rs
transfer.rs	transfer_to_new.rs	

And Solidity Files:

/			
	DeployFactory.sol	EmptyVerifier.sol	Storage.sol
	ZkLink.sol	ZkLinkPeriphery.sol	Small files
bridge/			
	LayerZeroBridge.sol	LayerZeroStorage.sol	
interfaces/			
	Iverifier.sol		

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.

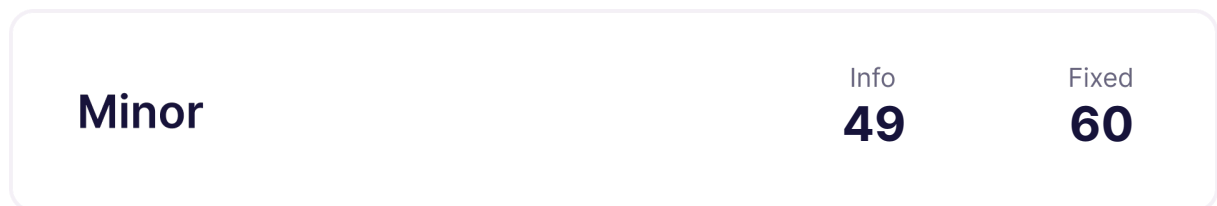
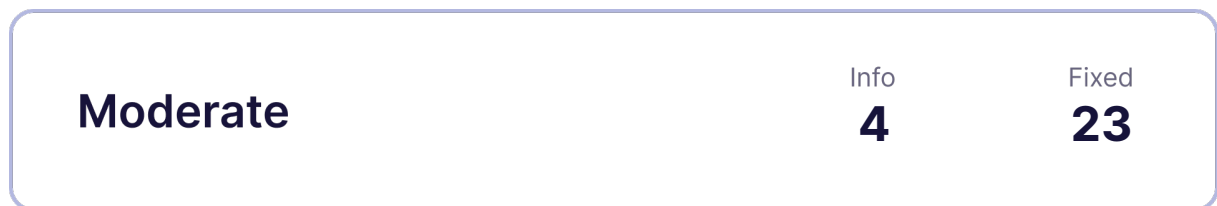
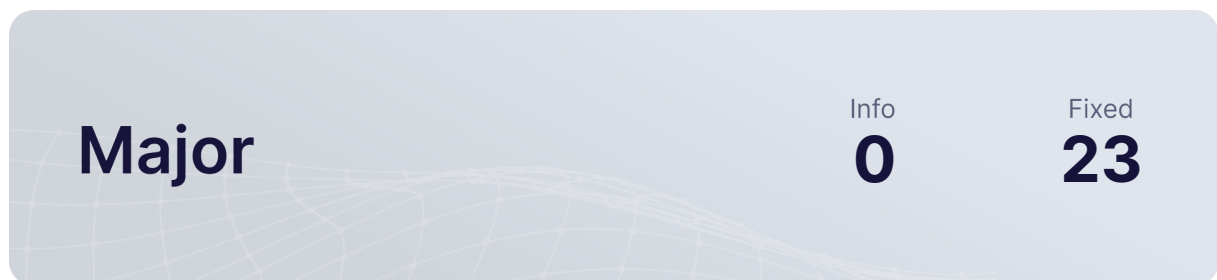
- **General Code Assessment.** The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, 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 12 critical, 47 major, and a few less important issues. All identified Critical issues have been fixed.



Fixed 118 out of 171 issues

6 Critical Issues

CVF-1. FIXED

- **Category** Flaw

- **Source** exit_circuit.rs

Description This code effectively does nothing. Should probably enforce an equality of `is_required_source_token_and_target_token` to true.

Client Comment *Modified to Boolean::enforce_equal.*

```
68 68 Boolean::and(  
69 69     cs.namespace(|| "require correct token"),  
70 70     &is_required_source_token_and_target_token,  
71 71     &Boolean::constant(true),  
72 72 )?;
```

CVF-2. FIXED

- **Category** Flaw

- **Source** withdraw.rs

Description 'b' must be amount+fee for chunk0 and amount for chunk1, whereas here it can be any of them in both chunks. This may lead to fund loss.

Client Comment *The check for chunk0 and chunk1 respectively contains `is_user_b_correct` and `is_global_asset_b_correct`.*

```
89 89 let is_b_correct = {  
90 90     let is_user_b_correct = Boolean::from(Expression::equals(  
91 91         cs.namespace(|| "is_user_b_correct"),  
92 92         op_data.b.get_number(),  
93 93         sum_amount_fee.clone(),  
94 94     )?);  
95 95     let is_global_asset_b_correct = Boolean::from(Expression::equals  
96 96         ↪ (   
97 97         cs.namespace(|| "is_global_asset_b_correct"),  
98 98         op_data.b.get_number(),  
99 99         Expression::from(op_data[WithdrawArgs::FullAmount].  
100 100             ↪ get_number()),  
101 101     )?);  
102 102     multi_or(  
103 103         cs.namespace(|| "is_b_correct_in_chunk0"),  
104 104         &[is_user_b_correct, is_global_asset_b_correct],  
    )?  
};
```

CVF-3. FIXED

- **Category** Flaw
- **Source** withdraw.rs

Description This allows any fourth chunk to pass the function.

```
208 208 boolean_or(  
209 209     cs.namespace(|| "is valid withdraw op"),  
210 210     &is_op_valid,  
211 211     &is_correct_chunk_numbers[3]
```

CVF-4. FIXED

- **Category** Overflow/Underflow
- **Source** order_matching.rs

Description This may overflow if the nonce has been just taken from the updated order, which is not checked for non-overflow.

Recommendation Consider checking all nonces for overflows.

Client Comment *Added nonce overflow check.*

```
529 529 Expression::from(pre_branch.order.nonce.get_number()) + Expression::  
    ↪ u64::<CS>(1),
```

CVF-5. FIXED

- **Category** Flaw
- **Source** deposit.rs

Description The "is_correct_chunk_numbers[3].clone()" allows the "deposit" function to successfully validate a chunk with index 3 even if its TX type is not deposit. So if some TX type (not necessary deposit) has at least four chunks, the fourth chunks will be considered valid regardless of its content.

```
134 134 &[is_chunk0_valid, is_chunk1_valid, is_chunk2_valid,  
    ↪ is_correct_chunk_numbers[3].clone()],
```

CVF-7. FIXED

- **Category** Overflow/Underflow
- **Source** utils.rs

Description Overflow is possible here.

Client Comment *Considering `E::Fr::CAPACITY=253`, I checked the upper limit of parameters a and b , both a and b are less than 2^{126} , so that the subsequent multiplication will not overflow. 2^{126} is about $8 \cdot 10^{37}$, and can accommodate any currency with a total of 10^{18} and a precision of 18. It's enough for most coins.*

```
343 343 let product = a.mul(
```

CVF-8. FIXED

- **Category** Overflow/Underflow
- **Source** utils.rs

Description Overflow is possible here. The quotient variable must be range checked first.

Client Comment *Considering `E::Fr::CAPACITY=253`, I checked the upper limit of parameters a and b , both a and b are less than 2^{126} , so that the subsequent multiplication will not overflow. 2^{126} is about $8 \cdot 10^{37}$, and can accommodate any currency with a total of 10^{18} and a precision of 18. It's enough for most coins.*

```
444 444 let quotient_mul_b = quotient.mul(
```

CVF-9. FIXED

- **Category** Flaw
- **Source** utils.rs

Description This condition is not sound if the middle product overflows.

Recommendation Consider checking that both $a \cdot \text{magnify}$ and $b \cdot (q+1)$ do not overflow.

Client Comment *Considering `E::Fr::CAPACITY=253`, I checked the upper limit of parameters a and b , both a and b are less than 2^{126} , so that the subsequent multiplication will not overflow. 2^{126} is about $8 \cdot 10^{37}$, and can accommodate any currency with a total of 10^{18} and a precision of 18. It's enough for most coins.*

```
465 465 // b*q < a*magnify < b*(q+1)
```

CVF-10. FIXED

- **Category** Flaw
- **Source** utils.rs

Description All multiplications and additions in this function may overflow, and the range checks in the end of the function do not prevent it. For example, if $k=2^{128}-1$ but x and y being small, the $x*y$ may be between k^2 and $(k+1)^2$ as both overflow.

Recommendation Consider using big number arithmetic here.

Client Comment *Considering $E::Fr::CAPACITY=253$, I checked the upper limit of parameters a and b , both a and b are less than 2^{126} , so that the subsequent multiplication will not overflow. 2^{126} is about $8*10^{37}$, and can accommodate any currency with a total of 10^{18} and a precision of 18. It's enough for most coins.*

485 485

```
pub fn sqrt_enforce<E: Engine, CS: ConstraintSystem<E>>(</pre>
```

CVF-11. FIXED

- **Category** Flaw
- **Source** full_exit.rs

Description The variable `is_correct_chunk_numbers[3]` is not checked against anything and thus is true for any 4-th chunk, which makes the entire function to return true.

145 145

```
&[is_chunk0_valid, is_chunk1_valid, is_chunk2_valid,  
  ↪ is_correct_chunk_numbers[3].clone()]</pre>
```

CVF-12. FIXED

- **Category** Flaw
- **Source** forced_exit.rs

Description The `"is_correct_chunk_numbers[3]"` allows the `"forced_exit"` function to successfully validate a chunk with index 3 even if its TX type is not forced exit. So if some TX type (not necessary forced exit) has at least four chunks, the fourth chunks will be considered valid regardless of its content.

238 238

```
&is_correct_chunk_numbers[3]</pre>
```


CVF-13. FIXED

- **Category** Flaw

- **Source** order_matching.rs

Recommendation “ExpectBaseAmount” is added and then subtracted, so it could be removed from the expression. This shouldn’t be so.

```
636 +Expression::from(op_data[OrderMatchingArgs::ExpectQuoteAmount].  
    ↪ get_number())  
637 +   + op_data[OrderMatchingArgs::ExpectBaseAmount].get_number()  
638 +   + op_data[OrderMatchingArgs::ExpectQuoteAmount].get_number()  
639 +   - op_data[OrderMatchingArgs::ExpectBaseAmount].get_number(),
```

7 Major Issues

CVF-6. FIXED

- **Category** Flaw

- **Source** change_pubkey_offchain.rs

Description The witness is generated using `change_pubkey_offchain.account_id` as changer, whereas `op_data` carries `temp_account_id`. If these two variables differ, the proof will fail.

Client Comment *Removed `temp_account_id`, There's really no problem here, and it's redundant code, because in the state handler module, `temp_account_id` is also derived from the `account_id`.*

```
35 35     account_id: *change_pubkey_offchain.account_id,
37 37     temp_account_id: *change_pubkey_offchain.tx.account_id,
148 148 let account_id_fe = Fr::from_u64(change_pubkey_offchain.account_id
    ↪ as u64);
150 150 let temp_account_id_fe = Fr::from_u64(change_pubkey_offchain.
    ↪ temp_account_id as u64);
219 219     before: OperationBranch {
220 220         account_id: Some(account_id_fe),
271 271         frs_with_4_bytes: vec![
273 273             Some(temp_account_id_fe),
317 317         ChangePubkeyArgs::AccountId => &self.ces_with_4_bytes[1],
```

CVF-17. FIXED

- **Category** Bad naming

- **Source** circuit.rs

Description This variable has the same name as an argument.

Recommendation Consider using a different name.

