Report

v. 1.0

Customer

StarkWare

Cairo Audit

Perpetual

28th February 2022

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	28.02.22	A. Zveryanskaya	Initial Draft
0.2	28.02.22	A. Zveryanskaya	Minor revision
1.0	28.02.22	A. Zveryanskaya	Release

2 Introduction

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

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

exchange/definitions/		
constants.cairo		
exchange/		
order.cairo		
exchange/		
signature_message_hashes.cairo		
perpetual/definitions/		
constants.cairo	general_config_hash.cairo	general_config.cairo
objects.cairo	perpetual_error_code.cairo	
perpetual/oracle/		
oracle_price.cairo		
perpetual/order/		
limit_order.cairo	order.cairo	validate_limit_order.cairo
perpetual/output/		
data_availability.cairo	forced.cairo	program_input.cairo
program_output.cairo		

perpetual/position/			
	add_asset.cairo	check_smaller_holdings.cairo	funding.cairo
	hash.cairo	position.cairo	serialize_change.cairo
	status.cairo	update_position.cairo	validate_state_transition.cairo
perpetual/state/			
	state.cairo		
perpetual/transactions/			
	batch_config.cairo	conditional_transfer.cairo	deleverage.cairo
	deposit.cairo	execute_limit_order.cairo	forced_trade.cairo
	forced_withdrawal.cairo	funding_tick.cairo	liquidate.cairo
	oracle_prices_tick.cairo	trade.cairo	transaction.cairo
	transfer.cairo	withdrawal.cairo	
perpetual/			
	execute_batch_utils.cairo	execute_batch.cairo	main.cairo
starkware/			
	alloc.cairo	cairo_builtins.cairo	dict_access.cairo
	dict.cairo	find_element.cairo	hash_chain.cairo
	hash_state.cairo	hash.cairo	invoke.cairo
	math_cmp.cairo	math.cairo	memcpy.cairo
	merkle_multi_update.cairo	merkle_update.cairo	registers.cairo
	serialize.cairo	signature.cairo	small_merkle_tree.cairo
	squash_dict.cairo		

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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 3 critical, and a few less important issues. All identified Critical issues have been fixed.

Issues		
Severity	Info	Fixed
Critical	0	3
Major	0	3

Fixed 6 out of 6 issues

ABDK

6 Critical Issues

CVF-1 FIXED

- **Category** Flaw
- **Source** math.cairo

Description For high=low=2^128-1 we have diff=2^250-2^122+2^128-1 >2^250 .

Client Comment *This is fixed in a later cairo version (There, we have assert_250_bit instead of assert_le_250_bit and we check that 0 <= low < 2**122) . For now this is not a problem because in all the places we use this function, we can assume the inputs are between -2**224 and 2**224*

```
# Asserts that a <= b. More specifically, asserts that b - a is in
   ↪ the range [0, 2**250).
```

```
    assert diff = high * HIGH_PART_SHIFT + low
```

CVF-2 FIXED

- **Category** Flaw
- **Source** oracle_price.cairo

Description Should be ‘assert_nn_lt‘ to avoid collisions in ‘y‘.

```
assert_nn_le{range_check_ptr=range_check_ptr}(sig.timestamp,
   ↪ TIMESTAMP_BOUND)
```

```
        x=sig.signed_asset_id, y=sig.external_price *
           ↪ TIMESTAMP_BOUND + sig.timestamp)
```

CVF-3 FIXED

- **Category** Flaw
- **Source** order.cairo

Description Since ‘assert_nn_le‘ does not upper bound the second argument, it is possible to submit a small ‘full_amount‘ such that remaining_capacity is negative. It then possible to submit a very large ‘update_amount‘ that far exceeds ‘full_amount ‘

```
assert_nn_le{range_check_ptr=range_check_ptr}(update_amount,
    ↪ remaining_capacity)
```

7 Major Issues

CVF-6 FIXED

- **Category** Flaw
- **Source** math_cmp.cairo

Description This returns true for b<a+RANGE_CHECK_BOUND which includes many b bigger than RANGE_CHECK_BOUND. The code should include ‘is_nn(b)‘

Client Comment *There is an assumption that b < RANGE_CHECK_BOUND. Added documentation.*

```
return is_le(a, b)
```

CVF-9 FIXED

- **Category** Flaw
- **Source** general_config_hash.cairo

Description The common approach to hashing a multi-level structure is to hash the metadata along with the data in order to ensure that hashes of different structures never collide. In order to hash the metadata, one needs to design an injective encoding. One way to do it is to define recursively, where the concatenation of structures S=S1+S2+...Sk is encoded as Enc(S1+S2+...Sk) = hash(k || Enc(S1) || ... || Enc(Sk)) and Enc(F: felt) = hash(1)

Client Comment *We added a versioning to the hash. This way, we ensure the hash won’t collide with the hash of a future version that has a different structure.*

```
func general_config_hash{pedersen_ptr : HashBuiltin*}(
  ↪ general_config_ptr : GeneralConfig*) -> (
```

CVF-12 FIXED

- **Category** Unclear behavior
- **Source** order.cairo

Description The ”remaining_capacity” value is used before being assigned.

Recommendation Consider moving the error code calculation below the next statement.

```
if ids.update_amount > ids.remaining_capacity:
```

8 Moderate Issues

CVF-7 INFO

- **Category** Suboptimal
- **Source** find_element.cairo

Description Binary search would be much more efficient for sorted arrays.

Client Comment *Will be added to a future version of cairo. We will then update the perpetual code to use that version.*

```
func search_sorted{range_check_ptr}(array_ptr : felt*, elm_size,
    ↪ n_elms, key) -> (
```

CVF-8 INFO

- **Category** Procedural
- **Source** position.cairo

Description The same error code is returned in two very different situations: when the request public key is invalid (zero), and when the request public key doesn't match the position public key.

Recommendation Consider using different error codes for these two situations.

Client Comment *The error codes are used only for testing in order to check that the cairo code corresponds with our BE.*

```
    return (return_code=PerpetualErrorCode.INVALID_PUBLIC_KEY)
```

```
return (return_code=PerpetualErrorCode.INVALID_PUBLIC_KEY)
```

CVF-13 FIXED

- **Category** Flaw
- **Source** deposit.cairo

Description Should be "AMOUNT_UPPER_BOUND - 1" rather than just "AMOUNT_UPPER_BOUND".

```
assert_nn_le{range_check_ptr=range_check_ptr}(tx.amount,
    ↪ AMOUNT_UPPER_BOUND)
```

CVF-14 FIXED

- **Category** Flaw
- **Source** deleverage.cairo

Description Should be "AMOUNT_UPPER_BOUND - 1" instead of just "AMOUNT_UPPER_BOUND".

```
assert_nn_le{range_check_ptr=range_check_ptr}(tx.amount_synthetic, AMOUNT_UPPER_BOUND)
assert_nn_le{range_check_ptr=range_check_ptr}(tx.amount_collateral, AMOUNT_UPPER_BOUND)
```

CVF-15 FIXED

- **Category** Unclear behavior
- **Source** execute_batch.cairo

Description The validity period together with several funding ticks for the same timestamp value, allows reordering funding ticks with transactions, which could potentially be abused.

Recommendation Consider enforcing that funding and price ticks are always the first two transactions in a batch, so all other transactions in the same batch use the same prices and funding indexes.

Client Comment *dYdX are the source of the all transactions. Even if we add such a limitation, dYdX could censor price changes, delay execution until the next price updates, etc.*

```
batch_config.general_config.timestamp_validation_config.funding_validity_period)
```

CVF-16 FIXED

- **Category** Flaw
- **Source** execute_batch_utils.cairo

Description Should be "N_ASSETS_UPPER_BOUND - 1" instead of just "N_ASSETS_UPPER_BOUND".

```
assert_le{range_check_ptr=range_check_ptr}(
    general_config.n_synthetic_assets_info, N_ASSETS_UPPER_BOUND)
```

CVF-17 FIXED

- **Category** Flaw
- **Source** liquidate.cairo

Description Should be "AMOUNT_UPPER_BOUND - 1" rather than just "AMOUNT_UPPER_BOUND".

```
48  assert_nn_le{range_check_ptr=range_check_ptr}(tx.actual_collateral,
       ↪ AMOUNT_UPPER_BOUND)
    assert_nn_le{range_check_ptr=range_check_ptr}(tx.
       ↪ actual_liquidator_fee, AMOUNT_UPPER_BOUND)
```

CVF-18 FIXED

- **Category** Flaw
- **Source** forced_withdrawal.cairo

Description Should be "AMOUNT_UPPER_BOUND - 1" rather then just "AMOUNT_UPPER_BOUND".

```
36  assert_nn_le{range_check_ptr=range_check_ptr}(tx.amount,
       ↪ AMOUNT_UPPER_BOUND)
```

CVF-19 FIXED

- **Category** Flaw
- **Source** forced_trade.cairo

Description Should be "AMOUNT_UPPER_BOUND - 1" rather than just "AMOUNT_UPPER_BOUND".

```
105 assert_nn_le{range_check_ptr=range_check_ptr}(tx.amount_collateral,
       ↪ AMOUNT_UPPER_BOUND)
    assert_nn_le{range_check_ptr=range_check_ptr}(tx.amount_synthetic,
       ↪ AMOUNT_UPPER_BOUND)
```

CVF-20 FIXED

- **Category** Documentation
- **Source** funding.cairo

Description This statement is vague and difficult to verify.

Recommendation Consider giving explicit bounds for arguments in assumptions and asserting in the code that the arithmetic operations do not yield a number beyond certain bound.

Client Comment *There's no need to give explicit bounds because current_collateral_fxp is around 95 bits and the overflow limit is more than 251 bits. We documented this better.*

```
# Assumption: current_collateral_fxp does not overflow, it is a sum of 95 bit values.
```

CVF-21 INFO

- **Category** Unclear behavior
- **Source** add_asset.cairo

Description The public key verification is bypassed in case delta is zero.

