SHERLOCK

Security Review For Brevis

Collaborative Audit Prepared For:	**Brevis**
Lead Security Expert(s):	**cergyk** **sammy**
Date Audited:	**April 7 - May 19, 2025**
Final Commit:	**d5889f2**

Introduction

Brevis is a highly efficient ZK coprocessor that empowers smart contracts to read from the full historical on-chain data from all supported blockchains and run customizable computations in a completely trust-free way. With the power of trust-free historical data, Brevis enables exciting new use cases like data-driven DeFi, user retention and engagement features, trust-free active liquidity management, omnichain activity-based identity, and many more.

Scope

Repository: brevis-network/brevis-core

Audited Commit: 72fb4260101c59dce928f4cfa3c16547ecac991a

Final Commit: d5889f2067a32668e01899c1d34d056d217831b4

Files:

- block_syncer/src/blk_chunk_update.rs
- block_syncer/src/blocks/chunk_block.rs
- block_syncer/src/smt_anchor/smt_anchor_stark.rs
- block_syncer/src/smt_update/recursion_verify_with_ctl.rs
- block_syncer/src/smt_update/smt_update_stark.rs
- block_syncer/src/smt_update/util_sdk.rs
- block_syncer/src/smt_update/verify_with_ctl.rs
- block_syncer/src/verifier.rs
- gadgets/src/aggregation/aggregation.rs
- gadgets/src/keccak/keccak_stark.rs
- gadgets/src/keccak/logic.rs
- gadgets/src/keccak_sponge/keccak_sponge_stark.rs
- gadgets/src/logic.rs
- gadgets/src/lookup/bytes/byte_stark.rs
- gadgets/src/lookup/ex_type/ex_type_stark.rs
- gadgets/src/lookup/exp/gamma_stark.rs
- gadgets/src/lookup/path/path_stark.rs
- gadgets/src/recursion/get_challenge.rs
- gadgets/src/recursion/mod.rs
- gadgets/src/recursion/prover.rs

- gadgets/src/stark/proof.rs
- gadgets/src/stark/prove_with_p2.rs
- gadgets/src/stark/stark.rs
- gadgets/src/stark/utils.rs
- gadgets/src/stark/verifier_with_p2.rs
- gadgets/src/stark/verify_challenges.rs
- gnark/aggregate/circuits/agg_circuit.go
- gnark/appagg/circuits/app_agg_on_bn254.go
- gnark/hash2hash/circuits/common_circuit.go
- gnark/hash2hash/circuits/middle_agg.go
- gnark/hash2hash/circuits/receipt_hash_circuit.go
- gnark/hash2hash/circuits/storage_hash_circuit.go
- gnark/hash2hash/circuits/transaction_hash_circuit.go
- gnark/smt/circuits/smt_updater_circuit.go
- starky/src/macros.rs
- starky/src/poseidon2/stark.rs
- starky/src/receipt/logs/logs_decode.rs
- starky/src/receipt/logs/pub_input_stark.rs
- starky/src/receipt/mpt/inner_stark.rs
- starky/src/receipt/mpt/mpt_stark.rs
- starky/src/receipt/mpt/mpt_verify_with_ctl.rs
- starky/src/receipt/mpt/recursion_verify_with_ctl.rs
- starky/src/receipt/receipt/receipt_decode.rs
- starky/src/smt/block/block_decoder.rs
- starky/src/smt/smt_inclusion/smt_inclusion_stark.rs
- starky/src/storage/account/account_stark.rs
- starky/src/storage/account/inner_stark.rs
- starky/src/storage/account/recursion_verify_with_ctl.rs
- starky/src/storage/account/verify_with_ctl.rs
- starky/src/storage/account_leaf/account_leaf_stark.rs
- starky/src/storage/storage/inner_stark.rs

- starky/src/storage/storage/storage_stark.rs
- starky/src/transaction/mpt/inner_stark.rs
- starky/src/transaction/mpt/mpt_stark.rs
- starky/src/transaction/mpt/mpt_verify_with_ctl.rs
- starky/src/transaction/mpt/recursion_verify_with_ctl.rs
- starky/src/transaction/receipt/receipt_status_decode_stark.rs

Final Commit Hash

d5889f2067a32668e01899c1d34d056d217831b4

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
19	6	11

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Using static lengths for dynamic size fields allows for other fields manipulation in RLC argument

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/75

Context

Correspondance of large byte arrays between the different chips of the coprocessor are constrained by lookups of RLCs (random linear combination). However the way the RLC is computed on RLP decoded structures is underconstrained and allows manipulation of multiple fields.

Root cause

At the core the issue lies in the way dynamic length structures are handled when converted to a STARK friendly format (fixed size rows). The dynamic length structures are assigned a fixed length, large enough to accomodate them. This relies on the assumption that the RLC argument will be enough to prevent the user from providing inadequate input.

The RLC argument will collapse a byte array into a field element using the following formula:

RLC Argument

$$\text{RLC}(\mathbf{b};\gamma) = \sum_{i=0}^{n-1} b_i \cdot \gamma^{n-i}$$

These individual RLC values are then combined using offsets (`*_exp` columns in Brevis code):

$$\text{RLC} = \sum_{i=0}^{k-1} \text{RLC}_i \cdot \gamma^{o_i}$$

Unfortunately we can see that if the number of non-zero elements in array `i` is bigger than `o_i`, The RLC contribution of the ith array "interferes" over the RLC contribution of the `i+1th` array in a predictable way.

Abstract example

Let's take the simple example of two dynamically sized arrays of length max 2 (for the sake of simplicity the values of the actual RLP prefixes are omitted here)

$$A : [a_1, a_2]$$

$$B : [b_1, b_2]$$

Let's consider that the raw input is `A|B`, A is of length `2` and B is of length `1` (meaning `b_1 == 0`).

1/ Expected case:

coeff	gamma^2	gamma^1	gamma^0
A	a_1	a_2	
B			b_2
sum	a_1	a_2	b_2

2/ Malicious case:

Instead of providing the expected value for A and B, the prover modifies `a_2` and `b_1` into `a'_2` and `b'_1` such that:

$$a_2 = a'_2 + b'_1$$

The updated table becomes:

coeff	gamma^2	gamma^1	gamma^0
A	a_1	a'_2	
B		b'_1	b_2
sum	a_1	a_2	b_2

Which means the same RLC than the raw input for alternative values of A and B.

Concrete example

This vulnerability is the most pronounced in the `block_decoder` stark:

column.rs#L181-L184:

```
>>  pub extra_to_nonce: [T; 1024],
    pub basefee: [T; 16],
    pub basefee_u64: [T; 2],
>>  pub left_data: [T; 96],
```

The two fields `extra_to_nonce` and `left_data` have large fixed lengths (1024 and 96), and are unused. Using the manipulation described above, the unused parts of these fields can be used to modify other fields of the row, including the most critical:

column.rs#L170-L172:

```
pub root: [T; 32],
pub tx_hash: [T; 32],
pub receipt_hash: [T; 32],
```

Recommendation

To avoid the components from arrays which should be zeroed out to interfere with other fields, a new lookup type needs to be introduced, which would check that only N elements of an array are non zero.

Let's translate that to the `block_decoder` example above. If `extra_to_nonce` length RLP prefix is `32`, we should ensure that a lookup to `array_length` stark* is made:

```
lookup(ArrayLengthTable, extra_to_nonce_exp, time_data_exp, paddedextra_to_nonce)
```

Where `array_length` stark will ensure that `extra_to_nonce` array only has `extra_to_nonce_exp - time_data_exp` non-zero elements.

*`array_length` stark is yet to be implemented

Discussion

CergyK

@bytetang Have reviewed the usage of `padding_columns_check_ext_circuit, padding_columns_check_p` to fix this issue and it looks very good!

Some small remarks:

- `extension_or_leaf_value_elements_len` should be renamed to `gas_data_len`
  https://github.com/brevis-network/brevis-core/blob/audit_fix/starky/src/receipt/receipt/receipt_decode.rs
  allowbreak #L412
- `padding_columns_check_p, padding_columns_check_ext_circuit` should be checked on `local.key_path_nibbles` in the following starks:

starky/src/storage/account/account_stark.rs
starky/src/transaction/mpt/mpt_stark.rs starky/src/receipt/mpt/mpt_stark.rs

- `padding_columns_check_p, padding_columns_check_ext_circuit` should be checked on `extra_data` in `block_decoder.rs`

grrThea

I've submitted the fix here. Can you check and confirm? @CergyK

CergyK

FIx looks good

Issue H-2: Unconstrained multiplicities for cross-table-lookups

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/76

Context

Cross table lookups are used to enforce data consistency accross chips. The general mechanics of a lookup is that a tuple:

$$(a_0, ..., a_i, ..., a_n)$$

Is checked to be present both in a ”looking table” and a ”looked table” with a multiplicity ”looking_multiplicity” and ”looked_multiplicity”. Typically looked multiplicity can be set arbitrarily, as the looked table only enforces constraints on the tuple itself (for example correctness of a bitwise operation).

On the other hand, the multiplicity for looking table must be handled carefully, because it is additive accross different rows, and if it overflows the field order for a given tuple, the constraints in the looked table can be avoided. This typically happens if the sum of multiplicities for a given tuple accross rows becomes zero.

Attack scenario

The prover creates proofs relying on a false range-check operation. Let's take for example `(LEU, 1, 128, 0)`, which can be translated to `128 must be lower or equal unsigned to 0`. If the multiplicity is unconstrained, on one row the prover provides `-1` and on the next provides `1`.

As a result the sum of multiplicities for the tuple `(LEU, 1, 128, 0)` accross all rows becomes zero, and we don't need to provide a row for it in the range check stark.