Client Comment *Changed the variable name of the function entry.*

```
458 458 let next_chunk_number = Expression::conditionally_select(
```

CVF-24. FIXED

- **Category** Flaw
- **Source** order_matching.rs

Description Pubdata does not contain MakerIsSell flag, which makes it difficult to restore the operations.

Client Comment *Since pubdata contains MakerSellToken and TakerSellToken, MakerIsSell is not required. I've changed this part of the code a little bit to make it clearer.*

```
22 22 let mut pubdata_bits = Vec::with_capacity(OrderMatchingOp::CHUNKS *  
    ↪ CHUNK_BIT_WIDTH);
```

CVF-27. FIXED

- **Category** Overflow/Underflow
- **Source** order_matching.rs

Description Overflow is possible here

Client Comment *Overflow case, will not be selected. If it's selected, there's no way that overflow can happen here, because ActualBaseAmount is part of residue CircuitElement::conditionally_select_with_number_strict function will limit the result of selection does not exceed 128 - bit (with parameter y bits length is given priority to, The bits length of pre_branch.order.residue is 128), here if overflow happens, it must not comply with the 128bit constraint. Of course, in actual case, Here the ActualBaseAmount is itself part of pre_branch.order.residue, so naturally it won't overflow either. When another op executes this part of the code, although an overflow may occur, the overflow value will not be selected because the judgment criteria are not met.*

```
513 513 Expression::from(pre_branch.order.residue.get_number()) - op_data[  
    ↪ OrderMatchingArgs::ActualBaseAmount].get_number(),
```

CVF-28. FIXED

- **Category** Overflow/Underflow
- **Source** order_matching.rs

Description MakerBuyAmount is not restricted to any number of bits so that operations with it are prone to overflows.

Recommendation Consider making it the same 20-byte size as TakerBuyAmount

Client Comment *There is no overflow. If the value passed in does overflow, then the MakerBuyAmount and TakerBuyAmount in the check_op_data_part_args function are different from the value actually computed in the circuit and the check will fail. The final conditional selection constraint guarantees that the MakerBuyAmount and TakerBuyAmount will not exceed 128bits.*

```
540 540 let actual_amount = AllocatedNum::conditionally_select(  
541 541     cs.namespace(|| "actual_amount"),  
542 542     op_data[OrderMatchingArgs::MakerBuyAmount].get_number(),
```

CVF-29. FIXED

- **Category** Overflow/Underflow
- **Source** order_matching.rs

Description Overflow is possible here.

Client Comment *There's not that much to overflow. Assuming an overflow occurs here, the 128bits constraint here will not be satisfied.*

```
564 564 Expression::from(pre_branch.balance.get_number()) + op_data.b.  
    ↪ get_number(),
```

```
582 582 Expression::from(post_branch.balance.get_number()) + &  
    ↪ exchange_fee,  
583 583 Expression::from(post_branch.balance.get_number()) + &  
    ↪ actual_amount - &exchange_fee,
```

CVF-30. FIXED

- **Category** Overflow/Underflow
- **Source** order_matching.rs

Description Underflow is possible here

Client Comment *exchange_fee is calculated based on percentage actual_amount and Underflow is not possible. Assuming an underflow occurs here, the 128bits constraint here will not be satisfied.*

583 583

```
Expression::from(post_branch.balance.get_number()) + &actual_amount  
↔ - &exchange_fee,
```

CVF-32. FIXED

- **Category** Overflow/Underflow
- **Source** deposit.rs

Description Overflow is possible here.

Client Comment *I don't think this problem exists. We can't allow the total amount of a coin to exceed 2^{128} , and there will be no addition overflow. Assuming an overflow occurs here, the 128bits constraint here will not be satisfied.*

116 116
117 117

```
let updated_balance = Expression::from(cur.balance.get_number())  
+ Expression::from(op_data[DepositArgs::FullAmount].get_number()  
↔ );
```

CVF-33. FIXED

- **Category** Flaw
- **Source** change_pubkey_offchain.rs

Description There is no nonce overflow check here, while the circuit has such check. Thus, it is possible to generate a witness that cannot be proven.

Client Comment *We will do this checking in the state handler module(which is used for rapid transaction execution).*

182 182

```
acc.nonce.add_assign(&Fr::one());
```

CVF-36. FIXED

- **Category** Overflow/Underflow
- **Source** change_pubkey_offchain.rs

Description This operation may overflow.

Client Comment *I don't think this problem exists. We can't allow the total amount of a coin to exceed 2^{128} , and there will be no addition overflow. Assuming an overflow occurs here, the 128bits constraint here will not be satisfied.*

147 147

```
Expression::from(cur.balance.get_number()) + op_data[
    ↪ ChangePubkeyArgs::FeeUnpacked].get_number(),
```

CVF-40. FIXED

- **Category** Suboptimal
- **Source** utils.rs

Recommendation This function calculates: $a - a*n/d \leq b \leq a + a*n/d$ it would be more reasonable to calculate: $a*x/y \leq b \leq a*y/x$

Client Comment *This function is no longer used and has been removed.*

284 284

```
pub fn constraint_two_number_error<E:Engine, CS: ConstraintSystem<E
    ↪ >>{
```

CVF-41. FIXED

- **Category** Overflow/Underflow
- **Source** utils.rs

Description This length does not seem to be sufficient as $|a-b|$ may be BIT_WIDTH long.

Client Comment *This function is no longer used and has been removed.*

309 309

310 310

311 311

312 312

```
CircuitElement::from_number_with_known_length(
    cs.namespace(|| "chosen number as ce"),
    selected_number,
    FR_BIT_WIDTH - 2,
```

CVF-42. FIXED

- **Category** Overflow/Underflow
- **Source** utils.rs

Description This length does not seem to be sufficient as quotient may be BIT_WIDTH long.

```
406 406 CircuitElement::from_number_with_known_length(  
407 407     cs.namespace(|| "three precision quotient"),  
408 408     quotient,  
409 409     FR_BIT_WIDTH - 2  
410 410 )
```

CVF-43. FIXED

- **Category** Overflow/Underflow
- **Source** utils.rs

Description Overflow is possible here.

Client Comment For a and b , the maximum ($MAX_CALCULATION_BIT_WIDTH=126$) limit is done. Therefore, multiplication must not overflow ($126+126 < Fr::capacity=253$). 126bit is sufficient for most cryptocurrencies.

```
425 425 let magnify_a = a.mul(  
426 426     cs.namespace(|| "magnify_a"),  
427 427     &amplification_factor  
428 428 )?;
```

CVF-44. FIXED

- **Category** Flaw

- **Source** utils.rs

Description This length does not seem to be sufficient as product may be BIT_WIDTH long.

Client Comment For a and b , the maximum ($MAX_CALCULATION_BIT_WIDTH=126$) limit is done. Therefore, multiplication must not overflow ($126+126 < Fr::capacity=253$). 126bit is sufficient for most cryptocurrencies.

```
429 429 let magnify_a = CircuitElement::from_number_with_known_length(  
430 430     cs.namespace(|| "magnify_a with bits"),  
431 431     magnify_a,  
432 432     FR_BIT_WIDTH - 2  
433 433 )?;
```

```
452 452 let lower_bound = CircuitElement::from_number_with_known_length(  
453 453     cs.namespace(|| "lower_bound"),  
454 454     quotient_mul_b,  
455 455     FR_BIT_WIDTH - 2  
456 456 )?;
```

```
459 459 let upper_bound = CircuitElement::from_number_with_known_length(  
460 460     cs.namespace(|| "upper_bound"),  
461 461     upper_bound,  
462 462     FR_BIT_WIDTH - 2,  
463 463 )?;
```

CVF-46. FIXED

- **Category** Flaw

- **Source** transfer_to_new.rs

Description There is no nonce overflow check here, while the circuit has such check. Thus, it is possible to generate a witness that cannot be proven.

Client Comment We will do this checking in the state handler module. The Nonce type in the transaction is u32, so it cannot exceed 32bits. Therefore, you only need to check that the Nonce is not equal to $u32::MAX$.

```
277 277 acc.nonce.add_assign(&Fr::one());
```


CVF-47. FIXED

- **Category** Flaw
- **Source** withdraw.rs

Description There is no nonce overflow check here, while the circuit has such check. Thus, it is possible to generate a witness that cannot be proven.

Client Comment *We will do this checking in the state handler module.*

```
289 289 acc.nonce.add_assign(&Fr::one());
```

CVF-48. FIXED

- **Category** Flaw
- **Source** transfer.rs

Description There is no nonce overflow check here, while the circuit has such check. Thus, it is possible to generate a witness that cannot be proven.

Client Comment *We will do this checking in the state handler module.*

```
237 237 acc.nonce.add_assign(&Fr::one());
```

CVF-54. FIXED

- **Category** Procedural
- **Source** allocated_structures.rs

Description The vector lengths are inconsistent with those set in 'operation.rs'.

Recommendation Consider using named constants and define those in a common file.

Client Comment *Added constant.*

```
346 346 ces_with_bool: vec![ce_with_bool;2],
347 347 ces_with_1_byte: vec![ce_with_1_byte; 7],
348 348 ces_with_2_bytes: vec![ce_with_2_bytes.clone(); 7],
349 349 ces_with_4_bytes: vec![ce_with_4_bytes; 15],
350 350 ces_with_8_bytes: vec![ce_with_8_bytes; 4],
351 351 ces_with_15_bytes: vec![ce_with_15_bytes; 2],
352 352 ces_with_16_bytes: vec![ce_with_16_bytes.clone(); 12],
353 353 ces_with_20_bytes: vec![ce_with_20_bytes; 3],
354 354 ces_with_max_bytes: vec![ce_with_max_bytes; 1],
355 355 fee_packed_ces: vec![ce_with_2_bytes; 2],
356 356 fee_unpacked_ces: vec![ce_with_16_bytes.clone(); 2],
357 357 amount_packed_ces: vec![ce_with_5_bytes; 5],
358 358 amount_unpacked_ces: vec![ce_with_16_bytes; 5],
```

CVF-55. FIXED

- **Category** Flaw
- **Source** utils.rs

Description There is no check that the number of operations matches the number of pubdata chunks.

Recommendation Consider adding such a check.

Client Comment *Since NoOp might be populated later, I considered adding a check on the number of OperationUnits and the length of pubdata in the calculate_pubdata_commitment function.*

```
74 74 ops: Vec<OperationUnit<Engine>>,
75 75 pubdata: Vec<bool>,
```

CVF-57. FIXED

- **Category** Unclear behavior
- **Source** forced_exit.rs

Description There are no authorization checks for the initiator account. Does this mean that anybody may initiate a forced exit?

Client Comment *Can only ForcedExit inactive accounts. The initiator can be anyone, considering that some smart contracts charge money to the second layer, but the smart contract has no private key, because it cannot be used as a ChangePubKey, the funds at the second layer cannot be referred to the first layer, so forcedExit is required. Refer to <https://preview-docs.zk.link/docs/developer/terminology/#forcedexit>.*

```
26 26 pubdata_bits.extend(op_data[ForcedExitArgs::InitiatorAccountId].
    ↪ get_bits_be());
27 27 pubdata_bits.extend(op_data[ForcedExitArgs::InitiatorSubAccountId].
    ↪ get_bits_be());
```

CVF-59. FIXED

- **Category** Flaw
- **Source** forced_exit.rs

Description There is no nonce overflow check here, while the circuit has such check. Thus, it is possible to generate a witness that cannot be proven.