Client Comment *This verification is verified also in position_add_collateral and both position_add_collateral and position_add_asset are only used in update_position. Therefore we decided to move the public key verification to update_position altogether.*

```
if delta == 0:
    return (
```

```
# Verify public_key.
```

CVF-22 INFO

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

Description The actual signature variables are not used directly but rather elsewhere. This is error-prone. It is also unclear if the last two parameters are asserted to be the signature data, or it is just checked that the prover knows the signature. Note that in the latter case it becomes difficult to verify signatures from a public input.

Recommendation Consider making the signature part of the builtin

Client Comment *In the future we intend to verify it as well. Unfortunately, Cairo doesn't support it. In the meantime, we keep the interface as if it does, for future proofing.*

```
7 func verify_ecdsa_signature{ecdsa_ptr : SignatureBuiltin*}(
          message, public_key, signature_r, signature_s):
```

CVF-23 INFO

- **Category** Unclear behavior
- **Source** oracle_price.cairo

Description No range check is done for factors.

Recommendation Consider adding them here or into the assumptions.

Client Comment *The external price is checked at the start of the function. the resolutions are checked in validate_general_config in execute_batch_utils.cairo. FXP_32_ONE and EXTERNAL_PRICE_FIXED_POINT_UNIT are constants.*

```
115 let numerator = sig.external_price * collateral_resolution *
      ↪ FXP_32_ONE
```

```
117 tempvar denominator = asset_info.resolution *
      ↪ EXTERNAL_PRICE_FIXED_POINT_UNIT
```

CVF-24 INFO

- **Category** Suboptimal
- **Source** hash_state.cairo

Description This function is very complicated and uses too much memory. It can be efficiently computed as func hash_update_inner{hash_ptr : HashBuiltin*}(data_ptr : felt*, data_length : felt, input_hash : felt) -> (out : felt): if data_length=0 return input_hash hash_ptr.x=input_hash hash_ptr.y=[data_ptr] tempvar start_position=0; hash_loop: tempvar i = [ap-1]+1; #index of data we are feeding (hash_ptr+i*HashBuiltin.SIZE).x = (hash_ptr+(i-1)*HashBuiltin.SIZE).result #chaining the computation (hash_ptr+i*HashBuiltin.SIZE).y = [data_ptr+i] # feeding the data jmp hash_loop if i+1!=data_length let hash_ptr = hash_ptr+data_length*HashBuiltin.SIZE return out=(hash_ptr-HashBuiltin.SIZE).result

Client Comment *In cairo, calculating from an index the data that we need is as expensive as keeping an extra field for the data. This is because only the amount of instructions (cairo steps) is the thing that matters, not the amount of memory we used.*

```
func hash_update_inner{hash_ptr : HashBuiltin*}(
```

CVF-25 FIXED

- **Category** Flaw
- **Source** general_config_hash.cairo

Description Here array elements are hashed, but array lengths are not. This could lead to hash collisions.

Recommendation Consider hashing the array lengths along with array elements.

```
let (hash_state_ptr) = hash_update(
    hash_state_ptr,
    synthetic_asset_info_ptr.oracle_price_signed_asset_ids,
    synthetic_asset_info_ptr.n_oracle_price_signed_asset_ids)
```

```
let (hash_state_ptr) = hash_update(
    hash_state_ptr,
    synthetic_asset_info_ptr.oracle_price_signers,
    synthetic_asset_info_ptr.n_oracle_price_signers)
```

9 Minor Issues

CVF-26 FIXED

- **Category** Documentation
- **Source** withdrawal.cairo

Description 0x6 is 3 bits rather than 10.

Client Comment *It is 0x6 written in 10 bits.*

```
#   w2= 0x6 (10 bit) || vault_from (64 bit) || nonce (64 bit) ||
    ↪ expiration_timestamp (32 bit)
```

CVF-27 INFO

- **Category** Readability
- **Source** withdrawal.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func withdrawal_hash(
        pedersen_ptr : HashBuiltin*, withdrawal : Withdrawal*,
           ↪ asset_id_collateral) -> (
        pedersen_ptr : HashBuiltin*, message):
```

```
func execute_withdrawal(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : Withdrawal*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-28 INFO

- **Category** Procedural
- **Source** withdrawal.cairo

Recommendation Consider adding an assert that the results fits into a field element

Client Comment *This change is of low priority and will reduce the efficiency of the code. We should consider adding a comment if it isn't clear.*

```
let packed_message = WITHDRAWAL
let packed_message = packed_message * POSITION_ID_UPPER_BOUND +
   ↪ withdrawal.position_id
let packed_message = packed_message * NONCE_UPPER_BOUND + withdrawal
   ↪ .base.nonce
let packed_message = packed_message * AMOUNT_UPPER_BOUND +
   ↪ withdrawal.amount
let expiration_timestamp = withdrawal.base.expiration_timestamp
let packed_message = packed_message *
   ↪ EXPIRATION_TIMESTAMP_UPPER_BOUND + expiration_timestamp
let packed_message = packed_message * %[2**49%]  # Padding.
```

CVF-29 INFO

- **Category** Readability
- **Source** deposit.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big and of relatively low priority.*

```
func execute_deposit(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : Deposit*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-30 FIXED

- **Category** Documentation
- **Source** deposit.cairo

Description A comment is needed why we shift the amount here.

Recommendation A comment is needed why we shift the amount here.

```
assert modification.biased_delta = tx.amount + AMOUNT_UPPER_BOUND
```

CVF-31 INFO

- **Category** Readability
- **Source** deleverage.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func execute_deleverage(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : Deleverage*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-32 FIXED

- **Category** Documentation
- **Source** deleverage.cairo

Description It should be explained, where it is checked that without reducing the collateral the transaction is valid.

```
# Validates that deleverage ratio for the deleverager is the maximal
   ↪  it can be while being valid
# for the deleveragable. In other words, validates that if we reduce
   ↪  the collateral the
# deleveragable gets from the transaction by 1, the transaction is
   ↪ invalid.
```

CVF-33 FIXED

- **Category** Readability
- **Source** state.cairo

Recommendation Consider using the ”let local” syntax to simplify the code.

```
57 local squashed_positions_dict_end : DictAccess*
59 let (squashed_positions_dict_end_) = squash_dict(
63 squashed_positions_dict_end = squashed_positions_dict_end_
66 local squashed_orders_dict_end : DictAccess*
68 let (squashed_orders_dict_end_) = squash_dict(
72 squashed_orders_dict_end = squashed_orders_dict_end_
```

CVF-34 FIXED

- **Category** Readability
- **Source** state.cairo

Description This should be done via the ”alloc” function.

```
58 %{ ids.squashed_positions_dict = segments.add() %}
67 %{ ids.squashed_orders_dict = segments.add() %}
```

CVF-35 INFO

- **Category** Readability
- **Source** state.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big and of relatively low priority.*

115
```
func shared_state_apply_state_updates(
        hash_ptr : HashBuiltin*, shared_state : SharedState*,
        squashed_carried_state : SquashedCarriedState*,
           ↪ general_config : GeneralConfig*) -> (
        hash_ptr : HashBuiltin*, shared_state : SharedState*):
```

CVF-36 INFO

- **Category** Documentation
- **Source** state.cairo

Description It is unclear what is "new_positions_root" at the right side of the assignment.

128
```
%{ ids.new_positions_root = new_positions_root %}
```

138
```
    %{ ids.new_orders_root = new_orders_root %}
```

CVF-37 INFO

- **Category** Suboptimal
- **Source** state.cairo

Description The first two lines can be swapped so that there is no need to subtract 1 later.

Client Comment *We use output_start_ptr in serialize_word. Swapping the lines would force us to substract 1 in that line so it will not be more efficient.*

161
```
local output_start_ptr : felt* = output_ptr
```

164
```
let output_ptr = output_ptr + 1
```

177
```
let size = cast(output_ptr, felt) - cast(output_start_ptr, felt) - 1
```

CVF-38 INFO

- **Category** Readability
- **Source** main.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func main(
        output_ptr : felt*, pedersen_ptr : HashBuiltin*,
           ↪ range_check_ptr,
        ecdsa_ptr : SignatureBuiltin*) -> (
        output_ptr : felt*, pedersen_ptr : HashBuiltin*,
           ↪ range_check_ptr,
        ecdsa_ptr : SignatureBuiltin*):
```

CVF-39 INFO

- **Category** Readability
- **Source** execute_batch.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func execute_transaction(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*,
           ↪ batch_config : BatchConfig*,
        tx : Transaction*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

```
func execute_batch_transactions(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*,
           ↪ batch_config : BatchConfig*,
        n_txs : felt, tx : Transaction*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

```
func execute_batch(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, program_input : ProgramInput
           ↪ *, outputs : PerpetualOutputs*,
        txs : Transactions*, end_system_time) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-40 INFO

- **Category** Documentation
- **Source** execute_batch_utils.cairo

Description It is unclear what exactly this functions validates.

Recommendation Consider documenting.

Client Comment *This function is a helper function and what it does is best explained by the documentation of the external function. Documenting this function will only cause confusion.*

```
func validate_funding_indices_in_general_config_inner(
```

CVF-41 INFO

- **Category** Readability
- **Source** execute_batch_utils.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func validate_assets_config_inner(
        range_check_ptr, synthetic_assets_info_ptr : SyntheticAssetInfo*, n_synthetic_assets_info,
        prev_asset_id) -> (range_check_ptr):
```

```
func validate_assets_config(range_check_ptr, general_config : GeneralConfig*) -> (range_check_ptr):
```

```
func validate_general_config(range_check_ptr, general_config : GeneralConfig*) -> (range_check_ptr):
```

CVF-42 INFO

- **Category** Procedural
- **Source** execute_batch_utils.cairo

Description In many other files, "assert_le(x, y - 1)" is used instead of "assert_lt(x, y)". Consider using consistent approach across the code.

Client Comment *We decided to keep it.*

```
    assert_lt{range_check_ptr=range_check_ptr}(prev_asset_id,
        ↪ ASSET_ID_UPPER_BOUND)
```

```
assert_lt{range_check_ptr=range_check_ptr}(prev_asset_id,
    ↪ synthetic_assets_info_ptr.asset_id)
```

CVF-43 FIXED

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

Description There is already a "nonce" field inside "OrderBase" struct. Why to have another nonce outside?