Concrete examples

Multiple filters in multiple starks are vulnerable to this manipulation, and we are attempting to provide an exhaustive list here:

```
block_syncer/src/blocks/chunk_block.rs
- ChunkBlockCols::is_chunk_end

starky/src/receipt/column.rs
- LogCols::public_filter
- MptProofColumnsView::is_extension_elements_start
- MptProofColumnsView::is_branch
```

```
starky/src/smt/column.rs
- SmtInclusionColumnsView::is_blk
- BlkHeaderColumnsView::enable
- BlkHeaderColumnsView::not_zero_basefee

starky/src/storage/column.rs
- AccountColumnsView::not_single_extension_key
- AccountColumnsView::is_hash_state_root
- AccountColumnsView::is_extension_elements_start
- AccountColumnsView::not_first_elements_end
- AccountColumnsView::multi

- AccountLeafColumnsView::not_single_nonce
- AccountLeafColumnsView::not_single_balance

- StorageColumnsView::not_single_extension_key
- StorageColumnsView::not_single_extension_elements
- StorageColumnsView::not_first_elements_end
- StorageColumnsView::is_hash_state_root
- StorageColumnsView::multi
- StorageColumnsView::non_exist_extension
- StorageColumnsView::is_last_proof_elements_end
- StorageColumnsView::is_exist_extension_elements_start

starky/src/transaction/column.rs
- ReceiptStatusDecodeCols::enable
- MptProofColumnsView::not_single_extension_key // additionally should have a
↪  constraint to not always be false
- MptProofColumnsView::is_hash_tx_root // Should have a constraint to not always be
↪  false
- MptProofColumnsView::is_extension_elements_start
- MptProofColumnsView::not_first_elements_end
- MptProofColumnsView::is_leaf_elements_end // somewhat constrained, but not all
↪  cases covered
- MptProofColumnsView::multi
- MptProofColumnsView::is_branch // weakly constrained, booleaness should be
↪  checked explicitely
```

We attempt to provide an exhaustive list of manipulatable values, however we recommend that the boolean check is implemented exhaustively on all columns provided to a ctl filter.

Recommendation

The multiplicities of filter columns must always be checked to be:

- Must be Boolean (0 or 1)
- Only true if stark row is `enabled`, meaning it is not a padding row

Discussion

grrThea

@CergyK hello, why `MptProofColumnsView::not_single_extension_key` shouldn't always be false?

CergyK

> @CergyK hello, why `MptProofColumnsView::not_single_extension_key` shouldn't always be false?

It is never constrained to be true, so since it is a multiplicity for CTL, the prover can just always set it to zero to bypass the CTL constraint. It is similar to what is described in https://github.com/sherlock-audit/2025-04-brevis-network/issues/79 allowbreak #issuecomment-2943178936, but not used as a constraint "filter"

CergyK

Reviewed fix for this issue, it seems we are still missing the constraints for the following listed columns:

```
starky/src/transaction/column.rs
- MptProofColumnsView::is_extension_elements_start
- MptProofColumnsView::not_first_elements_end
- MptProofColumnsView::is_leaf_elements_end
- MptProofColumnsView::multi
- MptProofColumnsView::is_branch
```

Am I missing something?

grrThea

Here is the fixed commit, please help check. @CergyK

CergyK

Fix looks good, missing multiplicities constrained

Issue H-3: Padding rows can break transition constraints in path_stark

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/77

Context

The `enable` column is used to indicate that the row is not a padding row. Most constraints are bypassed when the row is not `enabled`. A standard way to ensure that transition constraints are enforced for all valid rows, is to ensure that all padding rows are located at the end of the row list (Once a row is not-enabled, all following rows are not-enabled).

Root cause

This constraint on padding rows is not enforced in `path_stark.rs` and a malicious prover can interleave padding rows with valid rows to bypass the following state transition constraint:

path_stark.rs#L198:

```
yield_constr.constraint(filter.mul(next_idx - idx - P::ONES));
```

path_stark.rs#L208:

```
yield_constr.constraint(enable.mul(eval_local_rlc - local_rlc));
```

Recommendation

Implement constraints enforcing that once a row is non-enabled, all following rows are non-enabled:

```
+    yield_constr.constraint((P::ONES-enable)*(next.enable));
```

Additionally the same constraints should be implemented in the following starks:

```
starky/src/transaction/receipt/receipt_status_decode_stark.rs
starky/src/smt/block/block_decoder.rs
block_syncer/src/blocks/chunk_block.rs
starky/src/receipt/receipt/receipt_decode.rs
```

Discussion

CergyK

Fix looks good

Issue H-4: Lack of gamma lookup in path_stark

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/78

Vulnerability description

A random challenge value gamma is used to compute the accumulator for the RLC argument. In the path stark, the gamma value is not actually sampled from phase2, but directly provided by the prover. This means that the prover can freely provide any coefficient value and manipulate lookup validity.

path_stark.rs#L200-L204:

```
    let local_acc = p2_local_vals[2];
    let next_acc = p2_next_vals[2];
    let local_rlc = p2_local_vals[1];
    let next_rlc = p2_next_vals[1];
>>  let local_exp_coefficient = p2_local_vals[0];
```

Recommendation

Use a lookup to enforce that `local_exp_coefficient` is indeed `gammâoffset`

Discussion

CergyK

Fix looks good, lookup introduced as needed

Issue H-5: Some constraint filters can remain always false

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/79

Context

Some columns are used as a filter for other constraints, cancelling them in some cases. As a concrete example:

path_stark.rs#L196-L213:

```
            let enable = p1_local_vals[self.enable_index()];
            let idx = p1_local_vals[0];
            let next_idx = p1_next_vals[0];
            let end = p1_local_vals[self.end_index()];
>>          let start = p1_local_vals[self.start_index()];

            // constrain: next_idx - local_idx = 1
            let filter = enable * (P::ONES - end);
            yield_constr.constraint(filter.mul(next_idx - idx - P::ONES));

            let local_acc = p2_local_vals[2];
            let next_acc = p2_next_vals[2];
            let local_rlc = p2_local_vals[1];
            let next_rlc = p2_next_vals[1];
            let local_exp_coefficient = p2_local_vals[0];

            // constrain: next_acc = local_rlc + next_rlc * next_con_rlc_exp
            let eval_local_rlc = p1_local_vals[self.key_path_nibble_index()] *
 ↪  local_exp_coefficient;
            yield_constr.constraint(enable.mul(eval_local_rlc - local_rlc));

            let eval_next_acc = local_acc + next_rlc;
            yield_constr.constraint(filter.mul(next_acc - eval_next_acc));

>>          yield_constr.constraint(start.mul(local_rlc - local_acc));
```

In the snippet above, `start` is a column which forces a constraint `local_rlc == local_acc` when true. However `start` itself is never constrained to be true; which means that the constraint `local_rlc == local_acc` can always be bypassed.

Vulnerable columns

The same type of vulnerability is to be found in multiple starks accross the code:

```
block_syncer/src/blocks/chunk_block.rs
    - ChunkBlockCols::is_chunk_end

starky/src/transaction/column.rs
    - ReceiptStatusDecodeCols::is_left_end
    - MptProofColumnsView::is_extension_elements_start

starky/src/smt/column.rs
    - BlkHeaderColumnsView::not_last_row

block_syncer/src/smt_update/smt_update_stark.rs
    - ChunkHashColumnsView::is_blk
    - ChunkHashColumnsView::is_blk_rlp_end
    - ChunkHashColumnsView::is_chunk_end
```

Recommendation

A proper state transition must be enforced for start to be constrained to be true when appropriate:

```
+    yield_constr.constraint(end.mul(P::ONES - next.start));
```

> The rest of the fields mentioned in `Vulnerable columns` section must be constrained similarly

Discussion

bytetang

@CergyK `is_block_rlp_end` does not exist in the `ChunkBlockCols`. Did you mean the `is_blk_rlp_end` in the `smt_update_stark.rs`

CergyK

@bytetang I meant `is_chunk_end` in `ChunkBlockCols`

I confused myself here because I'm suggesting to renaming it `is_block_rlp_end` in issue https://github.com/sherlock-audit/2025-04-brevis-network/issues/101

> Fixed the report body

CergyK

Reviewed fix, it seems we are still missing the needed constraints on:

```
block_syncer/src/smt_update/smt_update_stark.rs
    - ChunkHashColumnsView::is_blk
```

Am I missing something?

bytetang

Look at the code here `let blk_rlp_chunk_rlc = local.is_blk * bytes_rlc_p(&blk_rlp_chunk, gamma.clone());` The `is_blk` is now used as the multiplier in the RLC calculation. The transition is accumulated with the accumulator (acc) RLC. This can prevent the `is_blk` as lookup filter from being cancelled.

CergyK

> Look at the code here `let blk_rlp_chunk_rlc = local.is_blk * bytes_rlc_p(&blk_rlp_chunk, gamma.clone());` The `is_blk` is now used as the multiplier in the RLC calculation. The transition is accumulated with the accumulator (acc) RLC. This can prevent the `is_blk` as lookup filter from being cancelled.

You are right, fix looks good

Issue H-6: input_block_bytes not range checked in keccak_sponge_stark.rs

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/80

Vulnerability description

`input_block_bytes` is a column array which is supposed to be representing a byte array. Unfortunately the individual values are never range checked to be in `[0, 255]`. As a result a malicious prover can provide `input_block_bytes` with out of bounds values, and spoof an invalid keccak hash.

> The byte values from `input_block_bytes` are aggregated into u32s before making a lookup keccak_sponge_stark.rs#L120-L126. Unfortunately this does not prevent the manipulation as the collapsing into u32s will allow for overflow.

Recommendation

Constrain individual values from `input_block_bytes` to be bytes.

Discussion

bytetang

fix commits: 63c72537fd9780fba1230baed6d2b26552089a79
4971fde6d81b90a93ea016a0066a9990b3eb4d01

CergyK

Looks good, lookups are missing in `starky/src/storage/account_leaf/multiple_stark.rs` but it looks like it is dead code

bytetang

Yes, multiple_stark.rs under the account_leaf folder here is just for a minimal test.

Issue H-7: mpt_key_u64 can be spoofed by shuffling branch elements

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/81

Context

To prove a transaction data point, we need to make a correspondance between the transaction data and the related receipt. The transaction and receipt MPT are different from storage and account MPTs because elements are not indexed by their hash, but by the RLP encoding of their `index` inside the block. `mpt_key_u64` is a u64 key derived from the RLP encoding of the transaction index, and is used to make the correspondance between tx and receipt.

Vulnerability detail

Due to how the RLC argument is implemented, if no monotony constraints are made on the RLC coefficients (used to aggregate multiple partial RLCs), partial RLCs can be shuffled as long as their RLC coefficient remains the same. This means that when incorporating a MPT branch we can swap the branch element data 5 with element data 6, but keeping `element_idx` values the same. That way it is possible to associate a succeeded receipt with a failed transaction (because the path followed to the receipt would be different than the one followed to the tx).