Client Comment *We will do this checking in the state handler module.*

```
248 248 acc.nonce.add_assign(&Fr::one());
```

CVF-60. FIXED

- **Category** Flaw
- **Source** element.rs

Description There is no check to ensure that the length doesn't exceed the field capacity.

Recommendation Consider adding such a check.

Client Comment *Added check.*

```
11 11 pub fn unsafe_empty_of_some_length(zero_num: AllocatedNum<E>, length  
    ↪ : usize) -> Self {
```

8 Moderate Issues

CVF-14. INFO

- **Category** Suboptimal
- **Source** exit_circuit.rs

Description Using SHA-256 for hashing public inputs is expensive.

Recommendation Consider using a zk friendly hash as in here <https://docs.google.com/drawings/d/1v5zGTuydDuT2cIF52twJAS71h4kQRuk8dIZLCCzSiaY/edit?usp=sharing>

Client Comment *After that, I'll think about it.*

188 188
189 189

```
let mut hash_block =  
    sha256::sha256(cs.namespace(|| "sha256 of pub data"), &  
        ↪ initial_hash_data)?;
```

333 333

```
let mut h = Sha256::new();
```

CVF-15. INFO

- **Category** Suboptimal
- **Source** circuit.rs

Description SHA-256 calls are expensive in circuits.

Recommendation Consider using an algebraic hash inside the circuit and SHA-256 in the contract as described here <https://docs.google.com/drawings/d/1v5zGTuyd-DuT2cIF52twJAS71h4kQRuk8dIZLCCzSiaY/edit?usp=sharing>

Client Comment *After that, I'll think about it.*

354 354

```
let mut hash_block = sha256::sha256(cs.namespace(|| "hash with pub  
    ↪ data and op offset commitment"), &pack_bits)?;
```

CVF-18. INFO

- **Category** Suboptimal
- **Source** circuit.rs

Description This function behaves differently for different operations and is away of the chunk structure of particular operations. Such approach is very error -prone.

Recommendation Consider moving all operation-specific logic into files named after particular operations, and keeping only operation-agnostic logic here.

Client Comment *Here's the logic for determining the circuits of different blocks based on contains_ops field, which is the binary bits of ops composition number. The ops composition number represents the minimum circuit execution selected based on the available ops composition numbers of the environment configuration and the transactions in the block.*

```
584 584 &[zk_link_ops[WithdrawOp::OP_CODE as usize].clone(),  
↪ is_correct_chunk_numbers[1].clone()],
```

```
590 590 zk_link_ops[OrderMatchingOp::OP_CODE as usize].clone(),  
591 591 zk_link_ops[ForcedExitOp::OP_CODE as usize].clone(),
```

CVF-20. INFO

- **Category** Suboptimal
- **Source** circuit.rs

Description This function behaves differently for different operations and is away of the chunk structure of particular operations. Such approach is very error -prone.

Recommendation Consider moving all operation-specific logic into files named after particular operations, and keeping only operation-agnostic logic here.

Client Comment *This function is redundant. The check for this function has already been done at the corresponding op.*

```
937 937 fn assert_global_assert_account<CS: ConstraintSystem<E>>(
```

CVF-22. INFO

- **Category** Unclear behavior
- **Source** withdraw.rs

Description This extends `pubdata_bits` with the current TX type, which could be different from the withdraw TX type.

Recommendation Consider extending with the correct withdraw TX type.

Client Comment *With `zk_link_ops` in `base_flags`, compare `global_variables.chunk_data.tx_type` with all `tx_type`.*

```
22 22 pubdata_bits.extend(global_variables.chunk_data.tx_type.get_bits_be  
    ↪ ()); //TX_TYPE_BIT_WIDTH=8
```

CVF-23. INFO

- **Category** Unclear behavior
- **Source** withdraw.rs

Description This extends `serialized_tx_bits` with the current TX type, which could be different from the withdraw TX type.

Recommendation Consider extending with the correct withdraw TX type.

Client Comment *With `zk_link_ops` in `base_flags`, compare `global_variables.chunk_data.tx_type` with all `tx_type`.*

```
56 56 serialized_withdraw_bits.extend(global_variables.chunk_data.tx_type.  
    ↪ get_bits_be());
```

CVF-25. INFO

- **Category** Unclear behavior
- **Source** order_matching.rs

Description This extends `pubdata_bits` with the current TX type, which could be different from the order matching TX type.

Recommendation Consider extending with the correct order matching TX type.

Client Comment *With `zk_link_ops` in `base_flags`, compare `global_variables.chunk_data.tx_type` with all `tx_type`.*

```
23 23 pubdata_bits.extend(global_variables.chunk_data.tx_type.get_bits_be  
    ↪ ());
```

CVF-26. INFO

- **Category** Unclear behavior
- **Source** order_matching.rs

Description This extends `serialized_tx_bits` with the current TX type, which could be different from the order matching TX type.

Recommendation Consider extending with the correct order matching TX type.

Client Comment *With `zk_link_ops` in `base_flags`, compare `global_variables.chunk_data.tx_type` with all `tx_type`.*

115 115

```
serialized_tx_bits_version.extend(global_variables.chunk_data.  
    ↪ tx_type.get_bits_be());
```

CVF-37. INFO

- **Category** Procedural
- **Source** utils.rs

Description This file contains both, circuit fragments and normal Rust utility functions.

Recommendation Consider separating these two classes of utilities into two files.

Client Comment *Then we'll consider splitting up.*

1 1

```
// Workspace deps
```

CVF-61. INFO

- **Category** Unclear behavior
- **Source** order_matching.rs

Description It is unclear why this check is not needed anymore.

Client Comment *Removing this check is because the presence of order in the state tree is a waste of a layer of data storage, and I need to upload a lot more pubdata (without uploading it, I won't be able to do layer2 state recovery). For the order information in the state tree, I only store the nonce and order residue. The previous check was just to make sure that orders that were not completely filled, and then sent a second time would not be replaced with something. So after careful consideration, I think the order hash can be deleted.*

```
143 -self.check_order_hash(  
144 -   cs.namespace(||"is_correct_order_hash_when_not_update"),  
145 -   pre_branch,  
146 -   &is_none_residue,  
147 -   &is_refresh_order,  
148 -   &zk_link_ops[OrderMatchingOp::OP_CODE as usize],  
149 -   is_correct_chunk_numbers,  
150 -   &serialized_order_bits_0,  
151 -   &serialized_order_bits_1,  
152 -)?;
```

CVF-62. INFO

- **Category** Unclear behavior
- **Source** LayerZeroBridge.sol

Description The call inside the loop sends the whole contract's balance somewhere, while the last line requires the contract's balance to be at least non-negative. This is possible only when the loop body was executed zero times.

Client Comment *The endpoint of LayerZero needs to set a refund address when sending a message. When the paid eth exceeds the actual requirement, the excess eth will be refunded. Here we use the contract address of LayerZeroBridge as the refund address, so excess eth will be returned to this address in each cycle.*

```
120 +for (uint i = 0; i < dstChainIds.length - 1; ++i) { // overflow is  
    ↪ impossible  
121 +   _bridgeZkLinkBlockProgress(synchHash, progress, dstChainIds[i],  
    ↪ payable(address(this)), zroPaymentAddress, adapterParams,  
    ↪ address(this).balance);  
122 +}  
  
124 +require(address(this).balance > originBalance, "Msg value is not  
    ↪ enough for the last send");
```


CVF-63. INFO

- **Category** Procedural
- **Source** circuit.rs

Recommendation Consider calling this function in each operation-specific call in order to distinguish between the two cases: when we have to check the prev.branch becomes post.branch, and when we do not

381 381

```
fn contains_double_account_modules(&self) -> bool {
```

CVF-64. INFO

- **Category** Overflow/Underflow
- **Source** circuit.rs

Description Underflow is possible here.

Recommendation Consider working with addition instead.

Client Comment *I'm assuming underflow, and it doesn't pass the equal check. There is no token id as large as in the underflow case.*

630 630

```
let real_l1_token = Expression::from(op_data[CommonArgs::L1Token].  
  ↪ get_number()) - Expression::u64::<CS>(USD_X_TOKEN_ID_RANGE as  
  ↪ u64);
```

CVF-65. INFO

- **Category** Suboptimal
- **Source** deposit.rs

Description These variables are not range-checked against the expected bitlengths.

Recommendation Consider asserting.

Client Comment *I don't think there are any overflow issues here, the data is handled by the state handler, and the type conversions are small to large.*

```
161 161 let account_id_fe = Fr::from_u64(deposit.account_id as u64);
162 162 let global_account_id_fe = Fr::from_u64(*GLOBAL_ASSET_ACCOUNT_ID as
    ↪ u64);
163 163 let chain_id_fe = Fr::from_u64(deposit.chain_id as u64);
164 164 let l1_source_token_fe = Fr::from_u64(deposit.l1_source_token as u64
    ↪ );
165 165 let l2_target_token_fe = Fr::from_u64(deposit.l2_target_token as u64
    ↪ );
166 166 let l1_source_token_after_mapping = Fr::from_u64(deposit.
    ↪ l1_source_token_after_mapping as u64);
167 167 let amount_as_field_element = Fr::from_big_uint(deposit.amount.into
    ↪ ());
```

CVF-66. INFO

- **Category** Overflow/Underflow
- **Source** order_matching.rs

Description Overflow here may cause false positive.

Client Comment *There's not that much to overflow.*

```
670 670 Expression::from(op_data[OrderMatchingArgs::ExpectQuoteAmount].
    ↪ get_number()) +
671 671 op_data[OrderMatchingArgs::ExpectBaseAmount].get_number(),
```

CVF-67. INFO

- **Category** Documentation
- **Source** deposit.rs

Description The same assignment was done for witness in chunk2 but not verified here.

Recommendation Consider explaining the inconsistency in the comment.

Client Comment *This is explained in the document <https://preview-docs.zk.link/docs/developer/terminology/#global-assets-account>, I will be right here to add a comment.*

```
97 97 chunk1_valid_flags.push(CircuitElement::equals(  
98 98     cs.namespace(|| "DepositArgs::ChainId == cur.sub_account_id in  
    ↪ chunk1"),  
99 99     &op_data[DepositArgs::ChainId],  
100 100     &cur.sub_account_id,  
101 101     )?);
```

CVF-68. INFO

- **Category** Overflow/Underflow
- **Source** change_pubkey_offchain.rs

Description These variables are not range-checked against the expected bitlengths.

Recommendation Consider asserting.

Client Comment *These checks are done in the state handler.*

```
148 148 let account_id_fe = Fr::from_u64(change_pubkey_offcahin.account_id  
    ↪ as u64);  
149 149 let sub_account_id_fe = Fr::from_u64(change_pubkey_offcahin.  
    ↪ sub_account_id as u64);  
150 150 let temp_account_id_fe = Fr::from_u64(change_pubkey_offcahin.  
    ↪ temp_account_id as u64);  
151 151 let validator_account_id_fe = Fr::from_u64(*FEE_ACCOUNT_ID as u64);  
  
153 153 let fee_token_fe = Fr::from_u64(change_pubkey_offcahin.fee_token as  
    ↪ u64);
```

CVF-69. INFO

- **Category** Unclear behavior
- **Source** utils.rs

Description This bit is always false when FR is 255 bits or shorter.

Client Comment *This is not always false, this bit is equivalent to the parity of the number.*

```
106 106 sig_r_packed_bits.push(signature_r_x_be_bits[FR_BIT_WIDTH_PADDED -  
↔ 1]);
```

CVF-70. INFO

- **Category** Suboptimal
- **Source** utils.rs

Description Conversion to a floating point number may lose precision, thus the unpacked number m , may differ from the original one.