```
member base : OrderBase*
member nonce : felt
```

CVF-44 FIXED

- **Category** Suboptimal
- **Source** transfer.cairo

Description This field is not used.

Recommendation Consider removing it.

```
member nonce : felt
```

CVF-45 INFO

- **Category** Readability
- **Source** transfer.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func transfer_hash(pedersen_ptr : HashBuiltin*, transfer : Transfer
   ↪ *, condition : felt) -> (
        pedersen_ptr : HashBuiltin*, message):
```

```
func execute_transfer(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : Transfer*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-46 INFO

- **Category** Readability
- **Source** transaction.cairo

Recommendation Consider defining all the transaction structs in this file to make the whole picture easier to understand.

Client Comment *We prefer splitting the files because then when we add a new transaction there are less places that we need to edit and each transaction is self contained.*

```
member tx : felt*
```

CVF-47 INFO

- **Category** Readability
- **Source** trade.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func execute_trade(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : Trade*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-48 INFO

- **Category** Readability
- **Source** oracle_prices_tick.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

26
```
func insert_oracle_prices_until_asset_id(
        range_check_ptr, oracle_price_ptr : OraclePrice*,
           ↪ n_oracle_prices, asset_id_bound,
        new_oracle_price_ptr : OraclePrice*) -> (range_check_ptr,
           ↪ n_new_oracle_prices):
```

50
```
func create_new_oracle_prices_and_validate_tick(
        range_check_ptr, prev_oracle_price_ptr : OraclePrice*,
           ↪ n_oracle_prices,
        tick_price_ptr : OraclePrice*, n_tick_prices,
           ↪ last_tick_asset_id,
        batch_config : BatchConfig*, new_oracle_price_ptr :
           ↪ OraclePrice*) -> (
        range_check_ptr, n_new_oracle_prices):
```

140
```
func execute_oracle_prices_tick(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : OraclePricesTick*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-49 FIXED

- **Category** Procedural
- **Source** oracle_prices_tick.cairo

Description In other places such checks look like: assert_le{range_check_ptr=range_check_ptr}(last_tick_asset_id, ASSET_ID_UPPER_BOUND - 1)

56
```
assert_le{range_check_ptr=range_check_ptr}(last_tick_asset_id + 1,
    ↪ ASSET_ID_UPPER_BOUND)
```

CVF-50 FIXED

- **Category** Documentation
- **Source** oracle_prices_tick.cairo

Description Should be "not smaller" instead of "larger".

```
# Check that new timestamp is larger than previous system time.
```

CVF-51 INFO

- **Category** Readability
- **Source** liquidate.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func execute_liquidate(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : Liquidate*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-52 FIXED

- **Category** Suboptimal
- **Source** liquidate.cairo

Description This could be simplified as: assert_nn_le{range_check_ptr=range_check_ptr}(synthetic_delta, -initial_liquidated_asset_balance)

```
assert_in_range{range_check_ptr=range_check_ptr}(
    -synthetic_delta, initial_liquidated_asset_balance, 1)
```

CVF-53 INFO

- **Category** Unclear behavior
- **Source** liquidate.cairo

Description This checks that the price is fine from the liquidator's point of view. How is it guaranteed that the price is fair for the party being liquidated?

Client Comment *There is no order for the liquidated party. This is because the liquidation is done without the liquidated party's agreement because the liquidated party has reached a liquidatable status. Therefore the liquidation doesn't need to be fair for that party.*

```
let (pedersen_ptr, range_check_ptr, ecdsa_ptr, carried_state) = execute_limit_order(
```

CVF-54 INFO

- **Category** Readability
- **Source** funding_tick.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func validate_funding_index_diff_in_range(
        range_check_ptr, max_funding_rate, funding_index_diff,
           ↪ timestamp_diff, price) -> (
        range_check_ptr):
```

```
func validate_funding_tick_inner(
        range_check_ptr, prev_funding_index_ptr : FundingIndex*,
        new_funding_index_ptr : FundingIndex*, oracle_price_ptr :
           ↪ OraclePrice*,
        last_new_funding_asset_id, args :
           ↪ ValidateFundingTickInnerArgs*) -> (
        range_check_ptr, prev_funding_index_ptr : FundingIndex*,
        new_funding_index_ptr : FundingIndex*, oracle_price_ptr :
           ↪ OraclePrice*):
```

```
func validate_funding_tick(
        range_check_ptr, carried_state : CarriedState*,
           ↪ general_config : GeneralConfig*,
        new_funding_indices : FundingIndicesInfo*) -> (
           ↪ range_check_ptr):
```

```
func execute_funding_tick(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
```

CVF-55 FIXED

- **Category** Documentation
- **Source** funding_tick.cairo

Description Should be "not smaller" instead of "larger".

```
# Check that new timestamp is larger than previous system time.
# If signatures will be required to verify OraclePricesTick, then
   ↪ the timestamps for the
# oracle prices in the carried state will be verified here.
assert_le{range_check_ptr=range_check_ptr}(
    carried_state.system_time, new_funding_indices.funding_timestamp
       ↪ )
```

CVF-56 INFO

- **Category** Readability
- **Source** forced_withdrawal.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func execute_forced_withdrawal(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : ForcedWithdrawal*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-57 INFO

- **Category** Readability
- **Source** forced_trade.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions*

```
func try_to_trade(
        range_check_ptr, carried_state : CarriedState*,
           ↪ position_buyer : Position*,
        position_seller : Position*, public_key_buyer,
           ↪ public_key_seller, synthetic_asset_id,
        amount_collateral, amount_synthetic, general_config :
           ↪ GeneralConfig*) -> (
        range_check_ptr, position_buyer : Position*, position_seller
           ↪  : Position*, return_code):
```

```
func execute_forced_trade(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : ForcedTrade*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-58 INFO

- **Category** Suboptimal
- **Source** forced_trade.cairo

Description This should be executed only in case the buyer's position update was successful.

Client Comment *This should be classified as suboptimal and not flaw. Since forced trade is rare, added a comment on the potential optimization instead.*

```
let (range_check_ptr, local updated_position_seller, local
   ↪ funded_position_seller,
    local return_code_b) = update_position(
    range_check_ptr=range_check_ptr,
    position=position_seller,
    request_public_key=public_key_seller,
    collateral_delta=amount_collateral,
    synthetic_asset_id=synthetic_asset_id,
    synthetic_delta=-amount_synthetic,
    global_funding_indices=carried_state.global_funding_indices,
    oracle_prices=carried_state.oracle_prices,
    general_config=general_config)
```

CVF-59 INFO

- **Category** Unclear behavior
- **Source** forced_trade.cairo

Description This potentially hides the second error.

Client Comment *This is not an issue. Because A was executed before B, It makes sense to return A's error. We could add a comment about it but IMO it's unnecessary*

```
return_code = return_code_a
```

CVF-60 INFO

- **Category** Suboptimal
- **Source** math.cairo

Description This function could be simplified as: [ap] = 1; ap++ [ap] = [ap - 1] / value

Client Comment *The suggested code is less efficient. The current code performs 1 cairo step in the succesive flow (the negative flow is irrelevant as the program crashes). The suggested code performs 2 cairo steps*

```
func assert_not_zero(value):
```

CVF-61 INFO

- **Category** Suboptimal
- **Source** math.cairo

Description This function could be simplified as: let diff = b - a [ap] = 1; ap++ [ap] = [ap - 1] / diff

Client Comment *The suggested code is less efficient. The current code performs 1 cairo step in the succesive flow (the negative flow is irrelevant as the program crashes). The suggested code performs 2 cairo steps*

```
func assert_not_equal(a, b):
```

CVF-62 INFO

- **Category** Documentation
- **Source** math.cairo

Description This function covers also negative 'a' which should be stated in the comment.

Client Comment *The function covers some of the negative values. I don't think there is an elegant way do describe it besides what is already written.*

```
# Verifies that a <= b (or more precisely 0 <= b - a <
   ↪ RANGE_CHECK_BOUND).
func assert_le{range_check_ptr}(a, b):
```

```
# Verifies that a <= b - 1 (or more precisely 0 <= b - 1 - a <
   ↪ RANGE_CHECK_BOUND).
```

CVF-63 INFO

- **Category** Suboptimal
- **Source** math.cairo

Description It would be more efficient to use range check builtin here directly, rather than call other functions.

Client Comment *This is dwarved by the range check usages and reduces readability*

```
assert_nn(b - a)
```

```
assert_le(a, b - 1)
```

CVF-64 FIXED

- **Category** Documentation
- **Source** math.cairo

Description For "a" this is not an assumption but rather an enforced constraint.

```
# Prover assumption: a, b < RANGE_CHECK_BOUND.
```

CVF-65 FIXED

- **Category** Documentation
- **Source** math.cairo

Description This comment is inaccurate. The actual condition being checked is more complicated. Consider explaining the real condition being checked.

```
# Asserts that value is in the range [lower, upper).
```

ABDK

CVF-66 INFO

- **Category** Procedural
- **Source** math.cairo

Description The RANGE CHECK BOUND constant should have been used here

Client Comment *This function doesn't exist in our latest cairo version.*

```
const HIGH_PART_SHIFT = %[2**250 // 2**128 %]
```

```
    assert range_check_builtin.bound == 2**128
```

CVF-67 INFO

- **Category** Bad datatype
- **Source** math.cairo

Description These constants should be named and declared in some globally imported file

Client Comment *These constants can have different values depending on the function they are in and are used with this value only here.*

```
const MAX_HIGH = %[(PRIME - 1) >> 128%]
const MAX_LOW = %[(PRIME - 1) & ((1 << 128) - 1)%]
```

```
assert value = high * %[2**128%] + low
```

CVF-68 FIXED

- **Category** Readability
- **Source** math.cairo

Description Should be MAX_HIGH-1

```
assert_le(high, MAX_HIGH)
```

CVF-69 INFO

- **Category** Suboptimal
- **Source** math.cairo

Description This function can be computed via ‘assert_lt_felt‘ to avoid code duplication

Client Comment *Computing one function with the other will reduce its efficiency.*

```
func assert_le_felt{range_check_ptr}(a, b):
```

CVF-70 INFO

- **Category** Suboptimal
- **Source** math.cairo

Description Either of these functions can be computed via the other one. Consider dropping one of them

Client Comment *Using one function in the other will reduce the efficiency of the function.*

```
func abs_value{range_check_ptr}(value) -> (abs_value):
```

```
func sign{range_check_ptr}(value) -> (sign):
```

CVF-71 INFO

- **Category** Readability
- **Source** math.cairo

Description Using the same name for different variables is discouraged

Client Comment *See CVF-153.*

```
tempvar abs_value = value * (-1)
```

```
return (abs_value=abs_value)
```

CVF-72 INFO

- **Category** Documentation
- **Source** math.cairo

Description It is unclear what ‘assumption‘ means. Is it the set of inputs on which the function is correct? Does it fail on other ones?