Recommendation

When incorporating a MPT branch, we need to ensure that `branch_exp` is bigger than previous element, that way it is impossible to reorder branch elements

Discussion

CergyK

Fix looks good

Issue H-8: Topics and data can be shuffled in logs_decode stark, proving incorrect public values

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/82

Context

The logic for selecting the piece of data to be extracted from a log (either a topic at index idx or data at index idx), relies on the `element_idx`. This index depends on the order in which the topics and data words are provided, but not necessarily the order in which they occur in the event (keep in mind that we can provide `topic_exp` out of order, and obtain the same accumulated rlc value)

Impact

An incorrect data word can be extracted from a log, and exported as a public input.

POC

Apply the following git diff, and run:

```
FAST_MODE=1 CASE=0 cargo test --package brevis-starky --lib --release --
↪  receipt::logs::logs_decode::test::test_prove_logs
```

```
diff --git a/starky/src/receipt/logs/logs_decode.rs
↪  b/starky/src/receipt/logs/logs_decode.rs
index 4605b49..f998915 100644
--- a/starky/src/receipt/logs/logs_decode.rs
+++ b/starky/src/receipt/logs/logs_decode.rs
@@ -345,6 +345,17 @@ impl<F: RichField + Extendable<D>, const D: usize>
↪  BrevisAir<F, D> for LogsDecod
         let len = rows_all.len();
         rows_all[len-1].last_row = F::ONE;

+        {
+          //@audit swap the order of the topics on two consecutive rows
+          //@audit this will allow to swap the value indices in the witness as
↪  well, falsifying public values
+          let tmp_topic_exp = rows_all[5].topic_exp;
+          let tmp_topic_segment = rows_all[5].topic_segment.clone();
+          rows_all[5].topic_exp = rows_all[4].topic_exp;
+          rows_all[5].topic_segment = rows_all[4].topic_segment.clone();
+          rows_all[4].topic_exp = tmp_topic_exp;
+          rows_all[4].topic_segment = tmp_topic_segment;
+        }
```

```
+
         let mut trace = vec![];
         rows_all.iter().for_each(|row| {
           trace.push(row.clone().into());
@@ -1366,14 +1377,11 @@ mod test {
         // let receipt11: Receipt = format_receipt::<F, D, C>(receipt_info11);

       let mut receipts = vec![
-        receipt3.clone(),
-        receipt2.clone(),
-        receipt3.clone(),
-        receipt2.clone()
+        receipt3.clone()
       ];

       // let mut receipts =
↪  vec![receipt3.clone(),receipt10.clone(),receipt11.clone()];
-      receipts.resize(SUB_PROOF_THUNK_SIZE, receipt3.clone());
+      receipts.resize(SUB_PROOF_THUNK_SIZE, receipt2.clone());

       let public_inputs = generate_receipt_public_input_fields::<F, D,
↪  C>(receipts.clone());

diff --git a/starky/test_data/receipt/receipt_test1.json
↪  b/starky/test_data/receipt/receipt_test1.json
index c032bd0..80aaa14 100644
--- a/starky/test_data/receipt/receipt_test1.json
+++ b/starky/test_data/receipt/receipt_test1.json
@@ -7,7 +7,7 @@
           "log_index": 1,
           "log_topic0":
↪  "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
           "value_from_topic": true,
-          "value_index": 1,
+          "value_index": 2,
           "value":
↪  "0x0000000000000000000000000000000000000000000000000000000000000000",
           "field_num_in_topics": 3
       },
@@ -16,7 +16,7 @@
           "log_index": 1,
           "log_topic0":
↪  "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
           "value_from_topic": true,
-          "value_index": 2,
+          "value_index": 1,
           "value":
↪  "0x000000000000000000000005c415f5d776dc6cd8c92ad7aa0ee7aaf0583ac4c",
           "field_num_in_topics": 3
```

```
        },
```

Recommendation

Enforce that `topic_exp` and `data_exp` are in desc order accross rows

Discussion

CergyK

Fix looks good

Issue H-9: Keccak stark does not enforce booleaness of bits in bit decomposition (pre-audit)

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/83

This issue was found during pre-audit, added here for completeness

Description

The keccak circuits have additional columns for bit decomposition in order to facilitate xor operations. However these columns are provided directly by the prover and are not checked to be boolean (in {0, 1});

Here we can see that the decomposition of `a_prime_prime_0_0_lo` and `a_prime_prime_0_0_hi` is correctly checked, but `a_prime_prime_0_0_bits` individual values booleaness is not checked: gadgets/src/keccak/keccak_stark.rs#L366-L379:

```
// A'''[0, 0] = A''[0, 0] XOR RC
let a_prime_prime_0_0_bits = (0..64)
    .map(|i| local_values[reg_a_prime_prime_0_0_bit(i)])
    .collect_vec();
let computed_a_prime_prime_0_0_lo = (0..32)
    .rev()
    .fold(P::ZEROS, |acc, z| acc.doubles() + a_prime_prime_0_0_bits[z]);
let computed_a_prime_prime_0_0_hi = (32..64)
    .rev()
    .fold(P::ZEROS, |acc, z| acc.doubles() + a_prime_prime_0_0_bits[z]);
let a_prime_prime_0_0_lo = local_values[reg_a_prime_prime(0, 0)];
let a_prime_prime_0_0_hi = local_values[reg_a_prime_prime(0, 0) + 1];
yield_constr.constraint(computed_a_prime_prime_0_0_lo - a_prime_prime_0_0_lo);
yield_constr.constraint(computed_a_prime_prime_0_0_hi - a_prime_prime_0_0_hi);
```

Recommendation

One should ensure that every column which is meant to be byte or bit, should be range checked in `keccak_stark`.

Discussion

bytetang

fix commit b3d70b15d21c5c5e181d93286072729058d3c2a8

CergyK

Fix looks good

Issue H-10: RLC coefficients are not constrained with regards to RLP prefixes

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/84

Context

The coprocessor uses RLC and lookups arguments to make the correspondance between the RLP decoding of a structure (`starky/src/receipt/logs/logs_decode.rs` stark), and the raw input bytes used for MPT inclusion proof (`starky/src/receipt/mpt/mpt_stark.rs`).

Partial RLC values are computed on the RLP decoded structure, and reassembled by using appropriate gamma exponents:

$$\text{RLC} = \sum_{i=0}^{k-1} \text{RLC}_i \cdot \gamma^{o_i}$$

Vulnerability description

Unfortunately the gamma exponents are not constrained with regards to the RLP prefix value (encoding length for the data segment).

This enables a malicious prover to modify individual values (especially if they have leading or trailing zeros) by shifting the gamma exponent, without affecting the overall RLC value

Recommendation

Strictly constrain the gamma exponents with regards to the decoded prefix. For example in `block_decoder.rs`:

```
+    yield_constr.constraint(gasused_prefix_exp - gaslimit_data_exp -
↪  (gaslimit_prefix - F::from_canonical_u64(128)));
```

This is a simple example since length is directly encoded in a single byte prefix `gaslimit_prefix`. For longer segments, a ”big” length must be computed from prefix and length array.

Discussion

CergyK

The fix is well implemented, and *_exp fields are now rightly constrained with regards to length of RLP values. However unless I am missing something, the constraints are missing for the following fields:

```
starky/src/smt/block/block_decoder.rs
  extra_data_coefficient
  mix_digest_coefficient
  extra_data_exp
  gaslimit_prefix_exp
  gaslimit_data_exp
  gasused_prefix_exp
  gasused_data_exp
  time_prefix_exp
  time_data_exp
  mix_digest_exp

starky/src/storage/column.rs
  key_path_nibbles_exp

starky/src/transaction/column.rs
  key_path_nibbles_exp

starky/src/receipt/column.rs
  key_path_nibbles_exp
```

grrThea

I've submitted the fix https://github.com/brevis-network/brevis-core/pull/153. Can you check and confirm? @CergyK

CergyK

*_exp fields for newly added fields have been correctly constrained now, fix looks good

Issue H-11: status should be constrained to be zero on rows which are not is_left_end in receipt_status_decode_stark

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/88

Description

in `receipt_status_decode_stark.rs`, `status` is a column which is used as a filter for the lookup against `receipt_mpt_stark`. This lookup implicitely allows the receipt to be used as a successful transaction (indeed if `status == 0` the lookup does not reconcile and the `receipt_mpt_stark` inclusion cannot be proved). Unfortunately `status` is only constrained when `is_left_end`:

receipt_status_decode_stark.rs#L257:

```
yield_constr.constraint(local.is_left_end.mul(local.field1 - local.status));
```

Which means that if not `is_left_end`, we can still set `status` to 1 and spoof transaction validity

Impact

Data point about a transaction can be proved even though the transaction was rejected

Recommendation

Additionally enforce that status is zero when not `is_left_end`:

```
+   yield_constr.constraint((P::ONES-local.is_left_end).mul(local.status));
```

Discussion

CergyK

Fix looks good

Issue H-12: left_rlc_acc is fully unconstrained in receipt_status_decode_stark.rs

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/89

Description

`receipt_status_decode_stark` uses a lookup argument to match the input used with the one `receipt_mpt_stark` uses. Unfortunately a partial rlc value provided in the columns `p2_local.left_rlc_acc` is left fully unconstrained.

receipt_status_decode_stark.rs#L288-L289:

```
// left fields
receipt_rlc += p2_local.left_rlc_acc;
```

Impact

This means that the rlc argument can be completely forged, since the prover can just provide the difference between `receipt_status_decode_stark` rlc and expected value in `receipt_mpt_stark`.

Recommendation

`p2_local.left_rlc` should be computed directly from p1_local.left and gamma like is done for big len:

receipt_status_decode_stark.rs#L281-L283C13:

```
let big_len: &[P] = &local.big_len.map(|b| b.into());
let big_len_rlc = bytes_rlc_p(big_len, gamma.clone());
receipt_rlc += big_len_rlc * p2_local.big_len_len_coefficient;
```

Then `p2_local.left_rlc_acc` should be constrained accross rows.

Discussion

CergyK

Fix looks good

Issue H-13: Some logical implications are only partially enforced

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/90

Description

Sometimes, arithmetic constraints attempt to translate logical implications as formulas. Unfortunately in code the implications are only translated partially in some cases. Let's study the concrete example of the relation established for columns `is_leaf_elements_end`, `is_elements_end` and `is_leaf` in `starky/src/transaction/mpt/mpt_stark.rs`.