Recommendation Consider replacing this strict check with a range check.

Client Comment *We check if the value is packable as soon as the transaction enters layer2. Non-packable transactions will be returned.*

```
584 584 let is_correct_a = CircuitElement::equals(  
585 585     cs.namespace(|| "a != a_unpacked"),  
586 586     a,  
587 587     &a_unpacked,  
588 588 )?;
```

CVF-71. FIXED

- **Category** Documentation
- **Source** signature.rs

Recommendation Consider making this assumption explicit in the function documentation.

Client Comment *Adopted.*

```
370 370 // only order of R is checked. Public key and generator can be  
↔ guaranteed to be in proper group!
```

CVF-72. FIXED

- **Category** Unclear behavior
- **Source** transfer.rs

Description The same data is added twice.

Client Comment *This function has been deprecated.*

```
157 157 append_be_fixed_width(  
158 158     &mut sig_bits,  
159 159     &self.before.witness.account_witness.pub_key_hash.unwrap(),  
160 160     NEW_PUBKEY_HASH_WIDTH,  
161 161 );  
162 162 append_be_fixed_width(  
163 163     &mut sig_bits,  
164 164     &self.before.witness.account_witness.pub_key_hash.unwrap(),  
165 165     NEW_PUBKEY_HASH_WIDTH,  
166 166 );
```

CVF-73. INFO

- **Category** Unclear behavior
- **Source** transfer_to_new.rs

Description The signed data format for a transfer to new transaction differs from the signed data format for a transfer transaction. This means that the sender needs to choose between these two transaction types, rather than the operator. If two users sign two transfer to new transactions to the same new address, only one of these transaction could be successfully executed.

Recommendation Consider using the same signed data format for both transactions.

Client Comment *Although the transaction construction codes of Transfer and TransferToNew look different, in fact, every field and length are identical and one-to-one corresponding. In the actual construction, tx_type of TransferToNew will also be used as tx_code of Transfer. This ensures that the Transfer transaction format is unique; This is because TransferToNew involves creating a new account and pubdata involves linking to NewAddress.*

```
57 57 serialized_tx_bits.extend(tx_code.get_bits_be());
58 58 serialized_tx_bits.extend(cur.account_id.get_bits_be());
59 59 serialized_tx_bits.extend(op_data[TransferToNewArgs::
    ↪ FromSubAccountId].get_bits_be());
60 60 serialized_tx_bits.extend(op_data[TransferToNewArgs::NewAddress].
    ↪ get_bits_be());
61 61 serialized_tx_bits.extend(op_data[TransferToNewArgs::ToSubAccountId
    ↪ ].get_bits_be());
62 62 serialized_tx_bits.extend(cur.token.get_bits_be());
63 63 serialized_tx_bits.extend(op_data[TransferToNewArgs::AmountPacked].
    ↪ get_bits_be());
64 64 serialized_tx_bits.extend(op_data[TransferToNewArgs::FeePacked].
    ↪ get_bits_be());
65 65 serialized_tx_bits.extend(cur.account.nonce.get_bits_be());
66 66 serialized_tx_bits.extend(op_data[TransferToNewArgs::Timestamp].
    ↪ get_bits_be());
```

CVF-74. FIXED

- **Category** Flaw
- **Source** full_exit.rs

Description The full exit transaction doesn't update nonce, but still performs a nonce overflow check. This makes it impossible to withdraw funds from an account with maxed nonce.

Client Comment *Removed.*

```
56 56 chunk0_valid_flags.push(no_nonce_overflow(  
57 57     cs.namespace(|| "no nonce overflow"),  
58 58     cur.account.nonce.get_number(),  
59 59 )?);
```

CVF-75. FIXED

- **Category** Procedural
- **Source** full_exit.rs

Recommendation This check could have been done in the 'a>b' check for which the target balance should be 'a' and amount should be 'b'.

Client Comment *Adopted.*

```
82 82 let is_balance_lt_surplus = CircuitElement::less_than_fixed(  
83 83     cs.namespace(|| "is balance less than surplus" ),  
84 84     &user_balance,  
85 85     &op_data[FullExitArgs::TargetChainSurplus],  
86 86 )?;
```

CVF-76. INFO

- **Category** Overflow/Underflow
- **Source** forced_exit.rs

Description These variables are not range-checked against the expected bitlengths.

Recommendation Consider asserting.

Client Comment *There are no overflow issues, and data out of bounds is checked by the state handler module and type serialization.*

```
204 204 let account_address_initiator_fe = Fr::from_u64(forced_exit.  
    ↪ initiator_account_id as u64);  
205 205 let account_address_target_fe = Fr::from_u64(forced_exit.  
    ↪ target_account_id as u64);  
206 206 let l2_source_token_fe = Fr::from_u64(forced_exit.l2_source_token as  
    ↪ u64);  
207 207 let l1_target_token_fe = Fr::from_u64(forced_exit.l1_target_token as  
    ↪ u64);  
208 208 let l1_target_token_after_mapping = Fr::from_u64(forced_exit.  
    ↪ l1_target_token_after_mapping as u64);  
209 209 let fee_token_fe = Fr::from_u64(forced_exit.fee_token as u64);  
210 210 let amount_as_field_element = Fr::from_big_uint(forced_exit.amount.  
    ↪ into());  
211 211 let target_sub_account_id = Fr::from_u64(forced_exit.  
    ↪ target_sub_account_id as u64);  
212 212 let initiator_sub_account_id = Fr::from_u64(forced_exit.  
    ↪ initiator_sub_account_id as u64);  
213 213 let chain_id = Fr::from_u64(forced_exit.chain_id as u64);
```

CVF-77. INFO

- **Category** Unclear behavior
- **Source** element.rs

Recommendation Should be "CAPACITY" instead of "NUM_BITS".

Client Comment *This should be NUM_BITS, and we should allow all possible values of SCALAR filed to be accepted.*

```
82 82 assert!(witness_bits.len() <= E::Fr::NUM_BITS as usize);
```


CVF-78. INFO

- **Category** Suboptimal

- **Source** full_exit.rs

Description This should be done only when "is_success" is true.

Client Comment *When is_success is false, exit_amount is None, and eventually unwrap_or_default is called, exit_amount=0, so I got rid of is_success.*

221 221

```
|bal| bal.value.sub_assign(&full_exit.exit_amount),
```

9 Minor Issues

CVF-79. FIXED

- **Category** Procedural
- **Source** exit_circuit.rs

Description The way how a zero element is obtained is different from circuit.rs.

Recommendation Consider using the same approach in both circuits.

Client Comment *Adopted.*

26 26

```
let zero = AllocatedNum::zero(cs.namespace(|| "zero"))?;
```

CVF-80. INFO

- **Category** Procedural
- **Source** exit_circuit.rs

Description Multiplication after division could lead to precision degradation.

Recommendation Consider multiplying before division.

Client Comment *The 18 precision used here is high enough.*

157 157

```
let withdraw_ratio = div_enforce(
```

163 163

```
let amount = multiply_enforce(
```

CVF-81. FIXED

- **Category** Procedural
- **Source** exit_circuit.rs

Description This constant is field specific.

Recommendation Consider naming it and putting into a common file.

Client Comment *Added constant BN256_MASK.*

339 339

```
hash_result[0] &= 0x1f; // temporary solution, this nullifies top  
    ↪ bits to be encoded into field element correctly
```

CVF-82. FIXED

- **Category** Bad datatype
- **Source** circuit.rs

Recommendation These numbers should be named constants.

Client Comment *Adopted.*

```
144 144 data[DepositOp::OP_CODE as usize] = vec![zero.clone(); 2];
145 145 data[TransferToNewOp::OP_CODE as usize] = vec![zero.clone(); 2];
146 146 data[WithdrawOp::OP_CODE as usize] = vec![zero.clone(); 2];
147 147 data[TransferOp::OP_CODE as usize] = vec![zero.clone(); 2];
148 148 data[FullExitOp::OP_CODE as usize] = vec![zero.clone(); 2];
149 149 data[ChangePubKeyOp::OP_CODE as usize] = vec![zero.clone(); 2];
150 150 data[ForcedExitOp::OP_CODE as usize] = vec![zero.clone(); 2];
151 151 data[OrderMatchingOp::OP_CODE as usize] = vec![zero.clone(); 3];
```

CVF-83. INFO

- **Category** Unclear behavior
- **Source** circuit.rs

Description This doesn't guarantee that all the holders are actually allocated. Removing any of the lines 143..151 wouldnt break this check.

Client Comment *This is just a basic op quantity check.*

```
155 155 assert_eq!(pubdata_holder.len(),
↪ ALL_DIFFERENT_TRANSACTIONS_TYPE_NUMBER);
```

CVF-84. INFO

- **Category** Bad datatype
- **Source** circuit.rs

Recommendation '7' should be a named constant.

Client Comment *It's weird to use a constant for 7 here, but our goal is to only set the first byte to 1.*

```
211 211 block_onchain_op_offset_bits.extend(vec![Boolean::constant(false);
↪ 7]);
```

CVF-85. INFO

- **Category** Suboptimal
- **Source** circuit.rs

Description Using 8 bits for a flag looks redundant.

Recommendation Consider using 1 bit per flag.

Client Comment *It's easier to use bytes in the contract.*

211 211

```
block_onchain_op_offset_bits.extend(vec![Boolean::constant(false);  
↪ 7]);
```

CVF-86. FIXED

- **Category** Bad naming
- **Source** circuit.rs

Description The variable name "pre_state_root" is very similar to the name of another variable: "prev_state_root".

Recommendation Consider using more distinct names.

Client Comment *Modified.*

261 261

```
let (pre_state_root, _, _) = check_account_data(
```

CVF-87. INFO

- **Category** Bad naming
- **Source** circuit.rs

Recommendation The name is confusing, as its value is actually the current chunk.

Client Comment *This is indeed the number used to describe the next chunk.*

408 408

```
next_chunk_number: &AllocatedNum<E>,
```

CVF-88. INFO

- **Category** Procedural

- **Source** circuit.rs

Recommendation These functions can be precomputed for all operations

Client Comment *This function is defined by generics and cannot be precomputed.*

```
417 417 let max_chunks_powers = generate_powers(  
418 418     cs.namespace(|| "generate powers of max chunks"),  
419 419     tx_type.get_number(),  
420 420     ALL_DIFFERENT_TRANSACTIONS_TYPE_NUMBER,  
421 421 )?;  
422 422  
423 423 let max_chunks_last_coeffs = generate_maxchunk_polynomial::<E>();
```

CVF-89. INFO

- **Category** Suboptimal

- **Source** circuit.rs

Recommendation Consider refactoring the code so that the equality of (a,b) is not needed across op_data .

Client Comment *Not sure what the problem is here.*

```
582 582 let withdraw_second = multi_and(  
583 583     cs.namespace(|| "withdraw_second"),  
584 584     &[zk_link_ops[WithdrawOp::OP_CODE as usize].clone(),  
585 585         ↪ is_correct_chunk_numbers[1].clone()],  
585 585 )?;
```

```
587 587 let skip_check_a_and_b = multi_or(  
588 588     cs.namespace(|| "skip_check_a_and_b"),  
589 589     &[  
590 590         zk_link_ops[OrderMatchingOp::OP_CODE as usize].clone(),  
591 591         zk_link_ops[ForcedExitOp::OP_CODE as usize].clone(),  
592 592         withdraw_second,  
593 593     ],
```

CVF-90. FIXED

- **Category** Suboptimal
- **Source** circuit.rs

Description Here a constant is converted to a circuit element at run time.