Client Comment *If this assumption is not true, the function’s result is undefined.*

```
214 # Assumption: 0 < div <= PRIME / rc_bound.
```

CVF-73 INFO

- **Category** Documentation
- **Source** math.cairo

Description Should be ‘<PRIME‘

Client Comment *q < rc_bound, but q+1 might be equal to rc_bound, and div <= PRIME / rc_bound. Therefore (q + 1) * div might be equal to PRIME*

```
218 # q * div + r < (q + 1) * div <= rc_bound * (PRIME / rc_bound) =
       ↪ PRIME.
```

CVF-74 FIXED

- **Category** Documentation
- **Source** math.cairo

Description Should be either "0 <= r <= div-1" or "0 <= r < div".

```
234 # Returns q and r such that. -bound <= q < bound, 0 <= r < div -1
       ↪ and value = q * div + r.
```

CVF-75 INFO

- **Category** Procedural
- **Source** math_cmp.cairo

Description If this is a constant affecting the range check builtin, it should be in some globally imported file

Client Comment *Currently, we only use it in math_cmp, so I don't think there is a reason to move it right now.*

```
const RC_BOUND = %[ 2**128 %]
```

CVF-76 INFO

- **Category** Suboptimal
- **Source** math_cmp.cairo

Description This function is similar to 'assert_nn'.

Recommendation Consider deduplicating the code.

Client Comment *Both of them are important. you can not use assert_nn in is_nn, and assert_nn is simpler.*

```
func is_nn{range_check_ptr}(a) -> (res):
```

CVF-77 INFO

- **Category** Suboptimal
- **Source** math_cmp.cairo

Description this code part is redundant since this case can be handled by 'need_felt_comparison'

Client Comment *Even though the flow where you don't need felt comparison can be handled in the same way as the flow where you do need it (need_felt_comparison), we handle it specially because we can handle it more efficiently.*

```
%{ memory[ap] = 0 if 0 <= ((-ids.a - 1) % PRIME) < range_check_builtin.bound else 1 %}
jmp need_felt_comparison if [ap] != 0; ap++
assert [range_check_ptr] = (-a) - 1
let range_check_ptr = range_check_ptr + 1
return (res=0)
```

CVF-78 FIXED

- **Category** Documentation
- **Source** math_cmp.cairo

Description This comment is inaccurate, Actually, the function checks that: 0 <= a < RANGE_CHECK_BOUND and a <= b < a + RANGE_CHECK_BOUND

```
# Returns 1 of 0 <= a <= b < RANGE_CHECK_BOUND.
# Returns 0 otherwise.
```

CVF-79 INFO

- **Category** Suboptimal
- **Source** math_cmp.cairo

Description This function is similar to 'assert_nn_le'. Consider deduplicating the code.

Client Comment *Both of them are important. you can not use assert_nn_le in is_nn_le, and assert_nn_le is simpler.*

```
func is_nn_le{range_check_ptr}(a, b) -> (res):
```

CVF-80 INFO

- **Category** Overflow/Underflow
- **Source** execute_limit_order.cairo

Description Consider adding an assert that this value is positive.

Client Comment *This is not considered a bug. If the user's order allows the operator to take more fees than the gained collateral it's the user's problem*

```
assert collateral_delta = actual_collateral - actual_fee
```

CVF-81 FIXED

- **Category** Procedural
- **Source** program_output.cairo

Description There seems to be ‘get_label_location‘ function for the same purpose

```
callback=asset_config_hash_serialize + __pc__ - ret_pc_label)
```

```
callback=modification_serialize + __pc__ - ret_pc_label)
```

```
callback=forced_action_serialize + __pc__ - ret_pc_label)
```

CVF-82 INFO

- **Category** Readability
- **Source** conditional_transfer.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func execute_conditional_transfer(
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, batch_config : BatchConfig*,
           ↪ outputs : PerpetualOutputs*,
        tx : ConditionalTransfer*) -> (
        pedersen_ptr : HashBuiltin*, range_check_ptr, ecdsa_ptr :
           ↪ SignatureBuiltin*,
        carried_state : CarriedState*, outputs : PerpetualOutputs*):
```

CVF-83 INFO

- **Category** Readability
- **Source** update_position.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func is_asset_id_tradable(
        range_check_ptr, synthetic_asset_id, synthetic_delta,
        global_funding_indices : FundingIndicesInfo*, oracle_prices
           ↪ : OraclePrices*) -> (
        range_check_ptr, return_code):
```

```
func update_position(
        range_check_ptr, position : Position*, request_public_key,
           ↪ collateral_delta,
        synthetic_asset_id, synthetic_delta, global_funding_indices
           ↪ : FundingIndicesInfo*,
        oracle_prices : OraclePrices*, general_config :
           ↪ GeneralConfig*) -> (
        range_check_ptr, updated_position : Position*,
           ↪ funded_position : Position*, return_code):
```

```
func update_position_in_dict(
        range_check_ptr, positions_dict : DictAccess*, position_id,
           ↪ request_public_key,
        collateral_delta, synthetic_asset_id, synthetic_delta,
        global_funding_indices : FundingIndicesInfo*, oracle_prices
           ↪ : OraclePrices*,
        general_config : GeneralConfig*) -> (
        range_check_ptr, positions_dict : DictAccess*,
           ↪ funded_position : Position*,
        updated_position : Position*, return_code):
```

CVF-84 INFO

- **Category** Suboptimal
- **Source** update_position.cairo

Description The regular search function makes a number of range checks to find an element. In order to check if the element is present in an N-element array, at most N constraints is needed even for an unsorted array.

Client Comment *Will be added to a future version of cairo. We will then update the perpetual code to use that version.*

```
let (_, success) = search_sorted{range_check_ptr=range_check_ptr}(
```

```
let (_, success) = search_sorted{range_check_ptr=range_check_ptr}(
```

CVF-85 FIXED

- **Category** Documentation
- **Source** update_position.cairo

Description The dict manager functionality is not documented.

```
%{ ids.initial_position = __dict_manager.get_dict(ids.positions_dict)[ids.position_id] %}
```

CVF-86 FIXED

- **Category** Documentation
- **Source** find_element.cairo

Description It might be helpful to know which element out of many is returned in practice: first, last, medium, etc.

```
#   search for it.
```

CVF-87 INFO

- **Category** Suboptimal
- **Source** position.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func position_add_collateral(range_check_ptr, position : Position*,
  ↪ delta, public_key) -> (
        range_check_ptr, position : Position*, return_code):
```

```
func position_get_asset_balance(range_check_ptr, position : Position
  ↪ *, asset_id) -> (
        range_check_ptr, balance):
```

CVF-88 INFO

- **Category** Bad naming
- **Source** hash.cairo

Recommendation Consider renaming "pedersen_ptr" into "hash_ptr" to make "hash2" invocations more convenient.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func position_hash_assets{pedersen_ptr : HashBuiltin*}(
```

```
func position_hash{pedersen_ptr : HashBuiltin*}(position : Position
  ↪ *) -> (position_hash):
```

```
func hash_position_updates_inner{pedersen_ptr : HashBuiltin*}(
```

```
func hash_position_updates{pedersen_ptr : HashBuiltin*}(update_ptr :
  ↪  DictAccess*, n_updates) -> (
```

CVF-89 INFO

- **Category** Procedural
- **Source** hash.cairo

Description This function implements its own construction of a variable-input-length hash function based on a fixed-length compression function 'hash2'. This is error prone as all such constructions should be domain-separated to avoid collisions between.

Recommendation Consider extracting the data to be hashed to a single array and hash it using only a predefined set of VIL hash functions from the common library.

Client Comment *We can not make this change because it will change the hashes in the position tree.*

```
func position_hash{pedersen_ptr : HashBuiltin*}(position : Position *) -> (position_hash):
```

CVF-90 FIXED

- **Category** Readability
- **Source** hash.cairo

Description This should be replaced with and "alloc()" call.

```
%{ ids.hashed_updates_ptr = segments.add() %}
```

CVF-91 INFO

- **Category** Readability
- **Source** funding.cairo

Recommendation Consider using implicit arguments to improve readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func apply_funding_inner(
        range_check_ptr, assets_before : PositionAsset*, n_assets,
        global_funding_indices : FundingIndicesInfo*,
           ↪ current_collateral_fxp,
        assets_after : PositionAsset*) -> (range_check_ptr,
           ↪ collateral_fxp):
```

```
func position_apply_funding(
        range_check_ptr, position : Position*,
           ↪ global_funding_indices : FundingIndicesInfo*) -> (
        range_check_ptr, position : Position*):
```

CVF-92 FIXED

- **Category** Readability
- **Source** funding.cairo

Recommendation Consider using a high-level "if" statement to improve readability.