The desired relation would be:

However the arithmetic constraint in code:

mpt_stark.rs#L627-L628:

```
yield_constr.constraint((P::ONES - local.is_leaf_elements_end) *
↪  local.is_elements_end * local.is_leaf);
```

Achieves only the following:

And we need to add constraints in order to enforce the other side of the equivalence.

Concrete examples

This type of vulnerability is to be found in multiple starks across the code:

```
starky/src/receipt/column.rs
- MptProofColumnsView::is_leaf_elements_end
```

Impact

The impact may vary according to the specific occurence of the issue, but in many cases the state transitions can be broken and false proofs constructed.

Recommendation

mpt_stark.rs#L627-L628:

```
    yield_constr.constraint((P::ONES - local.is_leaf_elements_end) *
 ↪  local.is_elements_end * local.is_leaf);
-   // yield_constr.constraint(local.is_leaf_elements_end * (P::ONES -
 ↪  local.is_leaf));
+   yield_constr.constraint(local.is_leaf_elements_end * (P::ONES - local.is_leaf));
+   yield_constr.constraint(local.is_leaf_elements_end * (P::ONES -
 ↪  local.is_elements_end));
```

Discussion

CergyK

Suggested fix has been applied, and added for

starky/src/storage/account/account_stark.rs

starky/src/transaction/mpt/mpt_stark.rs

Issue H-14: Remaining slot_hash_str should be truncated to key_path_len when proving non-existence case

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/91

Context

Values are stored in the storage MPT only when they have already been written. By EVM convention, non-written values are equivalent to be zero. We need to prove non-existence of a storage leaf when we want to prove that a storage value is zero.

In that case this means that descending the MPT following the storage slot path would stop early on a branch or extension node; The coprocessor proves that the remainder after having followed the path for X steps, of the slot hash is different from the extension key path we have found.

Description

Unfortunately when processing the difference between the remainder of the slot hash and current extension key path, the remainder of the slot hash is not truncated to match the length of the extension key path; As a result we can prove non-existence of a storage slot (keep in mind this proves that the storage value is 0), for any path having an extension.

storage/generation.rs#L178-L281:

```
    let slice_slot_hash_str = slot_hash_str[key_path_idx..].to_string();

    if slice_slot_hash_str.starts_with(&extension_key_path_hex_without_type) {
        ... // Existence case
    } else {
        // no existent leaf node
        assert_eq!(is_last_proof, true);
        row.non_exist_extension = F::ONE;

        //@audit slice_slot_hash_str includes the whole remainder until the end. If
↪   the extension key path is strictly shorter, the xor will always be != 0.
>>      let nibbles_without_type = hex_2_nibbles(slice_slot_hash_str.clone());
>>      let slice_slot_hash_str_with_type =
↪   extension_node_type_padded(slice_slot_hash_str.clone(),
↪   row.is_leaf.to_canonical_u64());
>>      let nibbles = hex_2_nibbles(slice_slot_hash_str_with_type);

        row.key_path_len = F::from_canonical_u64(nibbles_without_type.len() as u64
↪   - 1);
```

```
        row.full_key_path_len = F::from_canonical_u64(nibbles.len() as u64 - 1);

        let padded_nibbles = padded_cols(EXTENSION_KEY_WIDTH, nibbles_without_type);
        let padded_full_nibbles = padded_cols(EXTENSION_KEY_WIDTH, nibbles);

        row.key_path_nibbles
            .iter_mut()
            .enumerate()
            .for_each(|(i, r)| *r = F::from_noncanonical_u64(padded_nibbles[i] as
↪   u64));
        row.key_path_full_nibbles
            .iter_mut()
            .enumerate()
            .for_each(|(i, r)| *r = F::from_noncanonical_u64(padded_full_nibbles[i]
↪   as u64));
        row.key_path_nibbles_exp = F::ZERO;
    }

    let ext_res_key = res[0][1..res[0].len()-1].to_vec();
    let ext_res_key_u8: Vec<u8> = ext_res_key.iter().map(|v| *v as u8).collect();
    let ext_res_key_hex = hex::encode(ext_res_key_u8);


    let ext_res_key_nibbles = hex_2_nibbles(ext_res_key_hex);
    let padded_nibbles = padded_cols(EXTENSION_KEY_WIDTH, ext_res_key_nibbles);


    row.extension_key_elements
        .iter_mut()
        .enumerate()
        .for_each(|(i, r)| *r = F::from_noncanonical_u64(padded_nibbles[i]));


    row.extension_key_elements_exp =
↪   F::from_noncanonical_u64(*res[0].last().unwrap());

    row.key_path_full_nibbles.iter().zip(row.extension_key_elements.iter()).enumera
↪   te().for_each(|(i,(b,c))| {
        if b.eq(c) { row.extension_hash_xor[i] = F::ZERO } else {
↪   row.extension_hash_xor[i] = F::ONE }
    });
    let mut sum = F::ZERO;

    //@audit Even though hash xor is only used in the non-existence case, it is
↪   always computed
>>  row.extension_hash_xor.iter().for_each(|v| { sum += *v; });
>>  row.key_path_xor_sum = sum;
```

Recommendation

We need to make sure that the part taken from the remaining slot hash is only `key_path_len`, and use the new lookup introduced by #75 to check that after `key_path_len` elements, `row.key_path_nibbles` has only zero elements.

Additionally generation should be fixed:

storage/generation.rs#L178-L281:

```
-    let slice_slot_hash_str = slot_hash_str[key_path_idx..].to_string();
+    let slice_slot_hash_str =
 ↪  slot_hash_str[key_path_idx..key_path_idx+key_path_len].to_string();
```

POC

Apply the following patch:

```
diff --git a/starky/src/storage/storage/generation.rs
 ↪  b/starky/src/storage/storage/generation.rs
index a86e8a6..829010e 100644
--- a/starky/src/storage/storage/generation.rs
+++ b/starky/src/storage/storage/generation.rs
@@ -177,7 +177,7 @@ pub fn generate_storage_row<F: RichField + Extendable<D>, const
 ↪  D: usize>(

         let slice_slot_hash_str = slot_hash_str[key_path_idx..].to_string();

-        if slice_slot_hash_str.starts_with(&extension_key_path_hex_without_type) {
+        if false {
           if element_idx == 0 && !is_last_proof {
             row.multi = F::ONE;
           }
diff --git a/starky/test_data/storage/storage_witness_data_7.json
 ↪  b/starky/test_data/storage/storage_witness_data_7.json
index a0a025f..9362924 100644
--- a/starky/test_data/storage/storage_witness_data_7.json
+++ b/starky/test_data/storage/storage_witness_data_7.json
@@ -1,6 +1,6 @@
 {
   "account_address": "0x030ba81f1c18d280636f32af80b9aad02cf0854e",
-  "slot": "0xbba75e81bcb06eb2c37e4991ae54514c225e4a239c62a15f38f2d0e32dd4e120",
+  "slot": "0x00000000000000000000000000000000000000000000000000000000033e62a",
   "slot_value": "0",
   "account_proofs": [
```

```
    "0xf90211a0a8586d6a0fb7fd55d02b35c01597228d0b9a69aff31b4441b99f4bcd2e14b3cfa04
↪  056e819eef9ad0eeed72a2ceb80e1fe6717831b908cb9a16a1808c0c6c33ecfa02da7421f68db00
↪  9d2070987ca06d3b15c7f677d0bb2a045a15482f4109cd2289a01adb78aba20ea3afe8090836095
↪  3283beade4d21244d916826ad6e52e5c2af24a02d83ffc7476d6e62205e544ec7a31d0382e6d9a3
↪  2f886a8fdf09c1615645f205a0c39da6f4c791c8fcb3162659c64a39088fcb08249d4857a24ff55
↪  9f84b782783a09cda2101868440c9fd48f5fde8c8f3e5339cb12772e937b25f860b8374ac15e0a0
↪  6f0401d949ea6220191f105b566d3e718fff5dc28fdd30e24accc0127ff12af1a0e97bb6abfbd16
↪  9785fbc6cf686609e904c811e72942bdb02b4edf460e97fc1f3a0613b0da30057e99d3a2cfe3cb9
↪  4f91324c94e89111abb181b20131601b96fbb6a0d676204cf7133314e0b34f450a596d6882bc6ae
↪  d67208f5eaa700876aea57e23a0bac651b00c9bb185134e841d2da8b22fa2f34c53eb349819825a
↪  4f6925f27384a01224776c54280fe2a0283732bca3a33094fd034e23eabc93cd77b21034ab0f97a
↪  0107a8e9f1531e39199cc2ade694fdbbd58f773866cfd87942e38eb7d4186997ea04b447c473102
↪  026f99aa87ae33006fb11f54a690cb25c94e8f13130ebefd2d77a01a5dbab30cbd1465d3342c0bc
↪  8a50f3f0cedf895d941ede206ed97e28dfa00a580",
```

And run the following command:

```
FAST_MODE=1 CASE=0 cargo test --package brevis-starky --lib --release --
↪  storage::account::account_stark::test::test_account_proof
```

Discussion

CergyK

Fix looks good. Conceptually this is the same issue as #75, so it is expected that it is the same fix. The only difference here is that the issue happened with the legitimate generation code as provided

Issue H-15: Mutually exclusive cases must be nullified when unused

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/93

Description

When processing MPT inclusion proofs, multiple cases can co-exist and fields for a row serve multiple purposes depending on case flags.

One example of this is MPT node processing, where node can be `extension`, `leaf`, `branch`. The corresponding flags are declared in the stark row:

- is_extension
- is_leaf
- is_branch

There are rlc fields which are only used when the associated flag is true:

- is_branch => p2_local.branch_node_rlc, p2_local.branch_coefficients
- is_leaf, is_extension => p2_local.extension_elements_coefficient, p2_local.extension_elements

For example `p2_local.branch_node_rlc` is only constrained to be equal to computed `branch_node_rlc`, over `branch_values` when `is_branch`: storage/storage_stark.rs#L1083-L1087:

```
yield_constr.constraint(
    local
        .is_branch
        .mul(p2_local.branch_node_rlc - branch_node_rlc),
);
```

This means that when `is_extension`, `p2_local.branch_node_rlc` can be an arbitrary value, used to modify any column while keeping a valid accumulated RLC.