Recommendation Consider doing at compile time.

Client Comment *Adopted.*

```
653 653 let usdx_tokene_id_upper_bound = CircuitElement::  
      ↪ from_fe_with_known_length(
```

CVF-91. FIXED

- **Category** Bad datatype
- **Source** circuit.rs

Recommendation This should be a named constant.

Client Comment *Adopted.*

```
656 656 8
```

CVF-92. FIXED

- **Category** Procedural

- **Source** circuit.rs

Recommendation The signature verification logic should be moved moved into a separate function.

Client Comment *Adopted.*

```
736 736 let public_generator = self
737 737     .jubjub_params
738 738     .generator(FixedGenerators::SpendingKeyGenerator);

740 740 let generator = ecc::EdwardsPoint::witness(
741 741     cs.namespace(|| "allocate public generator"),
742 742     Some(public_generator.clone()),
743 743     self.jubjub_params,
744 744 )?;
745 745 let (public_generator_x, public_generator_y) = public_generator.
    ↪ into_xy();
746 746 generator.get_x().assert_number(
747 747     cs.namespace(|| "assert generator x is constant"),
748 748     &public_generator_x,
749 749 )?;
750 750 generator.get_y().assert_number(
751 751     cs.namespace(|| "assert generator y is constant"),
752 752     &public_generator_y,
753 753 )?;
754 754 let signer_key = unpack_point_if_possible(
755 755     cs.namespace(|| "unpack pubkey"),
756 756     &op.signer_pub_key_packed,
757 757     self.rescue_params,
758 758     self.jubjub_params,
759 759 )?;
760 760 let signature_data = verify_circuit_signature(
761 761     cs.namespace(|| "verify circuit signature"),
762 762     &op_data,
763 763     &signer_key,
764 764     &op.signature_data,
765 765     self.rescue_params,
766 766     self.jubjub_params,
767 767     generator,
768 768 )?;
769 769 (Some(signer_key), Some(signature_data))
```

CVF-93. INFO

- **Category** Bad datatype
- **Source** circuit.rs

Recommendation This should be a named constant

Client Comment *This type is determined by generics and constant cannot be created.*

947 947 `&AllocatedNum::one::<CS>()`

CVF-94. FIXED

- **Category** Suboptimal
- **Source** circuit.rs

Recommendation This can be replaced by a simple check of three variables being all equal to 0.

Client Comment *Adopted.*

1044 1044 `let is_account_empty = {`

CVF-95. FIXED

- **Category** Procedural
- **Source** circuit.rs

Recommendation This comment should be resolved or removed.

Client Comment *Removed.*

1083 1083 `// TODO: Add AllocatedNum to leaf.`

CVF-96. INFO

- **Category** Suboptimal
- **Source** circuit.rs

Recommendation This function could be simplified. Just construct a series of polynomials P_1, P_2, \dots, P_n such that $P_i(i) = 1$, and $P_i(j) = 0$ for j in $\{1, 2, \dots, i - 1, i + 1, \dots, n\}$. Then calculate the result as: $P_1(c) c_1 + P_2(c) c_2 + \dots + P_n(c) c_n$. Here c is the chunk number to select, and c_1, c_2, \dots, c_n are pubdata chunks.

Client Comment *The Recommendation approach does not save much constraint because:*

1. *The calculation of each polynomial requires a linear combination constraint*
2. *Calculate x^1, x^2, \dots, x^3 powers also require n constant constraints*
3. *$P_1(c) c_1 + P_2(c) c_2 + \dots$ Plus $P_n(c) c_n$ requires n multiplicative constraints plus a linear combination*

So it doesn't feel like it's reducing constraints

1173 1173

```
pub fn select_pubdata_chunk<E: JubjubEngine, CS: ConstraintSystem<E>
    ↪ >>{
```

CVF-97. FIXED

- **Category** Suboptimal
- **Source** circuit.rs

Recommendation This function could be simplified as: $\text{multi_or}(x_1, x_2, \dots, x_n) = (x_1 + x_2 + \dots + x_n) \neq 0$

Client Comment *Rewritten, moved to utils.rs line 119.*

1207 1207

```
pub fn multi_or<E: JubjubEngine, CS: ConstraintSystem<E>>{
```

CVF-98. FIXED

- **Category** Procedural
- **Source** circuit.rs

Recommendation This low-level utility function should be moved to some other file.

Client Comment *Moved to utils.rs line 119.*

1207 1207

```
pub fn multi_or<E: JubjubEngine, CS: ConstraintSystem<E>>{
```

CVF-99. INFO

- **Category** Suboptimal
- **Source** circuit.rs

Recommendation These values could be precalculated.

Client Comment *This function is defined by generics and cannot be precomputed.*

1232 1232

```
let empty_node_hashes = calculate_empty_account_tree_hashes::(  
    ↪ params, tree_depth);
```

CVF-100. INFO

- **Category** Suboptimal
- **Source** circuit.rs

Recommendation This function can be precomputed.

Client Comment *This function is defined by generics and cannot be precomputed.*

1258 1258

```
fn generate_maxchunk_polynomial<E: JubjubEngine>() -> Vec<E::Fr> {
```

CVF-101. INFO

- **Category** Suboptimal
- **Source** deposit.rs

Description Data loss is possible when converting types here.

Recommendation Consider using checked conversion.

Client Comment *There is no data loss here because both data are converted from a small type to a large type.*

33 33

```
l2_target_token: *deposit.tx.l2_target_token as u32,
```

CVF-102. FIXED

- **Category** Suboptimal
- **Source** deposit.rs

Description Here a variable name carries the data bitlength, which is a compile-time constant. If the constant changes then the variable name should change.

Recommendation Consider putting the expected bitlength into an immutable variable of the struct so that it can be matched with the provided bitlength.

```
47 47 &self.args.frs_with_1_byte[0].unwrap(),
```

```
62 62 &self.args.frs_with_2_bytes[1].unwrap(),
```

```
67 67 &self.args.frs_with_2_bytes[0].unwrap(),
```

```
72 72 &self.args.frs_with_16_bytes[0].unwrap(),
```

```
78 78 &self.args.frs_with_20_bytes[0].unwrap(),
```

CVF-103. FIXED

- **Category** Procedural
- **Source** deposit.rs

Recommendation This constant should be named.

Client Comment *Adopted.*

```
90 90 let mut commitment = vec![false; DepositOp::CHUNKS * 8];
```

CVF-104. FIXED

- **Category** Bad naming
- **Source** deposit.rs

Recommendation Constant 0 should be named.

```
195 195 get_audits(tree, *GLOBAL_ASSET_ACCOUNT_ID, deposit.chain_id, deposit  
↔ .l1_source_token_after_mapping, 0);
```

```
204 204 (deposit.chain_id, deposit.l1_source_token_after_mapping, 0),
```

CVF-105. FIXED

- **Category** Procedural
- **Source** deposit.rs

Recommendation Consider removing this data.

Client Comment *Removed.*

```
272 272 frs_with_8_bytes: vec![ // need or not
273 273     Some(Fr::zero()), Some(Fr::from_u64(u32::MAX as u64))
274 274 ].into(),
```

CVF-106. INFO

- **Category** Suboptimal
- **Source** withdraw.rs

Description Outputting not an actual nonce seems odd.

Recommendation Consider always outputting the signed nonce

Client Comment *Considering.*

```
37 37 pubdata_bits.extend(nonce.get_bits_be()); // NONCE_BIT = 32
```

CVF-107. FIXED

- **Category** Documentation
- **Source** withdraw.rs

Recommendation This comment looks incorrect

```
66 66 serialized_withdraw_bits.extend(op_data[WithdrawArgs::IsFastWithdraw
↔ ].clone().into_padded_be_bits(8)); //ETH_ADDRESS=160
```

CVF-108. FIXED

- **Category** Bad datatype
- **Source** order_matching.rs

Recommendation This should be a named constant.

Client Comment *Added constant ORDERS_BIT_WIDTH.*

```
106 106 orders_bits.resize(1424, Boolean::constant(false));
```

CVF-109. FIXED

- **Category** Readability
- **Source** order_matching.rs

Recommendation nonece → nonce

```
183 183 let select_order_nonece = CircuitElement::conditionally_select(
```

CVF-110. FIXED

- **Category** Bad naming
- **Source** order_matching.rs

Recommendation The variable name is misleading as it covers chunks zero and one, rather than just zero.

```
261 261 let is_sub_account_correct_in_chunk_0 = multi_and(
```

CVF-111. INFO

- **Category** Suboptimal
- **Source** order_matching.rs

Recommendation These two flags could be merged into one that is calculated for the chunks 0, 1, and 2.

Client Comment *The feeling here is that chunk2 cannot be merged into chunk 1 or 2, and the situation is different.*

```
261 261 let is_sub_account_correct_in_chunk_0 = multi_and(
```

```
265 265 let is_sub_account_correct_in_chunk_2 = multi_and(
```

CVF-112. FIXED

- **Category** Unclear behavior
- **Source** order_matching.rs

Description Should it be 'chunk0, 3'?

```
307 307 cs.namespace(|| "select post token in chunk0-2"),
```

CVF-113. FIXED

- **Category** Unclear behavior
- **Source** order_matching.rs

Description This variable is always true as two flags may never be equal due to 'matching_trading_relationship' flag enforcement.

Client Comment *Fixed MakerIsSell to MakerSlotId. Fixed TakerIsSell to TakerSlotId.*

```
337 337 let is_different_slot = CircuitElement::equals(  
338 338     cs.namespace(|| "is different slot"),  
339 339     &op_data[OrderMatchingArgs::MakerIsSell],  
340 340     &op_data[OrderMatchingArgs::TakerIsSell],  
341 341 )?.not();
```

CVF-114. FIXED

- **Category** Suboptimal
- **Source** order_matching.rs

Description This is always equal to is_self_swap (see above)

```
342 342 let is_self_swap_and_different_slot = Boolean::and(  
343 343     is_self_swap,  
344 344     is_different_slot,  
345 345 )
```

CVF-115. FIXED

- **Category** Suboptimal
- **Source** order_matching.rs

Description This is always true (see above)

```
347 347 boolean_or(  
348 348     cs.namespace(|| "if is_self_swap {is_different_slot}"),  
349 349     &is_self_swap_and_different_slot,  
350 350     &is_self_swap.not()  
351 351 )?
```

CVF-116. FIXED

- **Category** Suboptimal
- **Source** order_matching.rs

Description There is no need to check this in every chunk.

Recommendation Consider checking only in one chunk.

Client Comment *Adopted.*

```
395 395 base_flags.push(is_price_ok.clone());
```

CVF-117. INFO

- **Category** Suboptimal
- **Source** deposit.rs

Recommendation Constant `DepositOp::CHUNKS` should be used here.

Client Comment *Considering.*

```
134 134 &[is_chunk0_valid, is_chunk1_valid, is_chunk2_valid,  
↪ is_correct_chunk_numbers[3].clone()],
```

CVF-118. FIXED

- **Category** Documentation
- **Source** change_pubkey_offchain.rs

Description The role of this field is unclear.

Recommendation Consider documenting.

Client Comment *Removed.*

```
7 7 pub temp_account_id: u32,
```

CVF-119. INFO

- **Category** Suboptimal
- **Source** change_pubkey_offchain.rs

Description Data loss is possible when converting types here.