```
jmp body if n_assets != 0
```

```
body:
```

CVF-93 INFO

- **Category** Unclear behavior
- **Source** funding.cairo

Description Current timestamp is not taken into account, is it OK?

Client Comment *Yes, the funding_timestamp field is used in order to not keep the entire funding indices array in the position (The place we need this optimization is where we serialize the position changes).*

```
funding_timestamp=global_funding_indices.funding_timestamp)
```

CVF-94 INFO

- **Category** Readability
- **Source** validate_state_transition.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func check_valid_transition(
        range_check_ptr, updated_position : Position*,
           ↪ initial_position : Position*,
        oracle_prices : OraclePrices*, general_config :
           ↪ GeneralConfig*) -> (
        range_check_ptr, return_code):
```

CVF-95 INFO

- **Category** Readability
- **Source** status.cairo

Recommendation Consider using implicit arguments for readability

Client Comment *This change is big, we will consider to apply in future versions*

```
func position_get_status_inner(
        range_check_ptr, assets : PositionAsset*, n_assets,
           ↪ oracle_prices : OraclePrices*,
        general_config : GeneralConfig*, total_value_rep,
           ↪ total_risk_rep) -> (
        range_check_ptr, total_value_rep, total_risk_rep):
```

```
func position_get_status(
        range_check_ptr, position : Position*, oracle_prices :
           ↪ OraclePrices*,
        general_config : GeneralConfig*) -> (
        range_check_ptr, total_value_rep, total_risk_rep,
           ↪ return_code):
```

CVF-96 FIXED

- **Category** Readability
- **Source** status.cairo

Recommendation Consider using a high-level "if" statement for readability.

```
jmp body if n_assets != 0
```

```
body:
```

CVF-97 INFO

- **Category** Unclear behavior
- **Source** status.cairo

Description What if this will happen for a position due to funding payments? Will this position be locked?

Client Comment *Yes. Firstly, applying funding to a position doesn't change it in any way. We should've changed every position during the funding tick but because it is inefficient we have this caching mechanic. Secondly, Because we don't expect a position to have out of bounds TV/TR, we give ourselves the freedom to define such position as "frozen" (meaning that it can't be changed in any way)*

```
if res == 0:
    return (
        range_check_ptr=range_check_ptr,
        total_value_rep=0,
        total_risk_rep=0,
        return_code=PerpetualErrorCode.OUT_OF_RANGE_TOTAL_VALUE)
end
```

CVF-98 INFO

- **Category** Suboptimal
- **Source** status.cairo

Description The value "TR_UPPER_BOUND" is 2^128, so the "is_nn" function could be used instead of "is_le".

Client Comment *While I agree this isn't optimal, this is way more readable and maintainable (for the day we change TR_UPPER_BOUND)*

```
107 let (res) = is_le{range_check_ptr=range_check_ptr}(total_risk_rep,
      ↪ TR_UPPER_BOUND_REP - 1)
```

CVF-99 INFO

- **Category** Unclear behavior
- **Source** serialize_change.cairo

Recommendation Consider checking this assumption via a static assert.

Client Comment *There's no easy way doing this currently.*

```
13 #   ASSET_ID_UPPER_BOUND * (BALANCE_UPPER_BOUND -
     ↪ BALANCE_LOWER_BOUND) < PRIME.
```

CVF-100 INFO

- **Category** Readability
- **Source** serialize_change.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions*

```
31 func serialize_position_change_inner(
           range_check_ptr, output_ptr : felt*, n_prev_position_assets,
           prev_position_assets : PositionAsset*, n_new_position_assets
              ↪ ,
           new_position_assets : PositionAsset*) -> (range_check_ptr,
              ↪ output_ptr : felt*):
```

```
91 func serialize_position_change(range_check_ptr, output_ptr : felt*,
     ↪ dict_access : DictAccess*) -> (
           range_check_ptr, output_ptr : felt*):
```

CVF-101 INFO

- **Category** Suboptimal
- **Source** serialize_change.cairo

Description These two code chunks basically do the same. Consider refactoring the code to avoid duplication.

Client Comment *This is over complication. There is no need to create another function for this.*

```
        with output_ptr:
            serialize_asset(asset_id=new_asset_id, balance=
               ↪ new_position_assets.balance)
        end
```

```
with output_ptr:
    serialize_asset(asset_id=new_position_assets.asset_id, balance=
       ↪ new_position_assets.balance)
end
```

CVF-102 INFO

- **Category** Readability
- **Source** check_smaller_holdings.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func check_smaller_in_synthetic_holdings_inner(
        range_check_ptr, n_updated_position_assets,
           ↪ updated_position_assets : PositionAsset*,
        n_initial_position_assets, initial_position_assets :
           ↪ PositionAsset*) -> (
        range_check_ptr, return_code):
```

```
func check_smaller_in_synthetic_holdings(
        range_check_ptr, updated_position : Position*,
           ↪ initial_position : Position*) -> (
        range_check_ptr, return_code):
```

CVF-103 FIXED

- **Category** Documentation
- **Source** check_smaller_holdings.cairo

Description The check passes when one position is zero and the other is not. Strictly speaking, in this case the position signs are different.

Recommendation Consider rephrasing like: Check that updated_balance and initial_balance have the same sign or one of the balances is zero.

```
# Check that updated_balance and initial_balance have the same sign.
```

CVF-104 INFO

- **Category** Readability
- **Source** add_asset.cairo

Recommendation Consider using implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func get_old_asset(
        range_check_ptr, asset_ptr : PositionAsset*, asset_found,
        global_funding_indices : FundingIndicesInfo*, asset_id) -> (
        range_check_ptr, balance, funding_index, return_code):
```

```
func add_asset_inner(
        range_check_ptr, n_assets, assets_ptr : PositionAsset*,
           ↪ res_ptr : PositionAsset*,
        global_funding_indices : FundingIndicesInfo*, asset_id,
           ↪ delta) -> (
        range_check_ptr, end_ptr : PositionAsset*, return_code):
```

```
func position_add_asset(
        range_check_ptr, position : Position*,
           ↪ global_funding_indices : FundingIndicesInfo*,
        asset_id, delta, public_key) -> (range_check_ptr, position :
           ↪  Position*, return_code):
```

CVF-105 FIXED

- **Category** Suboptimal
- **Source** add_asset.cairo

Description It seems that there could be at most one asset with given "asset_id" in the array. If so, performing the second search just to find the right_start_ptr is redundant. Simply assign right_start_ptr = left_end_ptr + PositionAsset.SIZE in case the left_start_ptr points to the position whose asset is "asset_id", and assign right_start_ptr = left_end_ptr otherwise.

```
let (right_start_ptr : PositionAsset*) = search_sorted_lower{
   ↪ range_check_ptr=range_check_ptr}(
    array_ptr=assets_ptr, elm_size=PositionAsset.SIZE, n_elms=
       ↪ n_assets, key=asset_id + 1)
```

CVF-106 FIXED

- **Category** Unclear behavior
- **Source** add_asset.cairo

Description Is this equivalent to an "alloc" call?

```
local res_assets_ptr : PositionAsset*
%{ ids.res_assets_ptr = segments.add() %}
```

CVF-107 INFO

- **Category** Readability
- **Source** data_availability.cairo

Description These functions should use implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
17 func output_changed_positions(
           range_check_ptr, output_ptr : felt*, squashed_dict :
              ↪ DictAccess*, n_entries) -> (
           range_check_ptr, output_ptr : felt*):
```

```
35 func output_availability_data(
           range_check_ptr, output_ptr : felt*, squashed_state :
              ↪ SquashedCarriedState*,
           perpetual_outputs_start : PerpetualOutputs*,
              ↪ perpetual_outputs_end : PerpetualOutputs*) -> (
           range_check_ptr, output_ptr : felt*):
```

CVF-108 INFO

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

Description Is it possible to derive the public key from a message and a signature? If so, then public key could be make a result rather than an argument of this function.

Client Comment *We don't have a usecase where we don't have the public key, therefore it is easier to do it this way.*

```
8 message, public_key, signature_r, signature_s):
```

CVF-109 INFO

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

Description Why there is no "v" (sign) value?

Client Comment *We use the curve point as a public key, therefore we dont need a sign value.*

```
8 message, public_key, signature_r, signature_s):
```

CVF-110 INFO

- **Category** Suboptimal
- **Source** serialize.cairo

Description When serializing structures, this functions is often called several times. Consider implementing efficient version of this functions that serialize several words at once, i.e. 2, 3, 4, etc. This would make serialization more efficient.

Client Comment *We prefer readability over performance in this case. Further more, we plan to support function inlining in the future and then this code will be more efficient than a function that serializes several words.*

```
func serialize_word{output_ptr : felt*}(word):
```

CVF-111 INFO

- **Category** Unclear behavior
- **Source** oracle_price.cairo

Description The "%[...%]" syntax is not documented. It is unclear what does it mean.

Client Comment *This syntax is deprecated in newer Cairo versions. What it basically means is that the compiler interprets the code inside the %[...%] as a python command that is expected to return an integer and the compiler changes the code as if that integer was written there.*

```
const TIMESTAMP_BOUND = %[2**32%]
```

CVF-112 INFO

- **Category** Readability
- **Source** oracle_price.cairo

Description These functions should use implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func check_price_signature(
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, hash_ptr :
           ↪ HashBuiltin*,
        time_bounds : TimeBounds*, asset_info : SyntheticAssetInfo*,
           ↪  median_price,
        collateral_resolution, sig : SignedOraclePrice*) -> (
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, hash_ptr :
           ↪ HashBuiltin*, is_le, is_ge):
```

```
func check_oracle_price_inner(
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, hash_ptr :
           ↪ HashBuiltin*,
        time_bounds : TimeBounds*, asset_info : SyntheticAssetInfo*,
           ↪  median_price,
        collateral_resolution, sig : SignedOraclePrice*, n_sigs,
           ↪ last_signer, n_le, n_ge) -> (
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, hash_ptr :
           ↪ HashBuiltin*, n_le, n_ge):
```

```
func check_oracle_price(
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, hash_ptr :
           ↪ HashBuiltin*,
        time_bounds : TimeBounds*, asset_oracle_price :
           ↪ AssetOraclePrice*,
        asset_info : SyntheticAssetInfo*, collateral_info :
           ↪ CollateralAssetInfo*) -> (
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, hash_ptr :
           ↪ HashBuiltin*):
```

```
func check_oracle_prices_inner(
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, hash_ptr :
           ↪ HashBuiltin*, n_oracle_prices,
        asset_oracle_prices : AssetOraclePrice*,
           ↪ n_synthetic_assets_info,
        synthetic_assets_info : SyntheticAssetInfo*, time_bounds :
           ↪ TimeBounds*,
        general_config : GeneralConfig*) -> (
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, hash_ptr :
           ↪ HashBuiltin*):
```

(286)

CVF-113 FIXED

- **Category** Documentation
- **Source** oracle_price.cairo

Description This assumes that the denominator is even.