Concrete examples

```
starky/src/storage/storage/storage_stark.rs (is_extension, is_leaf, is_branch)
starky/src/storage/account/account_stark.rs (is_extension, is_leaf, is_branch)
```

Recommendation

Force unused value to be zero when the appropriate flag is not set:

```
+    yield_constr.constraint(
+        (P::ONES - local.is_branch).mul(p2_local.branch_node_rlc)
+    );
```

POC

```
diff --git a/starky/src/storage/storage/generation.rs
↪  b/starky/src/storage/storage/generation.rs
index 707048d..c69e8ff 100644
--- a/starky/src/storage/storage/generation.rs
+++ b/starky/src/storage/storage/generation.rs
@@ -335,6 +335,13 @@ pub fn generate_storage_row<F: RichField + Extendable<D>,
↪  const D: usize>(

         row.extension_elements_exp = F::from_noncanonical_u64(*ext_res_len);
         row.row_exp = F::from_canonical_u64(*ext_res_len);
+
+        if row.chunk_idx.to_canonical_u64() == 26 &&
↪  row.row_idx.to_canonical_u64() == 2 && element_idx == STORAGE_BRANCH_NODES_NUM
↪  - 1 {
+          row.branch_exp = row.extension_value_prefix_exp;
+          row.extension_value_prefix += F::ONE;
+          log::error!("Element has extension_elements_prefix
↪  padded_element_ext_val {} {} {} {:?}", storage_index, proof_idx, element_idx,
↪  padded_element_ext_val);
+          log::error!("Element has extension value length {:?} {} {}",
↪  row.extension_value_big_len, row.extension_value_prefix_exp,
↪  row.extension_value_prefix);
+        }
       } else {
         if element_idx < branch_fields_num {
           row.is_branch = F::ONE;
diff --git a/starky/src/storage/storage/storage_stark.rs
↪  b/starky/src/storage/storage/storage_stark.rs
index 318c220..c2e1c6f 100644
--- a/starky/src/storage/storage/storage_stark.rs
+++ b/starky/src/storage/storage/storage_stark.rs
@@ -641,6 +641,14 @@ impl<F: RichField + Extendable<D>, const D: usize>
↪  BrevisAir<F, D> for StoragePr
                             p2_row.branch_acc = branch_nodes_rlc;
                         }

+                        if row.chunk_idx.to_canonical_u64() == 26 &&
↪  row.row_idx.to_canonical_u64() == 2 && row.is_elements_end.is_one() {
+                            let branch_rlc = F::NEG_ONE;
+                            p2_row.branch_node_rlc = branch_rlc;
```

```
+                            branch_nodes_rlc += branch_rlc *
 ↪  p2_row.branch_coefficients;
+                            p2_row.branch_acc = p2_row.branch_node_rlc *
 ↪  p2_row.branch_coefficients;
+                            log::error!("BRANCH_ACC {} {} {} {:?} is_branch: {}",
 ↪  p2_row.branch_acc, p2_row.branch_node_rlc, p2_row.branch_coefficients,
 ↪  row.branch_values, row.is_branch);
+                        }
+
                         if row.is_elements_end.to_canonical_u64() == 1 {
                             storage_rlc += row.prefix * p2_row.prefix_coefficient;
```

Discussion

CergyK

Fix is partial, it seems that `extension_elements_rlc` is not zeroed out when row is not leaf or extension receipt/mpt/mpt_stark.rs#L951-L952

grrThea

Here is the fixed commit. Please help confim. @CergyK

CergyK

@grrThea `local.extension_value_rlc` and `local.extension_elements_rlc` should be constrained to be zero when `is_branch`.

Also this should be implemented for all of the mpts: `receipt`, `transaction` and `storage` (because the code is very similar)

grrThea

@CergyK I've updated it. commit. please have a look.

CergyK

@grrThea could you please fix transaction, account and storage tests (in FAST_MODE=1)?

```
thread 'storage::storage::storage_stark::test::test_storage_proof' panicked at
 ↪  starky/src/storage/storage/verify_with_ctl.rs:173:17:
assertion `left == right` failed: Hash0 element 0 mismatch
  left: 9799574663422773101
 right: 8148666717762591294
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

grrThea

> @grrThea could you please fix transaction, account and storage tests (in FAST_MODE=1)?

```
thread 'storage::storage::storage_stark::test::test_storage_proof' panicked at
↪  starky/src/storage/storage/verify_with_ctl.rs:173:17:
assertion `left == right` failed: Hash0 element 0 mismatch
  left: 9799574663422773101
 right: 8148666717762591294
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Fixed.

CergyK

@grrThea could you please fix transaction, account and storage tests (in FAST_-MODE=1)?

```
thread 'storage::storage::storage_stark::test::test_storage_proof' panicked
↪  at starky/src/storage/storage/verify_with_ctl.rs:173:17:
assertion `left == right` failed: Hash0 element 0 mismatch
  left: 9799574663422773101
 right: 8148666717762591294
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Fixed.

Thank you. Fix looks good

Issue H-16: Incorrect Constraint for Zero-Value Limbs in AppAggCircuit

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/99

Description

In the `AppAggCircuit.Define` method, when processing certain public inputs from `innerWitness.Public` (specifically for constructing `smtRootHash` and `outCommitment`), the circuit intends to use only a subset of the `Limbs` of a field element. For example, it might use the lower 2 limbs out of 4. The implicit assumption is that the higher, unused limbs of these field elements must be zero.

The current implementation attempts to enforce this by using `api.Add(limb, 0)`. For example:

```
// brevis-core/gnark/appagg/circuits/app_agg_on_bn254.go
// ...
// Example for smtRootHash part from innerWitness.Public[1]
for i := 3; i > 1; i-- {
    api.Add(innerWitness.Public[1].Limbs[i], 0) // Problematic line: Does not
↪  constrain Limb to be 0
}
// ...
```

This pattern is repeated for `innerWitness.Public[1]`, `innerWitness.Public[2]`, `innerWitness.Public[3]`, and `innerWitness.Public[4]` where their higher limbs are processed.

The core issue is that `api.Add(variable, 0)` in the gnark frontend API merely returns the `variable` itself and **does not constrain or assert that `variable` is equal to zero.** As a result, the circuit does not correctly enforce that these specific upper limbs are zero. This can lead to a soundness vulnerability, as a malicious prover could potentially use non-zero values for these "unused" limbs, and the circuit would still accept the proof, potentially leading to invalid state transitions or assertions being bypassed.

Mitigation

To correctly enforce that these specific limbs are zero, the `api.Add(limb, 0)` calls must be replaced with `api.AssertIsEqual(limb, 0)`.

The corrected code structure should be:

```
// brevis-core/gnark/appagg/circuits/app_agg_on_bn254.go
// ...
// Corrected example for smtRootHash part from innerWitness.Public[1]
for i := 3; i > 1; i-- {
```

```
    api.AssertIsEqual(innerWitness.Public[1].Limbs[i], 0) // Correctly constrains
↪   the limb to be zero
}
// ...
```

This change should be applied to all instances where unused limbs are intended to be constrained to zero within the `Define` method of `AppAggCircuit`.

Issue H-17: Chunk change constraint in pub_input_stark.rs should be constraint_transition instead of constraint_first_row

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/100

Description

In `pub_input_stark.rs`, there is a constraint to enforce that `chunk_idx` changes only every 4 times `filter` is true:

logs/pub_input_stark.rs#L160:

```
yield_constr.constraint_first_row((next.chunk_idx - local.chunk_idx) *
↪  ((local.chunk_idx + P::ONES) * P::from(FE::from_canonical_u64(4)) -
↪  local.filter_sum));
```

Unfortunately the constraint is enforced only for the first row of the trace (indeed `constraint_first_row` is used), whereas it should be enforced for all transition rows of the trace.

Recommendation

Use `constraint_transition` instead of `constraint_first_row`:

logs/pub_input_stark.rs#L160:

```
- yield_constr.constraint_first_row((next.chunk_idx - local.chunk_idx) *
↪  ((local.chunk_idx + P::ONES) * P::from(FE::from_canonical_u64(4)) -
↪  local.filter_sum));
+ yield_constr.constraint_transition((next.chunk_idx - local.chunk_idx) *
↪  ((local.chunk_idx + P::ONES) * P::from(FE::from_canonical_u64(4)) -
↪  local.filter_sum));
```

Discussion

CergyK

Fixed as recommended

Issue H-18: branch_key_path not constrained with regards to element_idx in storage non-existence

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/104

Description

MPT non-inclusion argument in storage relies on first following the key path for as long as possible and then finding an extension which does not match the remainder of the path. In the case of a branch, we attempt to follow the node at the index having the value of the current nibble;

This is translated by the following equality in the generation code (`row.multi` column is to trigger an inner lookup): storage/generation.rs#L354-L358:

```
if branch_key_path == element_idx as u64 {
    if !is_last_proof {
        row.multi = F::ONE;
    }
}
```

Unfortunately, since `element_idx` is not even a part of the row at all, it is impossible to enforce the constraint that `branch_key_path == element_idx`.

A malicious prover can use the correct `branch_key_path`, but follow along the wrong `element_idx` of the tree (setting `row.multi = F::ONE` on the wrong element_idx). The prover is unlikely to find a match for the remainder of the path by following the wrong element, but is able to prove non-existence!

Impact

A malicious prover can prove non-existence for a slot which actually has a value in the contract, by following the wrong element index in the branch node (but providing correct `branch_key_path` value)

Recommendation

Add `element_idx` column to the row, enforce sequencing of `element_idx` for the branch node, and constraint that `branch_key_path == element_idx` in circuits

Discussion

CergyK

Fix looks good, inner lookup now happens only when `is_branch && element_idx == key_path_nibbles[EXTENSION_KEY_WIDTH - 1]`

Issue H-19: Storage public inputs are unconstrained during recursion in account proof

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/109

Description

When aggregating multiple starky proofs (smt inclusion, block decoder, storage decoder, account decoder), it is crucial to ensure the starky receive the same public inputs. This is enforced by the recursion circuit for (account decoder <-> smt inclusion) and (block decoder <-> account decoder) but is not enforced for (account decoder <-> storage decoder):

storage/account/recursion_verify_with_ctl.rs#L459-L469:

```
account_pt.clone().public_inputs.into_iter().enumerate().for_each(|(i, pi)| {
    let eq = builder.sub(blk_header_pt.public_inputs[i], pi);
    builder.assert_zero(eq);
});


account_pt.clone().public_inputs.into_iter().enumerate().for_each(|(i, pi)| {
    let eq = builder.sub(smt_inclusion_pt.public_inputs[i], pi);
    builder.assert_zero(eq);
});

//@audit missing equality check between account_pt and storage_pt public inputs

let pts = [&account_pt, &account_leaf_pt, &storage_pt, &blk_header_pt,
 ↪  &smt_inclusion_pt, &gamma_pt, &byte_pt, &path_pt, &ex_type_pt, &keccak_pt,
 ↪  &keccak_sponge_pt, &logic_pt, &poseidon2_pt, &account_inner_pt,
 ↪  &storage_inner_pt];
```

If the prover can pass any public values into the storage stark, this means the storage values are not actually constrained.

Recommendation

Enforce equality between `account_pt` and `storage_pt` public inputs:

storage/account/recursion_verify_with_ctl.rs#L459-L469:

```
    account_pt.clone().public_inputs.into_iter().enumerate().for_each(|(i, pi)| {
        let eq = builder.sub(blk_header_pt.public_inputs[i], pi);
        builder.assert_zero(eq);
    });
```

```
    account_pt.clone().public_inputs.into_iter().enumerate().for_each(|(i, pi)| {
        let eq = builder.sub(smt_inclusion_pt.public_inputs[i], pi);
        builder.assert_zero(eq);
    });

+    account_pt.clone().public_inputs.into_iter().enumerate().for_each(|(i, pi)| {
+        let eq = builder.sub(storage_pt.public_inputs[i], pi);
+        builder.assert_zero(eq);
+    });
```

Discussion

CergyK

Fixed as recommended

Issue M-1: Wrong field size for balance and nonce in account columns

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/85

Description

In the ethereum account structure, balance is supposed to be a uint256 and thus take 32 bytes, and nonce is uint64 (and thus take 8 bytes). However in the `AccountLeafColumnsView` structure these are encoded as 12 bytes and 32 bytes respectively:

storage/column.rs#L129-L132:

```
pub nonce_prefix: T,
pub nonce_data: [T; 32],
pub balance_prefix: T,
pub balance_data: [T; 12],
```

Recommendation

Even though it is quite unlikely that these sizes are actually reached for nonce and balance, it is more natural to use the adequate sizes:

storage/column.rs#L129-L132:

```
   pub nonce_prefix: T,
-  pub nonce_data: [T; 32],
+  pub nonce_data: [T; 8],
   pub balance_prefix: T,
-  pub balance_data: [T; 12],
+  pub balance_data: [T; 32],
```

Discussion

CergyK

Fix looks good

Issue M-2: All block chunks not incorporated in smt_update when blk_rlp_fields_count < end_blk_rlp_fields_count

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/86

Description

During an `smt_update` proof, the parent block rlp and end block rlp are incorporated simultaneously. Unfortunately the incorporation stops at the length of `blk_rlp_fields_count` which is the length of the parent block. This means that the latest bytes of the end block may not be ingested.

smt_update/generation.rs#L60-L75:

```
if i == 0 {
    let blk_rlp_len = parent_block_rlp_bytes.len();
    let blk_rlp_fields_count = if blk_rlp_len % BLK_RLP_FIELD_WIDTH == 0 {
        blk_rlp_len / BLK_RLP_FIELD_WIDTH
    } else {
        blk_rlp_len / BLK_RLP_FIELD_WIDTH + 1
    };


    let end_blk_rlp_len = end_block_rlp_bytes.len();
    let end_blk_rlp_fields_count = if end_blk_rlp_len % BLK_RLP_FIELD_WIDTH == 0 {
        end_blk_rlp_len / BLK_RLP_FIELD_WIDTH
    } else {
        end_blk_rlp_len / BLK_RLP_FIELD_WIDTH + 1
    };


    for j in 0..blk_rlp_fields_count - 1 {
        ... // ingest bytes from both end_block_rlp_bytes and parent_block_rlp_bytes
```

Impact

This is a completeness issue affecting the generation of the proof. The impact is probably low, because once the bug is encountered fixing it is trivial.

Recommendation

Take the max between `blk_rlp_fields_count` and `end_blk_rlp_fields_count` and iterate over that to create the rows:

smt_update/generation.rs#L60-L75:

```
    if i == 0 {
        let blk_rlp_len = parent_block_rlp_bytes.len();
        let blk_rlp_fields_count = if blk_rlp_len % BLK_RLP_FIELD_WIDTH == 0 {
            blk_rlp_len / BLK_RLP_FIELD_WIDTH
        } else {
            blk_rlp_len / BLK_RLP_FIELD_WIDTH + 1
        };


        let end_blk_rlp_len = end_block_rlp_bytes.len();
        let end_blk_rlp_fields_count = if end_blk_rlp_len % BLK_RLP_FIELD_WIDTH ==
 ↪  0 {
            end_blk_rlp_len / BLK_RLP_FIELD_WIDTH
        } else {
            end_blk_rlp_len / BLK_RLP_FIELD_WIDTH + 1
        };

-       for j in 0..blk_rlp_fields_count - 1 {
+       let blk_rlp_fields_to_ingest = max(blk_rlp_fields_count,
 ↪  end_block_rlp_bytes);
+       for j in 0..blk_rlp_fields_to_ingest - 1 {
            ... // ingest bytes from both end_block_rlp_bytes and
 ↪  parent_block_rlp_bytes
```

Discussion

bytetang

fix commit: 86ef0430eeea140f97d97ddf07055b6c54e99c32

CergyK

Fix looks good

Issue M-3: First two directions are not constrained properly in smt_anchor.rs

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/87

Description

In smt_anchor row generation we can see that the first two directions are hardcoded:

smt_anchor/generation.rs#L61-L86:

```
    let mut row0 = generate_extract_trace_row(c, row_idx,
↪   end_block_hash_bytes_u64.clone(), next_chunk_merkle_root_u64.clone());
    row0.is_first_row = F::ONE;
>>  row0.left = F::ONE;
    let mut node_hash: [u64; 8] = [0; 8];
    node_hash.iter_mut().enumerate()
        .for_each(|(i, r)| if i < 4 { *r = end_block_hash_bytes_u64[i] } else { *r
↪   = next_chunk_merkle_root_u64[i - 4] });


    // poseidon2(end_blk_hash,next_chunk_merkle_root) = next_chunk_root
    state.push_poseidon2_event(Poseidon2SpongeEvent{input: node_hash});
    let hash = hl_poseidon2(node_hash).to_vec();
    row0.hash.iter_mut()
        .enumerate()
        .for_each(|(i, r)| *r = F::from_noncanonical_u64(hash[i] as u64));
    rows.push(row0);
    row_idx += 1;


    let mut node = hash.clone();


    let mut row1 = generate_extract_trace_row(c, row_idx, node.clone(),
↪   vec![0,0,0,0]);
    // row1.is_extract_row = F::ONE;
>>  row1.right = F::ONE;
    let mut node_hash: [u64; 8] = [0; 8];
    node_hash.iter_mut().enumerate()
        .for_each(|(i, r)| if i < 4 { *r = 0 } else { *r = node[i - 4] });

    // poseidon2(0x00000..00, next_chunk_root) = chunk_root
```

However this is not constrained, and a malicious prover can thus change the order in which the provided nodes are hashed, not respecting the format during the insertion.

Impact

As a result of this manipulation, a standard prover may not be able to prove the inclusion into smt, unless also switching the order of hashing to match the insertion order.

Recommendation

Add special flags for this phase (`chunk_merkle_root` creation), and ensure the expected standard order is respected during insertion

Discussion

CergyK

The #79 issue type applies to `is_first_row`; after first row of the trace, the prover can always set `is_first_row` to zero

bytetang

@CergyK Added a new lookup to check the remaining rows order: https://github.com/brevis-network/brevis-core/pull/150/files

CergyK

@bytetang ok the fix looks good now

Issue M-4: extension_value_big_len is missing range check to check valid bytes

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/92

Description

Decoded rlp big lengths should be range checked for the following constraint to indeed constraint that all bytes are zero:

```
yield_constr.constraint(
    (P::ONES - local.is_ext_value_long_string - local.is_ext_value_long_array)
        * (local.extension_value_big_len[0]
            + local.extension_value_big_len[1]
            + local.extension_value_big_len[2]),
);
```

As is done currently for `big_len`:

storage/storage_stark.rs#L355-L366:

```
pub fn big_len_ctl_upper_bound_columns(&self, offset: usize) -> Vec<Column<F>> {
    let col= Column::<F>::single(offset_of!(StorageColumnsView<u8>, big_len) +
↪   offset);
    vec![Column::constant(F::ONE), col, Column::constant(F::from_canonical_u8(255))]
}