Recommendation Consider using checked conversion.

Client Comment *There is no data loss here because both data are converted from a small type to a large type.*

40 40

```
fee_token: *change_pubkey_offchain.tx.fee_token as u32,
```

42 42

```
nonce: Fr::from_u64(*change_pubkey_offchain.tx.nonce as u64),
```

CVF-120. FIXED

- **Category** Bad datatype
- **Source** change_pubkey_offchain.rs

Recommendation The indices should be named constants, or there should be a diagram in the code explaining why these elements are selected.

54 54

```
&self.args.frs_with_1_byte[0].unwrap(),
```

59 59

```
&self.args.frs_with_4_bytes[1].unwrap(),
```

64 64

```
&self.args.frs_with_1_byte[1].unwrap(),
```

69 69

```
&self.args.frs_with_20_bytes[0].unwrap(),
```

74 74

```
&self.args.frs_with_20_bytes[1].unwrap(),
```

79 79

```
&self.args.frs_with_4_bytes[2].unwrap(),
```

89 89

```
&self.args.fees_packed[0].unwrap(),
```

102 102

```
let mut commitment = vec![false; ChangePubKeyOp::CHUNKS * 8];
```

103 103

```
commitment[7] = true;
```


CVF-121. FIXED

- **Category** Suboptimal
- **Source** change_pubkey_offchain.rs

Description Here a variable name carries the data bitlength, which is a compile-time constant. If the constant changes then the variable name should change.

Recommendation Consider putting the expected bitlength into an immutable variable of the struct so that it can be matched with the provided bitlength

```
54 54 &self.args.frs_with_1_byte[0].unwrap(),  
55 55 CHAIN_ID_BIT_WIDTH
```

```
59 59 &self.args.frs_with_4_bytes[1].unwrap(),  
60 60 ACCOUNT_ID_BIT_WIDTH,
```

```
64 64 &self.args.frs_with_1_byte[1].unwrap(),  
65 65 SUB_ACCOUNT_ID_BIT_WIDTH,
```

```
69 69 &self.args.frs_with_20_bytes[0].unwrap(),  
70 70 NEW_PUBKEY_HASH_WIDTH,
```

```
74 74 &self.args.frs_with_20_bytes[1].unwrap(),  
75 75 ETH_ADDRESS_BIT_WIDTH,
```

```
79 79 &self.args.frs_with_4_bytes[2].unwrap(),  
80 80 NONCE_BIT_WIDTH,
```

CVF-122. INFO

- **Category** Readability
- **Source** change_pubkey_offchain.rs

Recommendation Should be 'change_pubkey_offchain'.

Client Comment Here it feels unnecessary, ChangePubkeyOffChainData is ChangePubkeyOffChainOp to ChangePubkeyOffChainDataWitness an intermediate product.

```
143 143 change_pubkey_offcahin: ChangePubkeyOffChainData,
```

CVF-123. FIXED

- **Category** Suboptimal
- **Source** change_pubkey_offchain.rs

Recommendation This flag is redundant. Just do: `cur.balance = conditionally_select (balance - fee, conditionally_select (balance + fee, cur.balance, is_chunk1), is_chunk_0);`

Client Comment *Adopted.*

```
151 151 let is_valid_first_or_second = boolean_or(
```

CVF-124. FIXED

- **Category** Procedural
- **Source** utils.rs

Recommendation This issue should be removed or resolved

Client Comment *Removed.*

```
68 68 // TODO: handle the case where it is not valid (ZKS-101)
69 69 // if !is_valid_signature {
70 70 //     return None;
71 71 // }
```

CVF-125. FIXED

- **Category** Suboptimal
- **Source** utils.rs

Description Reversing twice is suboptimal.

Recommendation Consider refactoring

Client Comment *Optimized.*

```
90 90 signature_r_x_be_bits.reverse();
```

```
96 96 signature_r_x_be_bits.reverse();
```

```
98 98 signature_r_y_be_bits.reverse();
```

```
104 104 signature_r_y_be_bits.reverse();
```

CVF-126. FIXED

- **Category** Suboptimal
- **Source** utils.rs

Description This code largely duplicates that of 'sign_sha256' function.

Recommendation Consider refactoring.

Client Comment *Removed.*

124 124

```
pub fn sign_sha<E>(
```

CVF-127. FIXED

- **Category** Suboptimal
- **Source** utils.rs

Recommendation This function could be simplified as: `multi_and (x1, x2, ..., xn) = (x1 + x2 + ... + xn == n)`

Client Comment *Rewritten.*

168 168

```
pub fn multi_and<E: Engine, CS: ConstraintSystem<E>>(
```

CVF-128. INFO

- **Category** Overflow/Underflow
- **Source** utils.rs

Description Overflow is possible here.

Recommendation Consider adding an explicit overflow check.

Client Comment *If this overflows, the unwrap will panic.*

258 258

```
E::Fr::from_str(&product.to_string()).unwrap()
```

CVF-129. INFO

- **Category** Overflow/Underflow
- **Source** utils.rs

Description Overflow is possible here.

Recommendation Consider adding an explicit overflow check.

Client Comment *If this overflows, the unwrap will panic.*

272 272

```
E::Fr::from_str(&quotient.to_string()).unwrap()
```



CVF-130. INFO

- **Category** Unclear behavior
- **Source** utils.rs

Description 'b' plays no role in the computation. Is it ok?

Client Comment *b is going to be involved, but b and a are both amplified values.*

```
317 317 let product = a.get_number().mul(
```

CVF-131. INFO

- **Category** Documentation
- **Source** utils.rs

Recommendation Consider documenting that the caller must ensure that precision is not too high

Client Comment *So far, we have layer2 with an accuracy of 18.*

```
417 417 precision:u64
```

CVF-132. INFO

- **Category** Suboptimal
- **Source** transfer_to_new.rs

Description Data loss is possible when converting types here.

Recommendation Consider using checked conversion.

Client Comment *Token is u16, there is no loss when converted to u32.*

```
36 36 token: *transfer_to_new.tx.token as u32,
```

CVF-133. FIXED

- **Category** Suboptimal
- **Source** transfer_to_new.rs

Recommendation This constant must be named.

Client Comment *Removed this function.*

```
182 182 &Fr::from_u64(5u64), //Corresponding tx_type
```

CVF-134. INFO

- **Category** Readability
- **Source** transfer_to_new.rs

Recommendation Subtracting the sum would be more readable.

Client Comment *The values here are all values that have been processed by the state handler. If you subtract a negative number, the state handler will return an insufficient balance error.*

```
280 280 bal.value.sub_assign(&amount_as_field_element);  
281 281 bal.value.sub_assign(&fee_as_field_element);
```

CVF-135. INFO

- **Category** Suboptimal
- **Source** transfer_to_new.rs

Recommendation Should be u32

Client Comment *This is just because from_u64 requires a u64 parameter.*

```
401 401 Some(Fr::from_u64(transfer_to_new.ts as u64)),
```

CVF-136. INFO

- **Category** Unclear behavior
- **Source** signature.rs

Description This bit is packed_key[248]. Why is called r_x_bit?

Client Comment *This bit is packed_key[256].*

```
41 41 let r_x_bit =  
42 42     AllocatedBit::alloc(cs.namespace(|| "r_x_bit"),  
    ↪ packed_key_bits_correct_order[0])?;
```

CVF-137. INFO

- **Category** Unclear behavior
- **Source** signature.rs

Description This drops elements 248 and 249 of the original array. Is this okay?

Client Comment *The elements 248 and 249 of the original array are not dropped. E::Fr::NUM_BITS=253(bn256).*

```
46 46 let r_y = CircuitElement::from_witness_be_bits(  
47 47     cs.namespace(|| "signature_r_y from bits"),  
48 48     &packed_key_bits_correct_order[start_of_y..],  
49 49 )?;
```

CVF-138. FIXED

- **Category** Readability
- **Source** signature.rs

Recommendation Consider renaming.

Client Comment *Modified.*

```
290 290 let hash_input = multipack::pack_into_witness(  
    
```

CVF-139. INFO

- **Category** Suboptimal
- **Source** signature.rs

Recommendation Truncating two output elements to halves does not make sense. It suffices to just take one element and use its bits.

Client Comment *In the essence we perform modular reduction, so to ensure uniformity we only take half of the bits, so non-uniformity is around $1 / (\text{char} / (E::Fs::CAPACITY / 2))$ that is around $1/2^{126}$.*

```
313 313 let s0 = sponge.squeeze_out_single(
314 314     cs.namespace(|| "squeeze first word form sponge"),
315 315     &rescue_params,
316 316 )?;
317 317
318 318 let s1 = sponge.squeeze_out_single(
319 319     cs.namespace(|| "squeeze second word form sponge"),
320 320     &rescue_params,
321 321 )?;
322 322
323 323 let s0_bits =
324 324     s0.into_bits_le_strict(cs.namespace(|| "make bits of first word
325 325         ↪ for FS challenge"))?;
326 326 let s1_bits =
327 327     s1.into_bits_le_strict(cs.namespace(|| "make bits of second word
328 328         ↪ for FS challenge"))?;
329 329
330 330 let take_bits = (<E as JubjubEngine>::Fs::CAPACITY / 2) as usize;
331 331
332 332 let mut bits = Vec::with_capacity(<E as JubjubEngine>::Fs::CAPACITY
333 333     ↪ as usize);
334 334 bits.extend_from_slice(&s0_bits[0..take_bits]);
335 335 bits.extend_from_slice(&s1_bits[0..take_bits]);
336 336 assert!(bits.len() == E::Fs::CAPACITY as usize);
```

CVF-140. INFO

- **Category** Readability
- **Source** withdraw.rs

Recommendation Subtracting the sum would be more readable.

Client Comment *The values here are all values that have been processed by the state handler. If you subtract a negative number, the state handler will return an insufficient balance error.*

```
292 292 bal.value.sub_assign(&amount_as_field_element);  
293 293 bal.value.sub_assign(&fee_as_field_element);
```

CVF-141. FIXED

- **Category** Unclear behavior
- **Source** transfer.rs

Description Seems this function is not used anywhere

Client Comment *Removed this function.*

```
140 140 pub fn get_sig_bits(&self) -> Vec<bool> {
```

CVF-142. FIXED

- **Category** Bad datatype
- **Source** transfer.rs

Recommendation This should be a named constant

Client Comment *Removed this function.*

```
144 144 &Fr::from_u64(5u64), //Corresponding tx_type
```


CVF-143. INFO

- **Category** Readability
- **Source** transfer.rs

Recommendation Subtracting the sum would be more readable.

Client Comment *The values here are all values that have been processed by the state handler. If you subtract a negative number, the state handler will return an insufficient balance error.*

```
240 240 bal.value.sub_assign(&amount_as_field_element);
241 241 bal.value.sub_assign(&fee_as_field_element);
```

CVF-144. INFO

- **Category** Bad datatype
- **Source** order_matching.rs

Recommendation Consider using designated types for that.

Client Comment *Considering.*

```
5 5 pub account: u32,
6 6 pub sub_account_id: u8,
7 7 pub slot_id: u32,
8 8 pub nonce: u32,
9 9 pub amount: u128,
10 10 pub price: u128,
11 11 pub is_sell: u8,
12 12 pub fee_ratio1: u8,
13 13 pub fee_ratio2: u8,
```

```
36 36 pub sub_account_id: u8,
37 37 pub tokens: (u32, u32),
38 38 pub fee: u128,
39 39 pub fee_token: u32,
40 40 pub submitter: u32,
```

```
43 43 pub is_refresh_order: (u8, u8),
```

CVF-145. INFO

- **Category** Suboptimal
- **Source** order_matching.rs

Description Data loss is possible when converting types here.