Recommendation Consider explaining why this is true.

```
    # Add denominator/2 to round.
    let (internal_price, _) = unsigned_div_rem{range_check_ptr=
       ↪ range_check_ptr}(
numerator + denominator / 2, denominator)
```

CVF-114 INFO

- **Category** Suboptimal
- **Source** oracle_price.cairo

Description One of ‘is_le‘ and ‘is_ge‘ is redundant. Just return ‘median_comparison‘

Client Comment *This change is big, we will consider to apply in future versions*

```
    is_le=1,
    is_ge=1)
```

```
is_le=1 - is_ge,
is_ge=is_ge)
```

CVF-115 FIXED

- **Category** Documentation
- **Source** oracle_price.cairo

Description It is unclear what ‘valid‘ means. The code implies that the asset ids of oracle prices are the subset of synthetic asset ids.

```
# Checks that a list of AssetOraclePrice instances are valid with
   ↪ respect to a GeneralConfig and a
```

CVF-116 FIXED

- **Category** Readability
- **Source** objects.cairo

Description There seems to be ‘get_label_location‘ function for the same purpose

```
get_fp_and_pc()
let __pc__ = [fp + 1]
```

```
ret_pc_label:
```

```
    callback=funding_index_serialize + __pc__ - ret_pc_label)
```

CVF-117 INFO

- **Category** Readability
- **Source** objects.cairo

Description This looks like a hack. Why not accessing the necessary information by ‘data‘ input?

Client Comment *This trick reduces the amount of steps we have in the function. Instead of creating a new OraclePrices object, we get it from the arguments of the function as explained in new added comment.*

```
return (oracle_prices=cast(fp_val - 2 - OraclePrices.SIZE,
   ↪ OraclePrices*))
```

CVF-118 INFO

- **Category** Documentation
- **Source** perpetual_error_code.cairo

Description Namespaces are not described in the documentation.

Client Comment *We are working on documenting the namespace feature.*

```
namespace PerpetualErrorCode:
```

CVF-119 INFO

- **Category** Documentation
- **Source** perpetual_error_code.cairo

Description This trick is not described in the documentation.

Client Comment *There's no trick here. The only thing this does is that if the program failed the error code that made it fail will appear in a hint variable. This is for internal use in our tests to know the failure reason.*

```
# If not, the function will put the error code in a hint variable
   ↪ before exiting.
```

CVF-120 INFO

- **Category** Bad naming
- **Source** cairo_builtins.cairo

Description The structure does not contain the signature itself nor the result of its verification. Perhaps it should be named differently.

Client Comment *This change is big, we will consider to apply in future versions.*

```
# A representation of a SignatureBuiltin struct, specifying the
   ↪ signature builtin memory structure.
```

CVF-121 INFO

- **Category** Documentation
- **Source** cairo_builtins.cairo

Description This builtin is not documented.

Client Comment *This builtin does not exist in the current version of Cairo.*

```
struct CheckpointsBuiltin:
```

CVF-122 INFO

- **Category** Documentation
- **Source** cairo_builtins.cairo

Description It is unclear what ‘required‘ is.

Client Comment *This builtin does not exist in the current version of Cairo.*

```
17 member required_pc : felt
   member required_fp : felt
```

CVF-123 FIXED

- **Category** Documentation
- **Source** hash_state.cairo

Description Semantics of the output value is unclear

```
16 func hash_init() -> (hash_state_ptr : HashState*):
```

```
73 func hash_update{hash_ptr : HashBuiltin*}(
```

```
87 func hash_update_single{hash_ptr : HashBuiltin*}(hash_state_ptr : HashState*, item) -> (
```

CVF-124 INFO

- **Category** Documentation
- **Source** hash_state.cairo

Description There is no link for it

Client Comment *hash_update is in the same file, I don’t see a reason to include a link.*

```
25 # A helper function for 'hash_update', see its documentation.
```

CVF-125 FIXED

- **Category** Documentation
- **Source** hash_state.cairo

Description This comment is confusing. The function actually computes H(... H(H(hash,data[0]),data[1]),..),data[n-1])

```
# Computes the hash of an array of items, not including its length.
```

CVF-126 INFO

- **Category** Bad naming
- **Source** hash_state.cairo

Description Variable names are confusing. Consider using ‘input_state‘ and ‘output_state‘

Client Comment *See CVF-153.*

```
data_ptr : felt*, data_length : felt, hash : felt) -> (hash : felt):
```

CVF-127 INFO

- **Category** Readability
- **Source** hash_state.cairo

Description Such code practice is highly discouraged

Recommendation Consider using different names

Client Comment *See CVF-153.*

```
return (hash=hash)
```

CVF-128 INFO

- **Category** Suboptimal
- **Source** hash_state.cairo

Description This variable is redundant as it is used only once.

Client Comment *We use a variable to avoid computing data_last_ptr in each iteration. The code is more efficient that way. Added a comment.*

```
local data_last_ptr : felt* = data_ptr + data_length - 1
```

CVF-129 INFO

- **Category** Suboptimal
- **Source** hash_state.cairo

Description The only loop variable needed is the current data index. All other variables could be easily derived from this value.

Client Comment *This doesn't make the code more optimal, see CVF-119.*

```
struct LoopLocals:
    member data_ptr : felt*
    member hash_ptr : HashBuiltin*
    member cur_hash : felt
end
```

CVF-130 INFO

- **Category** Suboptimal
- **Source** hash_state.cairo

Description These two assertions can be merged and thus make the variable 'cur_hash' needless.

Client Comment *This doesn't make the code more optimal, see CVF-119.*

```
prev_locals.hash_ptr.x = prev_locals.cur_hash
```

```
next_locals.cur_hash = prev_locals.hash_ptr.result; ap++
```

CVF-131 INFO

- **Category** Suboptimal
- **Source** hash_chain.cairo

Description This structure could be simplified. Actually, the only loop variable required is the current data index (starting from "data_length - 1" and decreasing till zero). All other variables could be easily derived from this only variable: loop_data_ptr = data_ptr + data_index + 1 loop_hash_ptr = hash_ptr + data_index * HashBuiltin.SIZE cur_hash = hash_ptr + cast(data_index * HashBuiltin.SIZE + HashBuiltin.SIZE, HashBuiltin*).result In such approach the very first iteration should be treated differently, as it uses the last data element instead of the previous hash result.

Client Comment *This doesn't make the code more optimal, see CVF-119.*

```
struct LoopLocals:
    member data_ptr : felt*
    member hash_ptr : HashBuiltin*
    member cur_hash : felt
end
```

CVF-132 FIXED

- **Category** Bad naming
- **Source** hash_chain.cairo

Description Should be 'data_length_ptr'

```
let data_length = ap
```

CVF-133 INFO

- **Category** Readability
- **Source** hash_chain.cairo

Description For readability it is better to allocate a single struct per loop that contains all variables for given iteration, and increase ap just once by its size.

Client Comment *This code is deprecated and hash_state should be used instead. We've kept it because there are other places that still use it and didn't got migrated to hash_state yet (This should be a task for Lior's team).*

```
[new_data] = [new_data_ptr]; ap++
```

```
[new_data] = current_hash.x; ap++
```

```
next_frame.data_ptr = new_data_ptr; ap++
next_frame.hash_ptr = curr_frame.hash_ptr + HashBuiltin.SIZE; ap++
next_frame.cur_hash = current_hash.result; ap++
```

CVF-134 FIXED

- **Category** Documentation
- **Source** general_config_hash.cairo

Description Should be "this asset".

```
# A synthetic asset entry contaning tis asset id and its config's hash.
␣␣␣␣␣␣
```

CVF-135 FIXED

- **Category** Unclear behavior
- **Source** general_config_hash.cairo

Description This assert doesn't guarantee that all the fields were hashed, because some fields are taken from the structs referred via pointer and some fields are actually not hashed.

Recommendation Consider also asserting the number of fields in the referred structs: "CollateralAssetInfo" and "FeePositionInfo". Also, consider explaining in a comment, why synthetic assets information is not hashed.

```
static_assert GeneralConfig.SIZE == 8
```

CVF-136 FIXED

- **Category** Unclear behavior
- **Source** general_config_hash.cairo

Description This code seems to allocate a chunk of memory. Is there a more elegant way to do this?

```
%{
    ids.asset_configs = asset_configs = segments.add()
    segments.finalize(
        asset_configs.segment_index,
        ids.general_config_ptr.n_synthetic_assets_info * ids.
           ↪ AssetConfigHashEntry.SIZE
    )
%}
```

CVF-137 INFO

- **Category** Unclear behavior
- **Source** constants.cairo

Description The "%[...%]" syntax is not documented. It is unclear what does it mean.