pub fn big_len_ctl_lower_bound_columns(&self, offset: usize) -> Vec<Column<F>> {
    let col= Column::<F>::single(offset_of!(StorageColumnsView<u8>, big_len) +
↪   offset);
    vec![Column::constant(F::ONE), Column::constant(F::from_canonical_u32(0)), col]
}
pub fn big_len_leu_filter(&self) -> Filter<F> {
    Filter::new_simple(Column::single(offset_of!(StorageColumnsView<u8>, enable)))
}
```

Recommendation

Marked medium-unimportant for now as this would actually be fixed if we enforce unused arrays to be zero via lookup as suggested in #75

Discussion

CergyK

Fix looks good, lookups introduced in:

account_stark.rs multiple_stark.rs multiple_stark.rs multiple_stark.rs storage_stark.rs

Issue M-5: non-existence should be provable on a branch node in storage MPT

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/94

Description

The coprocessor uses a specific flag non_exist_extension, to mark a row relative to a non existence check on an extension. Unfortunately non-existence can only be proven when the extension node does not match the remaining `slot_hash` path. If the path diverges on a branch node, meaning that the current key path nibble holds a 0x80 (empty) value in the branch, non-existence of the storage value cannot be proven.

Impact

This is a completeness issue which will prevent some proofs to be produced when the edge case is triggered

Recommendation

Introduce a `non_exist_branch` flag, which will ensure that the value 0x80 is present as the element at the current key path nibble in the branch

Discussion

CergyK

Not sure I understand why last_row condition has to be removed here @grrThea @bytetang?

The inner lookup should not be happening in this case right?

bytetang

@CergyK In order to fix the non-existence proving issue, we need to prove that the MPT key has been included in the last node and that the value must be an empty one. Removed the last row condition here to make sure the the last row key path can be lookup and selected correctly.

However, this might import another problem, The last row MPT proof inclusion check might fail due to it uses the same filter. we are considering adding an extra flag as a filter to make sure the last row MPT proof inclusion can be looked up correctly. The flag, which indicates the last node within a chunk, is provable.

CergyK

@bytetang I think I understand, the row having the current nibble cannot be the last stark row (is_last_proof) for the chunk because the whole branch (17 elts) need to be ingested in keccak to prove inclusion.

However anyhow the inner lookup should not be made in the branch non-existence case (when branch_key_path == element_idx). Adding a new flag sounds reasonable

grrThea

Here is the fixed pr. Please help confim. @CergyK

Issue M-6: Leaves having less than 32 bytes would be inlined in ethereum MPT

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/95

Context

Ethereum MPTs all implement an edge case optimisation which consist to inline tree nodes which are strictly less than 32 bytes. In most cases this should not happen, but can happen and be setup intentionally in these 2 cases:

- storage leaf
- transaction leaf (although very unlikely for a successful transaction as it requires empty "data" and empty "to" fields)

(account leaves, receipt leaves should be too big for it to ever happen)

As a reference here is the implementation of the trie encoding in go-ethereum: trie/node_enc.go#L43-L55:

```
func (n *fullnodeEncoder) encode(w rlp.EncoderBuffer) {
  offset := w.List()
  for _, c := range n.Children {
      if c == nil {
          w.Write(rlp.EmptyString)
      } else if len(c) < 32 {
          w.Write(c) // rawNode
      } else {
          w.WriteBytes(c) // hashNode
      }
  }
  w.ListEnd(offset)
}
```

Alternatively, if the node representation is bigger than 32 bytes, only the hash of it is stored as child of current tree node.

Description

Unfortunately the coprocessor code always assumes that the child of current node is a hash if we are not finished processing the whole path. This means that the inlined child will be falsely interpreted as a hash instead of it being raw-data.

Impact

It seems unlikely that a prover will be able to prove false information due to this bug (indeed a large chunk of the leaf data would be the remainder of the path, which is a substring of the hash of the storage slot). In some edge cases it would however be impossible to prove the value of a storage slot, which makes it a completeness issue

Recommendation

Some logic should be added to handle this edge case gracefully

Discussion

bytetang

@CergyK Found this go-ethereum code trie/node_enc.go#L43-L55 is only used to RLP encode MPT branch node. But the issue description is storage leaf /account leaf seems not relevant to the MPT branch node.

CergyK

> @CergyK Found this go-ethereum code trie/node_enc.go#L43-L55 is only used to RLP encode MPT branch node. But the issue description is storage leaf /account leaf seems not relevant to the MPT branch node.

Indeed, the issue is about how children of a branch node are encoded, so in this case leaves would be encoded using either `leafNodeEncoder` or `hashNode` IIRC. The mentioned code is about how the value is inlined when it is < 32 bytes and stored as hash when == 32 bytes

bytetang

Since this is a completeness issue and it only occurs in rare scenarios, the system decides not to support the scenario where leaves are less than 32 bytes in storage and transaction MPT proofs. In the prover at the beginning of the proof - starting stage, a panic is chosen instead of proceeding with the proof. Also, this will be stated in the ”Limitations” section of the official documentation.
https://github.com/brevis-network/brevis-core/commit/d3536d2a56dbfeaef4f313a4b861b79dc5452fe5

Issue L-1: Redundant IsEqual function

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/96

This issue has been acknowledged by the team but won't be fixed at this time.

Description

In `brevis-core/gnark/hash2hash/circuits/common_circuit.go`, the function `IsEqual` is redundant. Function `isEqual` with identical logic was already present and utilized internally by the `AssertVk` function.

Issue L-2: Incorrect Comparisons in AssertVk function

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/97

Description

The `AssertVk` function is intended to compare two `regroth16.VerifyingKey` structures field by field and return a variable that aggregates the results of these comparisons. However, several loops within the function contain incorrect field comparisons for the `CommitmentKey` portion of the verifying keys.

Bug Details

The following sections of the `AssertVk` function exhibit incorrect comparisons:

1. **Comparison of `CommitmentKey.G.P.Y` limbs:**
    - The loop iterating over `vk1.CommitmentKey.G.P.Y.A0.Limbs` compares each element with `vk2.CommitmentKey.G.P.X.A0.Limbs[i]` instead of `vk2.CommitmentKey.G.P.Y.A0.Limbs[i]`.

```
// ... existing code ...
for i, _ := range vk1.CommitmentKey.G.P.Y.A0.Limbs {
    res = api.Add(res, isEqual(api, vk1.CommitmentKey.G.P.Y.A0.Limbs[i],
↪   vk2.CommitmentKey.G.P.X.A0.Limbs[i])) // Should be
↪   vk2.CommitmentKey.G.P.Y.A0.Limbs[i]
}

for i, _ := range vk1.CommitmentKey.G.P.Y.A1.Limbs {
    res = api.Add(res, isEqual(api, vk1.CommitmentKey.G.P.Y.A1.Limbs[i],
↪   vk2.CommitmentKey.G.P.X.A1.Limbs[i])) // Should be
↪   vk2.CommitmentKey.G.P.Y.A1.Limbs[i]
}
// ... existing code ...
```

    - The loop iterating over `vk1.CommitmentKey.G.P.Y.A1.Limbs` compares each element with `vk2.CommitmentKey.G.P.X.A1.Limbs[i]` instead of `vk2.CommitmentKey.G.P.Y.A1.Limbs[i]`.
2. **Comparison of `CommitmentKey.GRootSigmaNeg.P.X` limbs:**
    - The loop iterating over `vk1.CommitmentKey.GRootSigmaNeg.P.X.A0.Limbs` compares each element with `vk2.CommitmentKey.G.P.X.A0.Limbs[i]` instead of `vk2.CommitmentKey.GRootSigmaNeg.P.X.A0.Limbs[i]`.

```
// ... existing code ...
for i, _ := range vk1.CommitmentKey.GRootSigmaNeg.P.X.A0.Limbs {
```

```
    res = api.Add(res, isEqual(api,
 ↪  vk1.CommitmentKey.GRootSigmaNeg.P.X.A0.Limbs[i],
 ↪  vk2.CommitmentKey.G.P.X.A0.Limbs[i])) // Should be
 ↪  vk2.CommitmentKey.GRootSigmaNeg.P.X.A0.Limbs[i]
}

for i, _ := range vk1.CommitmentKey.GRootSigmaNeg.P.X.A1.Limbs {
    res = api.Add(res, isEqual(api,
 ↪  vk1.CommitmentKey.GRootSigmaNeg.P.X.A1.Limbs[i],
 ↪  vk2.CommitmentKey.G.P.X.A1.Limbs[i])) // Should be
 ↪  vk2.CommitmentKey.GRootSigmaNeg.P.X.A1.Limbs[i]
}
// ... existing code ...
```

- The loop iterating over `vk1.CommitmentKey.GRootSigmaNeg.P.X.A1.Limbs` compares each element with `vk2.CommitmentKey.G.P.X.A1.Limbs[i]` instead of `vk2.CommitmentKey.GRootSigmaNeg.P.X.A1.Limbs[i]`.

3. **Comparison of `CommitmentKey.GRootSigmaNeg.P.Y` limbs:**

    - The loop iterating over `vk1.CommitmentKey.GRootSigmaNeg.P.Y.A0.Limbs` compares each element with `vk2.CommitmentKey.G.P.X.A0.Limbs[i]` instead of `vk2.CommitmentKey.GRootSigmaNeg.P.Y.A0.Limbs[i]`.

```
// ... existing code ...
for i, _ := range vk1.CommitmentKey.GRootSigmaNeg.P.Y.A0.Limbs {
    res = api.Add(res, isEqual(api,
 ↪  vk1.CommitmentKey.GRootSigmaNeg.P.Y.A0.Limbs[i],
 ↪  vk2.CommitmentKey.G.P.X.A0.Limbs[i])) // Should be
 ↪  vk2.CommitmentKey.GRootSigmaNeg.P.Y.A0.Limbs[i]
}

for i, _ := range vk1.CommitmentKey.GRootSigmaNeg.P.Y.A1.Limbs {
    res = api.Add(res, isEqual(api,
 ↪  vk1.CommitmentKey.GRootSigmaNeg.P.Y.A1.Limbs[i],
 ↪  vk2.CommitmentKey.G.P.X.A1.Limbs[i])) // Should be
 ↪  vk2.CommitmentKey.GRootSigmaNeg.P.Y.A1.Limbs[i]
}
// ... existing code ...
```

- The loop iterating over `vk1.CommitmentKey.GRootSigmaNeg.P.Y.A1.Limbs` compares each element with `vk2.CommitmentKey.G.P.X.A1.Limbs[i]` instead of `vk2.CommitmentKey.GRootSigmaNeg.P.Y.A1.Limbs[i]`.

This means that `G.P.X.A0.Limbs` and `G.P.X.A1.Limbs` from `vk2` are being compared against multiple different fields of `vk1`, leading to an incorrect overall comparison result.

The function is deprecated, so there is no direct impact on current usage. However, if this function were to be reused or referenced in the future, these errors could lead to incorrect verification key comparisons.

Issue L-3: Renaming variable suggestions

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/101

This issue has been acknowledged by the team but won't be fixed at this time.

Description

The following variables can be renamed for more clarity:

- `chunk_idx` can be renamed to `slice_idx`, to avoid confusion with `row.chunk_idx`

https://github.com/brevis-network/brevis-core/blob/dev/starky/src/transaction/receipt/generation.rs#L10-L17