Recommendation Consider using checked conversion.

Client Comment *There is no data loss here because both data are converted from a small type to a large type.*

68 68

```
submitter: *order_matching.submitter as u32,
```

73 73

```
*order_matching.tx.maker.base_token_id as u32,
```

74 74

```
*order_matching.tx.taker.quote_token_id as u32,
```

77 77

```
fee_token: *order_matching.tx.fee_token as u32,
```

CVF-146. FIXED

- **Category** Suboptimal

- **Source** order_matching.rs

Recommendation These code blocks could be significantly simplified by calculating `min(residue1, residue2)`.

Client Comment *Adopted and optimized.*

```
282 282 if residue1 < residue2 {
283 283     let actual_exchanged = residue1
284 284         .checked_mul(&maker.price.into())?
285 285         .checked_div(&precision_magnified)?;
286 286     (Fr::from_big_uint(residue1), Fr::from_big_uint(actual_exchanged
        ↪ ))
287 287 } else {
288 288     let actual_exchanged = residue2
289 289         .checked_mul(&maker.price.into())?
290 290         .checked_div(&precision_magnified)?;
291 291     (Fr::from_big_uint(residue2), Fr::from_big_uint(actual_exchanged
        ↪ ))
292 292 }
```

```
294 294 if residue1 < residue2 {
295 295     let actual_exchanged = residue1
296 296         .checked_mul(&maker.price.into())?
297 297         .checked_div(&precision_magnified)?;
298 298     (Fr::from_big_uint(actual_exchanged), Fr::from_big_uint(residue1
        ↪ ))
299 299 } else {
300 300     let actual_exchanged = residue2
301 301         .checked_mul(&maker.price.into())?
302 302         .checked_div(&precision_magnified)?;
303 303     (Fr::from_big_uint(actual_exchanged), Fr::from_big_uint(residue2
        ↪ ))
304 304 }
```

CVF-147. INFO

- **Category** Procedural
- **Source** order_matching.rs

Description These parameters are not used.

Recommendation Consider removing them.

Client Comment *It is used on line 655.*

```
314 314 let maker_fee_ratio2_fe = Fr::from_u64(matching.maker.fee_ratio2 as
      ↪ u64);
315 315 let taker_fee_ratio1_fe = Fr::from_u64(matching.taker.fee_ratio1 as
      ↪ u64);
```

CVF-148. INFO

- **Category** Procedural
- **Source** order_matching.rs

Description We did not review this function

Client Comment *crypto/src/circuit/account.rs line 136*

```
390 390 ord.update(
```

```
426 426 ord.update(
```

CVF-149. INFO

- **Category** Suboptimal
- **Source** full_exit.rs

Recommendation The "not" call is redundant here. Just interchange the values to be selected.

Client Comment *The recommended changes require modifying the function entry or creating a new zero variable with 128bits, which increases the circuit overhead.*

```
78 78 Expression::constant::<CS>(E::Fr::zero()),
79 79 &cur.balance,
80 80 &is_address_correct.not(),
```

CVF-150. FIXED

- **Category** Bad datatype
- **Source** operation.rs

Recommendation The vector lengths should be named constants.

Client Comment *Adopted, added constant.*

```
139 139 pub frs_with_bool: ArgumentsWithSameLength<E, 2>,
140 140 pub frs_with_1_byte: ArgumentsWithSameLength<E,7>,
141 141 pub frs_with_2_bytes: ArgumentsWithSameLength<E,3>,
142 142 pub frs_with_4_bytes: ArgumentsWithSameLength<E,5>,
143 143 pub frs_with_8_bytes: ArgumentsWithSameLength<E,4>,
144 144 pub frs_with_15_bytes: ArgumentsWithSameLength<E,2>,
145 145 pub frs_with_16_bytes: ArgumentsWithSameLength<E,5>,
146 146 pub frs_with_20_bytes: ArgumentsWithSameLength<E,2>,
147 147 pub frs_with_max_bytes: ArgumentsWithSameLength<E,1>,
148 148
149 149 pub fees_packed: ArgumentsWithSameLength<E,1>,
150 150 pub amounts_packed: ArgumentsWithSameLength<E,2>,
```

CVF-151. INFO

- **Category** Suboptimal
- **Source** allocated_structures.rs

Recommendation Consider asserting that the constants are not too big to make an overflow.

Client Comment *The constant in the circuit means that once it is set, it cannot be changed unless the verification key is upgraded in the contract.*

```
146 146 sub_account_id.get_number().mul(
```

```
149 149 )?.add(
```

```
165 165 sub_account_id.get_number().mul(
```

```
168 168 )?.add(
```

CVF-152. FIXED

- **Category** Suboptimal
- **Source** allocated_structures.rs

Description This function is superseded with the "convert_amounts" function.

Recommendation Consider removing this function or refactoring the code to avoid duplication.

Client Comment *Removed.*

```
391 391 fn get_amounts<CS: ConstraintSystem<E>>(
```

CVF-153. FIXED

- **Category** Documentation
- **Source** utils.rs

Description The semantics of this argument is unclear.

Recommendation Consider documenting.

Client Comment *Adopted.*

```
76 76 offset_commitment: Vec<bool>,
```

CVF-154. FIXED

- **Category** Procedural
- **Source** utils.rs

Recommendation These extensions could be done once after the loop. Just calculate the correct numbers of elements to be appended.

Client Comment *Adopted.*

```
94 94 self.pubdata.extend(vec![false; CHUNK_BIT_WIDTH]);  
95 95 self.offset_commitment.extend(vec![false; 8])
```

CVF-155. INFO

- **Category** Overflow/Underflow
- **Source** utils.rs

Description Overflow may be possible here.

Recommendation Consider asserting that no information is lost after truncation

Client Comment *I don't think there are any overflow issues here, the data is handled by the state handler, and the type conversions are small to large.*

121 121
122 122

```
Some(Fr::from_u64(*self.fee_account_id as u64)),  
Some(Fr::from_u64(*self.block_number as u64)),
```

151 151
152 152
153 153

```
block_number: Some(Fr::from_u64(*self.block_number as u64)),  
block_timestamp: Some(Fr::from_u64(self.timestamp)),  
validator_address: Some(Fr::from_u64(*self.fee_account_id as  
↪ u64)),
```

413 413

```
let slot_id = calculate_actual_slot(sub_account_id.into(), slot_id.  
↪ into()).0 as u32;
```

464 464

```
let mut balance = validator_leaf.subtree.remove(*actual_token as u32  
↪ ).unwrap_or_default();
```

466 466

```
validator_leaf.subtree.insert(*actual_token as u32, balance);
```

CVF-156. FIXED

• **Category** Procedural

• **Source** utils.rs

Recommendation This code should probably be removed

Client Comment *Adopted.*

```
159 159 pub fn generate_dummy_sig_data(
160 160     bits: &[bool],
161 161     rescue_hasher: &RescueHasher<Bn256>,
162 162     rescue_params: &Bn256RescueParams,
163 163     jubjub_params: &AltJubjubBn256,
164 164 ) -> (SignatureData, Fr, Fr, Fr, Fr, Fr) {
165 165     let rng = &mut XorShiftRng::from_seed([0x3dbe_6258, 0x8d31_3d76,
166     ↪ 0x3237_db17, 0xe5bc_0654]);
167 166     let p_g = FixedGenerators::SpendingKeyGenerator;
168 167     let private_key = PrivateKey::<Bn256>(rng.gen());
169 168     let sender_pk = PublicKey::from_private(&private_key, p_g, &
170     ↪ jubjub_params);
171 169     let (sender_x, sender_y) = sender_pk.0.into_xy();
172 170     let mut sig_bits_to_hash = bits.to_vec();
173 171     assert!(sig_bits_to_hash.len() <= MAX_CIRCUIT_MSG_HASH_BITS);
174 172
175 173     sig_bits_to_hash.resize(MAX_CIRCUIT_MSG_HASH_BITS, false);
176 174     let (first_sig_part_bits, remaining) = sig_bits_to_hash.split_at
177     ↪ (Fr::CAPACITY as usize);
178 175     let remaining = remaining.to_vec();
179 176     let (second_sig_part_bits, third_sig_part_bits) = remaining.
180     ↪ split_at(Fr::CAPACITY as usize);
181 177     let first_sig_part: Fr = le_bit_vector_into_field_element(&
182     ↪ first_sig_part_bits);
183 178     let second_sig_part: Fr = le_bit_vector_into_field_element(&
184     ↪ second_sig_part_bits);
185 179     let third_sig_part: Fr = le_bit_vector_into_field_element(&
186     ↪ third_sig_part_bits);
187 180     let sig_msg = rescue_hasher.hash_bits(sig_bits_to_hash.clone());
188 181     let mut sig_bits: Vec<bool> = BitIterator::new(sig_msg.into_repr
189     ↪ ()).collect();
190 182     sig_bits.reverse();
191 183     sig_bits.resize(256, false);
```


CVF-157. FIXED

- **Category** Suboptimal

- **Source** utils.rs

Recommendation These two lines could be replaced with a single "resize_grow_only" call.

Client Comment *Adopted.*

```
198 198 assert!(sig_bits_to_hash.len() <= MAX_CIRCUIT_MSG_HASH_BITS);
```

```
200 200 sig_bits_to_hash.resize(MAX_CIRCUIT_MSG_HASH_BITS, false);
```

CVF-158. FIXED

- **Category** Suboptimal

- **Source** utils.rs

Description This code assumes the field size fits 256 bits, which may not be the case in the future.

Recommendation Consider asserting it explicitly in the code.

Client Comment *Fixed, and added constant.*

```
225 225 public_data_initial_bits.extend(vec![false; 256 - block_number_bits.  
    ↪ len()]);
```

```
231 231 public_data_initial_bits.extend(vec![false; 256 - validator_id_bits.  
    ↪ len()]);
```

```
242 242 let mut packed_old_root_bits = vec![false; 256 - old_root_bits.len()  
    ↪ ];
```

```
249 249 let mut packed_new_root_bits = vec![false; 256 - new_root_bits.len()  
    ↪ ];
```

```
256 256 let mut timestamp_bits = vec![false; 256 - timestamp_unpadded_bits.  
    ↪ len()];
```

```
532 532 let signer_pub_key_packed = vec![Some(false); 256];
```

CVF-159. INFO

- **Category** Suboptimal
- **Source** utils.rs

Recommendation Designated types from tx::* should be used here.

Client Comment *Considering.*

```
312 312 account_id: u32,  
313 313 sub_account_id: u8,  
314 314 token: u32,  
315 315 slot_id: u32,  
  
347 347 token: u32,  
  
381 381 account_address: u32,  
382 382 slot_number: u32,  
  
395 395 account_id: u32,  
396 396 (sub_account_id, token_id, slot_id): (u8, u32, u32),  
  
453 453 validator_address: u32,  
454 454 token: u32,  
455 455 fee: u128,
```

CVF-160. INFO

- **Category** Bad datatype
- **Source** utils.rs

Recommendation 32 should be a named constant

Client Comment *I think adding a comment is fine, since most regular signature rs are 32 bytes long.*

```
499 499 let (r_bytes, s_bytes) = sig_bytes.split_at(32);
```

CVF-161. FIXED

- **Category** Procedural
- **Source** utils.rs

Recommendation Consider moving this code to tests.