Client Comment *This syntax is deprecated in newer Cairo versions. What it basically means is that the compiler interprets the code inside the %[...%] as a python command that is expected to return an integer and the compiler changes the code as if that integer was written there.*

```
 8 const ASSET_ID_UPPER_BOUND = %[2**120%]
```

```
11 const BALANCE_UPPER_BOUND = %[2**63%]
```

```
14 const TOTAL_VALUE_UPPER_BOUND = %[2**63%]
   const TOTAL_VALUE_LOWER_BOUND = -%[2**63%]
```

```
17 const TOTAL_RISK_UPPER_BOUND = %[2**64%]
```

```
19 const N_ASSETS_UPPER_BOUND = %[2**16%]
20 const POSITION_MAX_SUPPORTED_N_ASSETS = %[2**6%]
```

```
23 const FXP_32_ONE = %[2**32%]
```

```
26 const EXTERNAL_PRICE_FIXED_POINT_UNIT = %[10**18%]
```

```
28 const ORACLE_PRICE_QUORUM_LOWER_BOUND = %[1%]
   const ORACLE_PRICE_QUORUM_UPPER_BOUND = %[2**32%]
```

```
31 const POSITION_ID_UPPER_BOUND = %[2**64%]
   const ORDER_ID_UPPER_BOUND = %[2**64%]
```

```
34 const FUNDING_INDEX_UPPER_BOUND = %[2**63%]
   const FUNDING_INDEX_LOWER_BOUND = -%[2**63%]
```

```
38 const RISK_FACTOR_LOWER_BOUND = %[1%]
```

```
43 const PRICE_UPPER_BOUND = %[2**64%]
```

```
45 const EXTERNAL_PRICE_UPPER_BOUND = %[2**120%]
```

(47, 52)

CVF-138 FIXED

- **Category** Unclear behavior
- **Source** merkle_multi_update.cairo

Description This relies on undocumented compiler behavior that is subject to change.

Recommendation Consider refactoring the code to make it more predictable.

```
# Locals 0 and 1 are taken by non deterministic jumps.
```

CVF-139 FIXED

- **Category** Suboptimal
- **Source** merkle_multi_update.cairo

Description These variables are used only once and can be eliminated

```
local_left_index = index * 2; ap++
```

```
tempvar height_minus_1 = height - 1
```

CVF-140 FIXED

- **Category** Readability
- **Source** merkle_multi_update.cairo

Description In the former case the expressions "height - 1" and "index * 2" are inlined, while in the forder case the very same expression are precomputed and accessed by references.

Recommendation Consider using consistent approach in both cases.

```
merkle_multi_update_inner(
    height=height - 1, prev_root=hash0.x, new_root=hash1.x, index=
      ↪ index * 2)
```

```
merkle_multi_update_inner(
    height=height_minus_1, prev_root=hash0.y, new_root=hash1.y,
      ↪ index=local_left_index + 1)
```

CVF-141 INFO

- **Category** Procedural
- **Source** dict.cairo

Description We did not audit this witness generation file.

```
from starkware.cairo.common.dict import DictManager
```

CVF-142 FIXED

- **Category** Unclear behavior
- **Source** dict.cairo

Description It is unclear what "initial_dict" is.

```
memory[ap] = __dict_manager.new_dict(segments, initial_dict)
del initial_dict
```

CVF-143 INFO

- **Category** Documentation
- **Source** dict.cairo

Description The semantics of the implicit argument is unclear.

Client Comment *The documentation about implicit arguent (not as part of a builtin) will be added to a future version of cairo.*

```
func dict_read{dict_ptr : DictAccess*}(key : felt) -> (value : felt)
   ↪ :
```

```
func dict_write{dict_ptr : DictAccess*}(key : felt, new_value : felt
   ↪ ):
```

```
func dict_update{dict_ptr : DictAccess*}(key : felt, prev_value :
   ↪ felt, new_value : felt):
```

```
func dict_squash{range_check_ptr}(
```

CVF-144 FIXED

- **Category** Unclear behavior
- **Source** dict.cairo

Description Why do we have ‘assert‘ in one function and do not in the other?

Recommendation Consider using the same approach.

```
assert dict_ptr.key = key
assert dict_ptr.new_value = new_value
```

```
dict_ptr.key = key
```

```
dict_ptr.new_value = new_value
```

CVF-145 INFO

- **Category** Procedural
- **Source** squash_dict.cairo

Description Passing the array length is more natural than the second pointer

Client Comment *This change is big and of relatively low priority.*

```
dict_accesses : DictAccess*, dict_accesses_end : DictAccess*,
```

CVF-146 FIXED

- **Category** Readability
- **Source** squash_dict.cairo

Description Defining local variables like this is hard ot read and error-prone.

```
let first_key = [fp + 1]
let big_keys = [fp + 2]
ap += 2
```

CVF-147 INFO

- **Category** Bad naming
- **Source** squash_dict.cairo

Description Here four different variables have the same name.

Recommendation Consider renaming for readability.

Client Comment *See CVF-153.*

```
    squashed_dict=squashed_dict,
```

```
return (squashed_dict=squashed_dict)
```

CVF-148 INFO

- **Category** Unclear behavior
- **Source** squash_dict.cairo

Description The function returns the address of the first allocated element after ‘squashed_dict‘, which is likely useless.

Recommendation Consider dropping this returned value.

Client Comment *It is in use.*

```
# squashed_dict - end pointer to squashed_dict.
```

CVF-149 INFO

- **Category** Readability
- **Source** squash_dict.cairo

Description Passing the total dict length instead of dict_accesses_end_minus1 would make the code more readable

Client Comment *This might be fixed in a future cairo version.*

```
range_check_ptr, dict_accesses : DictAccess*,
   ↪ dict_accesses_end_minus1 : felt*, key,
remaining_accesses, squashed_dict : DictAccess*, big_keys) -> (
```

CVF-150 INFO

- **Category** Suboptimal
- **Source** squash_dict.cairo

Description This variable is redundant as it is always a linear function of the previous one.

Client Comment *This doesn't make the code more efficient, see CVF-119.*

```
member index_delta : felt
```

CVF-151 FIXED

- **Category** Documentation
- **Source** squash_dict.cairo

Description It is unclear from the code what the last allocated variables are.

Recommendation Consider adding comments

```
let prev_loop_locals = cast(ap - LoopLocals.SIZE, LoopLocals*)
let loop_temps = cast(ap, LoopTemps*)
let loop_locals = cast(ap + LoopTemps.SIZE, LoopLocals*)
```

CVF-152 INFO

- **Category** Readability
- **Source** squash_dict.cairo

Description The values are allocated in the order different from their order in the struct declaration.

Recommendation Consider using the right order.

Client Comment *The second line depends on the first one, switching them will be less efficient.*

```
loop_locals.access_ptr = prev_loop_locals.access_ptr + loop_temps.
   ↪ ptr_delta; ap++
```

```
loop_locals.value = access.new_value; ap++
```

CVF-153 INFO

- **Category** Readability
- **Source** squash_dict.cairo

Description Using the range check pointer evolution to track the number of accesses makes code less readable.

Client Comment *We do that for optimization. Added a comment .*

```
185 tempvar n_used_accesses = last_loop_locals.range_check_ptr -
    ↪ range_check_ptr
```

CVF-154 FIXED

- **Category** Documentation
- **Source** squash_dict.cairo

Description There should be a comment that ap points to the address of the next access in the original list.

```
200 let next_key = [ap]
```

CVF-155 INFO

- **Category** Readability
- **Source** squash_dict.cairo

Description Code of form ‘a=a‘ looks weird.

Recommendation Consider using distinct variable names

Client Comment *The syntax a=a is rather natural in cairo: We use the syntax a=a in cairo for reference rebinding because the memory in cairo is immutable. This is equivalent to variable assignments in other languages.*

```
210 tempvar dict_accesses = dict_accesses
```

```
212 tempvar next_key = next_key
    tempvar remaining_accesses = remaining_accesses
```

```
217 tempvar dict_accesses = dict_accesses
```

```
219 tempvar next_key = next_key
```

CVF-156 INFO

- **Category** Suboptimal
- **Source** memcpy.cairo

Description This function copies one word per iteration, which is suboptimal.

Recommendation Consider copying several words, such as 8 or 16 per iteration. In case the length is not a factor of the number of words copied per iteration, the output could either be padded or a separate tail loop could be used to copy the remainder.

Client Comment *Will be added to a future version of cairo. We will then update the perpetual code to use that version.*

```
func memcpy(dst : felt*, src : felt*, len):
```

CVF-157 INFO

- **Category** Suboptimal
- **Source** memcpy.cairo

Description Using a struct is redundant: a mere counter suffices.

Client Comment *This doesn't make the code more efficient, see CVF-119.*

```
let frame = cast(ap - LoopFrame.SIZE, LoopFrame*)
```

CVF-158 FIXED

- **Category** Procedural
- **Source** program_input.cairo

Description CarriedState object is not used in this file

```
from services.perpetual.cairo.state.state import CarriedState,
   ↪ SharedState
```

CVF-159 INFO

- **Category** Procedural
- **Source** forced.cairo

Description There is ‘get_label_location‘ method for this

Client Comment *We’re using the fp and not the pc here. For more explanation why, see the comment on CVF-110.*

```
func forced_withdrawal_action_new(public_key, position_id, amount)
    ↪ -> (
```

```
func forced_trade_action_new(
```

CVF-160 INFO

- **Category** Suboptimal
- **Source** forced.cairo

Recommendation Consider replacing the ”if” statement with an assert like this: assert forced_type = ForcedActionType.FORCED_TRADE This would make the false ”asssert” and ”jmp” statements redundant.

Client Comment *This way it is consistant with other parts of our code, and will be easier to add more cases that way.*

```
if forced_type == ForcedActionType.FORCED_TRADE:
```

```
assert 1 = 0
jmp rel 0
```

CVF-161 INFO

- **Category** Suboptimal
- **Source** hash.cairo

Description These two operations can be merged in a more readable ‘return result=(hash_ptr-HashBuiltin.SIZE).result

Client Comment *We chose to stay with the current implementation.*

```
let result = hash_ptr.result
```

```
return (result=result)
```

CVF-162 INFO

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

Description The argument is a message encoding rather than a message hash.