Client Comment *Adopted.*

```
643 643 /// Provides a vector of copies of this `SigDataInput` object, all  
        ↔ with one field  
644 644 /// set to incorrect value.  
645 645 /// Used for circuit tests.  
646 646 pub fn corrupted_variations(&self) -> Vec<Self> {
```

CVF-162. INFO

- **Category** Suboptimal
- **Source** forced_exit.rs

Recommendation `op_data` elements can be used here as they are checked against `pre_branch` in the code.

Client Comment *It is possible for each `OperationUint` to be executed on a different account, so it is appropriate to use `pre_branch` here.*

```
55 55 serialized_tx_bits.extend(pre_branch.account_id.get_bits_be());  
56 56 serialized_tx_bits.extend(pre_branch.sub_account_id.get_bits_be());
```

CVF-163. INFO

- **Category** Documentation
- **Source** forced_exit.rs

Recommendation Consider explaining in the doc why 'a' is computed correctly for this chunk.

Client Comment *Since we checked that `op_data.a` should be greater than or equal to `op_data.b` before executing all `op`, to ensure that the user balance would be greater than the amount deducted in the subsequent execution of multiple `op`, so we need to do an additional constraint check that the user balance is consistent with `op_data.a`.*

```
120 120 chunk1_valid_flags.push(is_a_correct);
```

CVF-164. FIXED

- **Category** Suboptimal

- **Source** serialization.rs

Recommendation All lengths must be global named constants.

Client Comment *Adopted, added constant.*

```
115 115 pub frs_with_bool: ArgumentsWithSameLength<Engine,2>,
```

```
117 117 pub frs_with_1_byte: ArgumentsWithSameLength<Engine,7>,
```

```
119 119 pub frs_with_2_bytes: ArgumentsWithSameLength<Engine,3>,
```

```
121 121 pub frs_with_4_bytes: ArgumentsWithSameLength<Engine,5>,
```

```
123 123 pub frs_with_8_bytes: ArgumentsWithSameLength<Engine,4>,
```

```
125 125 pub frs_with_15_bytes: ArgumentsWithSameLength<Engine,2>,
```

```
127 127 pub frs_with_16_bytes: ArgumentsWithSameLength<Engine,5>,
```

```
129 129 pub frs_with_20_bytes: ArgumentsWithSameLength<Engine,2>,
```

```
131 131 pub frs_with_max_bytes: ArgumentsWithSameLength<Engine,1>,
```

```
134 134 pub fees_packed: ArgumentsWithSameLength<Engine,1>,
```

```
136 136 pub amounts_packed: ArgumentsWithSameLength<Engine,2>,
```

CVF-165. FIXED

- **Category** Procedural

- **Source** element.rs

Recommendation Consider implementing this function on top of the "into_padded_le_bits" function.

Client Comment *Adopted.*

```
28 28 pub fn into_padded_be_bits(self, to_length: usize) -> Vec<Boolean> {
```

CVF-166. FIXED

- **Category** Documentation
- **Source** element.rs

Description Due to this commented line data loss is possible.

Recommendation Consider either uncommenting this or adding the word "unsafe" to the name of the function to distinguish it from a very similar function "into_padded_le_bits".

Client Comment *Adopted.*

```
30 30 // assert!(to_length >= bits.len());
```

CVF-167. FIXED

- **Category** Suboptimal
- **Source** element.rs

Description This check seems redundant, as self.length was already checked.

Recommendation Consider removing this check.

Client Comment *Adopted.*

```
41 41 assert!(n >= padded_bits.len());
```

CVF-168. FIXED

- **Category** Documentation
- **Source** element.rs

Description This comment is unclear.

Recommendation Consider elaborating more.

Client Comment *The comment and the following two lines of code have been removed because bits vector is originally E::Fr::NUM_BITS long.*

```
124 124 // this is safe due to "constants"
```

CVF-169. FIXED

- **Category** Documentation
- **Source** element.rs

Description This comment seems to be incorrect.

Client Comment *Modified to "chosen number as ce".*

```
207 207 cs.namespace(|| "chosen nonce"),
```

CVF-170. FIXED

- **Category** Suboptimal
- **Source** element.rs

Description Converting numbers via string looks weird.

Recommendation Consider implementing a more elegant approach.

Client Comment *Initialized the calculation with the BigUint type.*

```
311 311 let two = E::Fr::from_str("2").unwrap();  
312 312 let power = E::Fr::from_str(&length.to_string()).unwrap();
```

CVF-171. FIXED

- **Category** Procedural
- **Source** element.rs

Recommendation The function could be simplified by removing this line.

Client Comment *Initialized the calculation with the BigUint type.*

```
339 339 base.sub_assign(&E::Fr::one());
```

CVF-172. INFO

- **Category** Bad datatype
- **Source** full_exit.rs

Recommendation Consider using dedicated data types for these fields.

Client Comment *Considering.*

```
4 4 pub l2_source_token: u32,  
5 5 pub l1_target_token: u32,  
6 6 pub l1_target_token_after_mapping: u32,  
7 7 pub account_id: u32,  
8 8 pub to_chain_id: u8,  
9 9 pub sub_account_id: u8,
```

CVF-173. INFO

- **Category** Suboptimal
- **Source** full_exit.rs

Recommendation The '7' constant must be named.

Client Comment *It's weird to use a constant for 7 here, but our goal is to only set the first byte to 1.*

```
96 96 commitment[7] = true;
```

CVF-174. INFO

- **Category** Suboptimal
- **Source** forced_exit.rs

Recommendation This check seems redundant as one cannot generate a valid signature for a global asset account.

Client Comment *The purpose here is to prohibit anyone from using the global asset account as the target account for transfers, forced_exit operation, because the global asset account is only to record a layer of funds for each token in each chain, not the real token assets, to check whether the user's withdrawal token has enough funds in the target chain.*

```
36 +assert_ne!(forced_exit.target_account_id, GLOBAL_ASSET_ACCOUNT_ID);
```

CVF-175. INFO

- **Category** Suboptimal
- **Source** transfer_to_new.rs

Recommendation This check seems redundant as one cannot create a valid signature for the global asset account.

Client Comment *Same as the previous issue.*

```
29 +assert_ne!(transfer_to_new.to, GLOBAL_ASSET_ACCOUNT_ID);
```

CVF-176. INFO

- **Category** Suboptimal
- **Source** transfer.rs

Recommendation This check seems redundant, as one cannot create a valid signature for the global asset account.

Client Comment *Same as the previous issue.*

```
29 +assert_ne!(transfer.to, GLOBAL_ASSET_ACCOUNT_ID);
```

CVF-177. INFO

- **Category** Suboptimal
- **Source** forced_exit.rs

Recommendation This check seems redundant, as one cannot produce a valid signature for the global asset account.

Client Comment *Same as the previous issue.*

```
80 +base_flags.push(CircuitElement::equals(  
81 +   cs.namespace(|| "ForcedExit target account id != global asset  
   ↪ account(account_id=1)"),  
82 +   &op_data[ForcedExitArgs::TargetAccountId],  
83 +   &global_variables.explicit_one,  
84 +)??.not());
```


CVF-178. INFO

- **Category** Suboptimal
- **Source** transfer.rs

Recommendation This check seems redundant as one cannot generate a valid signature for the global asset account.

Client Comment *Same as the previous issue.*

```
84 +base_flags.push(CircuitElement::equals(  
85 +   cs.namespace(|| "FromAccount != global asset account(account_id  
    ↪ =1)"),  
86 +   &op_data[TransferArgs::FromAccountId],  
87 +   &global_variables.explicit_one,  
88 +)??.not());
```

CVF-179. INFO

- **Category** Suboptimal
- **Source** exit_circuit.rs

Description Returning vectors of the same values looks weird.

Recommendation Consider refactoring to avoid this.

Client Comment *Here, because of the usd token, each chain has multiple recharge tokens (usdc, usdt), so the circuit needs to calculate the withdrawal amount by the ratio of the total number of individual usdx tokens of a single chain to the total number of usd (the sum of each usdx token of each chain). But non-usd tokens do not provide so many tokens, because non-usd tokens have at most one recharge token per chain, so the total amount is just the sum of the recharge tokens (individual ones) for each chain. However, since the circuit is deterministic, I still have to consider n operationbranch for the usd token, and only the first operationbranch is needed for the non-usd token, and I just fill in some data for the other operationbranch. It is unlikely that the circuit will choose the non-first operationbranch in practice. So, in any case, I filled in some data for the witness, and, well, I didn't think of a better way to handle this part at the moment.*

```
258 +vec![global_account_witness; USDX_TOKEN_ID_RANGE as usize],  
259 +(vec![global_balance; USDX_TOKEN_ID_RANGE as usize],vec![  
    ↪ global_order; USDX_TOKEN_ID_RANGE as usize])
```

```
296 +vec![global_audit_path; USDX_TOKEN_ID_RANGE as usize],  
297 +(vec![global_audit_balance_path; USDX_TOKEN_ID_RANGE as usize],vec  
    ↪ ![global_audit_order_path; USDX_TOKEN_ID_RANGE as usize])
```

CVF-180. FIXED

- **Category** Procedural
- **Source** LayerZeroBridge.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
38 +receive() external payable {}
```

CVF-181. FIXED

- **Category** Bad datatype
- **Source** LayerZeroBridge.sol

Recommendation The type of the “_zklink” argument should be “IZkLink”.

```
43 +constructor(address _governor, address _zklink, ILayerZeroEndpoint  
↪ _endpoint) {
```

CVF-182. FIXED

- **Category** Suboptimal
- **Source** LayerZeroBridge.sol

Recommendation This check is redundant, as it is anyway possible to pass a dead zkLink address.

```
45 +require(_zklink != address(0), "ZkLink not set");
```

CVF-183. FIXED

- **Category** Readability
- **Source** LayerZeroBridge.sol

Recommendation Should be “underflow”.

```
118 +uint256 originBalance = address(this).balance - msg.value; //  
↪ overflow is impossible
```

```
125 +uint256 leftMsgValue = address(this).balance - originBalance; //  
↪ overflow is impossible
```

CVF-184. FIXED

- **Category** Suboptimal
- **Source** LayerZeroBridge.sol

Recommendation The length check is redundant, as the hash check supersedes it.

```
162 +require(path.length == srcPath.length && keccak256(path) ==  
    ↪ keccak256(srcPath), "Invalid src");
```

CVF-185. FIXED

- **Category** Bad datatype
- **Source** LayerZeroStorage.sol

Recommendation The type of this variable should be "IZkLink".

```
15 +address public zklink;
```

CVF-186. INFO

- **Category** Bad datatype
- **Source** DeployFactory.sol

Recommendation The type of the "_zkLinkTarget" should be "IZkLink".

Client Comment *IZkLink contains all interfaces, and its implementation is divided into two logical contracts: ZkLink and ZkLinkPeriphery.*

```
30 +constructor(IVerifier _verifierTarget, ZkLink _zkLinkTarget,  
    ↪ ZkLinkPeriphery _peripheryTarget, uint32 _blockNumber, uint256  
    ↪ _timestamp, bytes32 _stateHash, bytes32 _commitment, bytes32  
    ↪ _syncHash, address _firstValidator, address _governor, address  
    ↪ _feeAccountAddress) {
```

CVF-187. FIXED

- **Category** Procedural
- **Source** EmptyVerifier.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
11 +function initialize(bytes calldata) external {}
```

```
16 +function upgrade(bytes calldata upgradeParameters) external {}
```



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