Client Comment *The argument is a message hash, we extract the order_id from the first 64 bits of the hash.*

```
# Extracts the order_id from the message_hash.
```

CVF-163 INFO

- **Category** Suboptimal
- **Source** order.cairo

Description This assumption is redundant if the previous one holds

Client Comment *We chose to keep it.*

```
#   0 <= message_hash < SIGNED_MESSAGE_BOUND.
```

CVF-164 INFO

- **Category** Readability
- **Source** order.cairo

Recommendation Consider using an implicit argument for the range check pointer to improve readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func extract_order_id(range_check_ptr, message_hash) -> (range_check_ptr, order_id):
```

```
func update_order_fulfillment(
        range_check_ptr, orders_dict : DictAccess*, message_hash, update_amount, full_amount) -> (
        range_check_ptr, orders_dict : DictAccess*):
```

```
func validate_order_and_update_fulfillment(
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, orders_dict : DictAccess*, message_hash,
        order : OrderBase*, min_expiration_timestamp, update_amount, full_amount) -> (
        range_check_ptr, ecdsa_ptr : SignatureBuiltin*, orders_dict : DictAccess*):
```

CVF-165 INFO

- **Category** Bad naming
- **Source** signature_message_hashes.cairo

Description Renaming the "pedersen_ptr" implicit argument into "hash_ptr" would allow makeing the code cleaner and easier to read.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func limit_order_hash{pedersen_ptr : HashBuiltin*}(limit_order : ExchangeLimitOrder*) -> (
```

```
func transfer_hash{pedersen_ptr : HashBuiltin*}(transfer : ExchangeTransfer*, condition : felt) -> (
```

CVF-166 INFO

- **Category** Bad naming
- **Source** signature_message_hashes.cairo

Description The function should be called ‘exchange_limit_order_hash‘

Client Comment *This change affects another project so we decided to leave it like that.*

```
37 func limit_order_hash{pedersen_ptr : HashBuiltin*}(limit_order :
    ↪ ExchangeLimitOrder*) -> (
```

CVF-167 FIXED

- **Category** Procedural
- **Source** signature_message_hashes.cairo

Description Constants defined inside functions are hard to find.

Recommendation Consider gathering all the constants in one place.

```
51 const LIMIT_ORDER_WITH_FEES = 3
```

```
102 const TRANSFER_ORDER_TYPE = 4
    const CONDITIONAL_TRANSFER_ORDER_TYPE = 5
```

CVF-168 FIXED

- **Category** Suboptimal
- **Source** signature_message_hashes.cairo

Description This variable is used only once and can be eliminated

```
56 let expiration_timestamp = limit_order.base.expiration_timestamp
```

```
58     expiration_timestamp
```

CVF-169 INFO

- **Category** Unclear behavior
- **Source** signature_message_hashes.cairo

Description The "%[...%]" syntax is not documented. It is unclear what does it mean.

Client Comment *This syntax is deprecated in newer Cairo versions. What it basically means is that the compiler interprets the code inside the %[...%] as a python command that is expected to return an integer and the compiler changes the code as if that integer was written there.*

```
59  let packed_message1 = packed_message1 * %[2**17%]  # Padding.
```

```
132 let packed_message1 = packed_message1 * %[2**81%]  # Padding.
```

CVF-170 INFO

- **Category** Bad naming
- **Source** signature_message_hashes.cairo

Description These names look inconsistent with each other.

Recommendation Consider using names like: "fee_vault_id", "fee_asset_id", and "fee_max_amount".

Client Comment *This change affects another project so we decided to leave it like that.*

```
74  member src_fee_vault_id : felt
    member asset_id_fee : felt
    member max_amount_fee : felt
```

CVF-171 FIXED

- **Category** Readability
- **Source** signature_message_hashes.cairo

Description This two-layer hash description is confusing.

Recommendation Consider describing like this: The hash is defined as h(h(h(h(w1, w2), w3), w4, w5) for a normal transfer, where h is Starkware's Pedersen hash function and: w1 = asset_id w2 = asset_id_fee w3 = receiver_public_key w4 = sender_vault_id (64 bit) || receiver_vault_id (64 bit) || src_fee_vault_id (64 bit) || nonce (32 bit) w5 = 0x4 (15 bit) || amount (64 bit) || max_amount_fee (64 bit) || expiration_timestamp (32 bit) || 0 (81 bit)

```
# The hash is defined as h(h(w1, w2), w3) for a normal transfer,
   ↪ where h is Starkware's␣Pedersen
#␣hash␣function␣and:
#␣␣␣w1␣=␣h(h(asset_id,␣asset_id_fee),␣receiver_public_key)
#␣␣␣w2␣=␣sender_vault_id␣(64␣bit)␣||␣receiver_vault_id␣(64␣bit)
#␣␣␣␣␣␣␣||␣src_fee_vault_id␣(64␣bit)␣||␣nonce␣(32␣bit)
#␣␣␣w3␣=␣0x4␣(15␣bit)␣||␣amount␣(64␣bit)␣||␣max_amount_fee␣(64␣bit)␣
   ↪ ||␣expiration_timestamp␣(32␣bit)
#␣␣␣␣␣␣␣||␣0␣(81␣bit)
␣␣␣␣␣␣
```

CVF-172 INFO

- **Category** Suboptimal
- **Source** signature_message_hashes.cairo

Description The brackets are redundant here.

Client Comment *The brackets allow us to split the line into two line and avoiding a very long line.*

```
let packed_message1 = (
    packed_message1 * EXPIRATION_TIMESTAMP_UPPER_BOUND + transfer.
        ↪ base.expiration_timestamp)
```

CVF-173 INFO

- **Category** Readability
- **Source** validate_limit_order.cairo

Description This function should use implicit arguments for readability.

Client Comment *This change is big, we will consider to apply in future versions.*

```
func validate_limit_order_fairness(
        range_check_ptr, limit_order : LimitOrder*,
           ↪ actual_collateral, actual_synthetic,
        actual_fee) -> (range_check_ptr):
```

CVF-174 FIXED

- **Category** Documentation
- **Source** limit_order.cairo

Description All this logic is implemented elsewhere, so this comment is irrelevant to the actual code below.

```
# limit_order_hash:
# Computes the hash of a limit order.
#
# The hash is defined as h(h(h(h(w1, w2), w3), w4), w5) where h is
   ↪ the
# starkware pedersen function and w1,...w5 are as follows:
# w1= token_sell
# w2= token_buy
# w3= token_fee
# w4= amount_sell (64 bit) || amount_buy (64 bit) || amount_fee (64
   ↪ bit) || nonce (32 bit)
# w5= 0x3 (10 bit) || vault_fee_src (64 bit) || vault_sell (64 bit)
   ↪ || vault_buy (64 bit)
#    || expiration_timestamp (32 bit) || 0 (17 bit)
#
# Assumptions (bounds defined in services.perpetual.cairo.
   ↪ definitions.constants):
# amount_sell < AMOUNT_UPPER_BOUND
# amount_buy < AMOUNT_UPPER_BOUND
# amount_fee < AMOUNT_UPPER_BOUND
# nonce < NONCE_UPPER_BOUND
# position_id < POSITION_ID_UPPER_BOUND
# expiration_timestamp < EXPIRATION_TIMESTAMP_UPPER_BOUND.
```

CVF-175 INFO

- **Category** Bad naming
- **Source** limit_order.cairo

Description The name "hash_ptr" would be move conventional and would allow using functions from the standard library in a more convenient way.

Client Comment *The name "hash_ptr" is used only when we want to emphasize that another hash builtin can be used (for example, in hash2). In order to keep the code consistent, we've changed all occurences of hash_ptr into pedersen_ptr in the perpetual code as that application only uses pedersen .*

```
func limit_order_hash{pedersen_ptr : HashBuiltin*}(limit_order : LimitOrder*) -> (limit_order_hash):
```

CVF-176 INFO

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

Description A better name would be "Signature" as this structure doesn't have any fields related to an exchange order.

Client Comment *Aside from the signature, the struct contains nonce, public_key and expiration_timestamp.*

```
struct OrderBase:
```

CVF-177 INFO

- **Category** Unclear behavior
- **Source** constants.cairo

Description The "%[...%]" syntax is not documented. It is unclear, what does it mean.

Client Comment *This syntax is deprecated in newer Cairo versions. What it basically means is that the compiler interprets the code inside the %[...%] as a python command that is expected to return an integer and the compiler changes the code as if that integer was written there.*

```
const AMOUNT_UPPER_BOUND = %[2**64%]
const EXPIRATION_TIMESTAMP_UPPER_BOUND = %[2**32%]
const NONCE_UPPER_BOUND = %[2**32%]
const VAULT_ID_UPPER_BOUND = %[2**64%]
```

CVF-178 INFO

- **Category** Bad naming
- **Source** small_merkle_tree.cairo

Description The function rather updates the tree than just creates it.

Recommendation Consider renaming.

Client Comment *We don't use this function but we fix internally.*

```
49 func small_merkle_tree{hash_ptr : HashBuiltin*}(
```

CVF-179 INFO

- **Category** Unclear behavior
- **Source** small_merkle_tree.cairo

Description We didn't review this function that seems to do most of the work.

Client Comment *There is no need to review this function.*

```
85 ids.new_root, ids.prev_root, preimage = get_preimage_dictionary(
```

CVF-180 FIXED

- **Category** Documentation
- **Source** merkle_update.cairo

Recommendation Consider adding comments why the division never overflows for height <log p.

```
49 index=index / 2)
```

```
70 index=(index - 1) / 2)
```

CVF-181 INFO

- **Category** Unclear behavior
- **Source** dict_access.cairo

Description The semantics of the struct fields is unclear

Client Comment *Added documentation.*

```
struct DictAccess:
```

CVF-182 INFO

- **Category** Unclear behavior
- **Source** alloc.cairo

Description This Python API is not documented.

Client Comment *We expect users to call alloc() and not use this API.*

```
%{ memory[ap] = segments.add() %}
```

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