```
pub fn generate_receipt_status_trace_rows2<F: RichField + Extendable<D>, const
 ↪  D: usize>(
        mut rows: Vec<ReceiptStatusDecodeCols<F>>,
        inputs: Vec<u64>,
        index: usize,
>>      chunk_idx: usize) -> Vec<ReceiptStatusDecodeCols<F>> {

    let mut row = ReceiptStatusDecodeCols::<F>::default();
>>  row.chunk_idx = F::from_canonical_u64(index as u64);
```

- `extension_elements_*`, `extension_key_*`, could be renamed to `extension_or_leaf_*` in mpt starks, since they are used for holding both extension and leaf data
- event_idx should be renamed to event_id in `PubInputLookupColumns`:

logs/pub_input_stark.rs#L28:

```
pub event_idx: [T; 6],
```

- `verify_smt_proofs` not only verifies smt proofs, but also makes an aggregated proof. Please consider renaming to `aggregate_smt_proofs`

src/verifier.rs#L25-L29:

```
pub fn verify_smt_proofs<F, InnerC, OuterC, const D: usize>(
    blocks_proof: &AggChild<F, InnerC, D>,
    smt_update_proof: &AggChild<F, InnerC, D>,
    block_chunk_merkle_proof: &AggChild<F, InnerC, D>,
) -> anyhow::Result<Proof<F, OuterC, D>>
```

- `start_block_hash` to `parent_block_hash`:

block_syncer/src/smt_update/util_sdk.rs#L8:

```
let start_block_hash: Vec<F> = hex::decode(smp_update_chunk.parent_block_hash.trim_
↪ start_matches("0x")).unwrap().iter().map(|&v| F::from_canonical_u64(v as
↪ u64)).collect();
```

- `is_chunk_end` to `is_block_rlp_end` (chunk is already used to designate a whole data point proof in multiple places):

blocks/chunk_block.rs#L38:

```
pub is_chunk_end: T,
```

Discussion

bytetang

It's partially optimized.

Issue L-4: Generation path for extension/leaf always generates 17 rows (same as branch)

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/102

This issue has been acknowledged by the team but won't be fixed at this time.

Description

Generally 17 rows are needed to process a branch of a MPT, because it contains 17 elements. However due to a lack of optimisation, the generation path for extension/leaf type nodes also generates 17 rows, when only one is needed.

transaction/mpt/generation.rs#L339-L340:

```
let max_fields_num = ext_fields_num.max(branch_fields_num);
if element_idx == max_fields_num - 1 {
```

> Here the example is taken from transaction mpt row generation, but it is applicable to all mpt row generations.

Recommendation

It should be safe to use the right number according to node type (`ext_field_num` if extension, and `branch_fields_num` if branch)

Discussion

grrThea

We just set a maximum and safe number of rows to store mpt_key_nibble, because the number of mpt_key_nibbles can be larger than ext_field_num.

Issue L-5: extension_value cannot have len_len because it is lower than 32 bytes

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/103

This issue has been acknowledged by the team but won't be fixed at this time.

Description

In the coprocessor RLP decoding of extension type nodes, a `len_len` is often included for completeness. However including `len_len` is unecessary, because we can be sure that the `extension_value` length is always less than 33 bytes as per ethereum spec.

storage/generation.rs#L287-L291:

```
//@audit ext value cannot be long string or long array
row.is_ext_value_long_string = if is_long_string { F::ONE } else { F::ZERO };
row.is_ext_value_long_array = if is_long_array { F::ONE } else { F::ZERO };


let (len_cols, len_byte_val) = parse_rlp_len(&res[1], len_len, is_big);
let payload_len = if is_big { len_byte_val } else { array_len };
```

Recommendation

The handling of big length for extension value and leaf value can be removed, since we know that the data is less than 33 bytes.

Discussion

grrThea

We have renamed the variable to is_extension_or_leaf_value_long_string because it actually applies to both extension and leaf values.

Issue L-6: Unused columns/variables

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/105

This issue has been acknowledged by the team but won't be fixed at this time.

Description

The following columns look unused and can be removed:

- `key_path_idx` in `StorageColumnsView`:

storage/column.rs#L219:

```
pub key_path_idx: T,
```

- `is_chunk_start` in `ChunkHashColumnsView`:

smt/column.rs#L69:

```
pub is_chunk_end: T,
```

- input_padding_div logic as defined in `receipt/mpt/mpt_generation.rs`:

receipt/mpt/mpt_generation.rs#L91-L94:

```
let input_padding_div = (PREIMAGE_SIZE_U8 - input_rem) / PREIMAGE_SIZE_U8;
let input_padding_rem = (PREIMAGE_SIZE_U8 - input_rem) % PREIMAGE_SIZE_U8;
row.input_padding_div = F::from_noncanonical_u64(input_padding_div as u64);
row.input_padding_rem = F::from_noncanonical_u64(input_padding_rem as u64);
```

- `local.is_elements_end` is unecessary here:

transaction/mpt/mpt_stark.rs#L629:

```
yield_constr.constraint_transition((P::ONES - local.is_leaf_elements_end) *
↪  local.is_elements_end * (next.chunk_idx - local.chunk_idx));
```

- Public inputs are unused in `account_leaf_stark`:

storage/account/prove_with_ctl.rs#L307:

```
(AccountLeafDecoder, account_leaf_stark, public_inputs),
```

- `last_proof_elements_end_sum` is always equal to `is_last_proof_elements_end` as long as there is no consecutive `is_last_proof_elements_end` rows and thus can be removed

storage/storage_stark.rs#L929-L934:

```
// constrain: is_last_proof_elements_end sum == 1
yield_constr.constraint_first_row(local.last_proof_elements_end_sum -
↪  local.is_last_proof_elements_end);
yield_constr.constraint_transition(local.is_last_proof_elements_end *
↪  (next.last_proof_elements_end_sum - next.is_last_proof_elements_end));
let next_leaf_elements_end_sum = local.last_proof_elements_end_sum +
↪  next.is_last_proof_elements_end;
yield_constr.constraint_transition((P::ONES - local.is_last_proof_elements_end) *
↪  (next_leaf_elements_end_sum - next.last_proof_elements_end_sum));
yield_constr.constraint(local.is_last_proof_elements_end *
↪  (local.last_proof_elements_end_sum - P::ONES));
```

- Block header doesn't need 2 u64 limbs for `basefee` (1 is well enough):

smt/column.rs#L182-L183:

```
pub basefee: [T; 16],
pub basefee_u64: [T; 2],
```

- branch_idxs is only used for generation in `storage_stark`

starky/src/storage/column.rs#L27:

```
pub branch_idxs: T,
```

Discussion

grrThea

partial fix

Issue L-7: inner_stark.rs logic is duplicated across all MPTs

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/106

This issue has been acknowledged by the team but won't be fixed at this time.

Description

All MPT inclusion starks make use of inner-lookups to break the traversal of the MPT into multiple rows/steps. We can notice that all of the minimal chips handling the inner-lookup have a very minimal logic, which is highly similar:

- starky/src/storage/storage/inner_stark.rs
- starky/src/transaction/mpt/inner_stark.rs
- starky/src/receipt/mpt/inner_stark.rs
- starky/src/storage/account/inner_stark.rs

Recommendation

It seems that it would improve the robustness and quality of the codebase to move this component into a common `inner_stark` in `starky/gadgets/` for example

Issue L-8: Some tests are failing on the audit branch

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/107

Description

Some tests, as defined in the scope of the audit are currently failing. While this does not necessarily demonstrate bugs in the proving code or circuits, this reduces robustness and makes code analysis harder. Please consider fixing the following tests:

- `smt_update::smt_update_stark::test::test_smt_update`:

```
---- smt_update::smt_update_stark::test::test_smt_update stdout ----
thread 'smt_update::smt_update_stark::test::test_smt_update' panicked at
↪  block_syncer/src/prover.rs:101:90:
called `Result::unwrap()` on an `Err` value: Cross-table lookup 1 verification
↪  failed. looked: 4, looking: [0, 2], 3579237600320740592 17183472645427181126
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Discussion

CergyK

Even though the tests are failing when run all together:

```
FAST_MODE=1 CASE=0 RUST_LOG=INFO cargo test --package block-syncer --lib --release


output:
---- prover::test::test_smt_witness stdout ----
thread 'prover::test::test_smt_witness' panicked at /Users/cergyk/.cargo/registry/s
↪  rc/index.crates.io-1949cf8c6b5b557f/env_logger-0.11.5/src/logger.rs:944:10:
env_logger::init_from_env should not be called after logger initialized:
↪  SetLoggerError(())

---- blocks::prover::test::prove_blocks stdout ----
thread 'blocks::prover::test::prove_blocks' panicked at /Users/cergyk/.cargo/regist
↪  ry/src/index.crates.io-1949cf8c6b5b557f/env_logger-0.11.5/src/logger.rs:944:10:
env_logger::init_from_env should not be called after logger initialized:
↪  SetLoggerError(())

---- smt_update::smt_update_stark::test::test_smt_update stdout ----
thread 'smt_update::smt_update_stark::test::test_smt_update' panicked at
↪  /Users/cergyk/.cargo/registry/src/index.crates.io-1949cf8c6b5b557f/env_logger-0
↪  .11.5/src/logger.rs:944:10:
env_logger::init_from_env should not be called after logger initialized:
↪  SetLoggerError(())
```

```
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

They are succeeding individually

Issue L-9: chunk_idx increase constraint should be constraint_transition in storage_stark

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/112

Description

Some constraints do not hold at the last row of the trace, because next row is the first one (wrap around). This is the case for the constraint enforcing `chunk_idx` increase in `storage_stark`:

storage/storage_stark.rs#L946-L947:

```
// constrain: chunk_idx gets incremental crossing boundary
yield_constr.constraint(local.is_last_proof_elements_end.mul(chunk_idx_inc -
↪  P::ONES));
```

This is a small completeness issue, in the case where the last row of the trace is `local.is_last_proof_elements_end`, this constraint would fail. This is quite unlikely as it would mean the whole `enabled` trace is a power of two, and does not require padding rows.

Recommendation

Make the constraint `constraint_transition`:

storage/storage_stark.rs#L946-L947:

```
    // constrain: chunk_idx gets incremental crossing boundary
-    yield_constr.constraint(local.is_last_proof_elements_end.mul(chunk_idx_inc -
↪  P::ONES));
+    yield_constr.constraint_transition(local.is_last_proof_elements_end.mul(chunk_
↪  idx_inc - P::ONES));
```

Issue L-10: ext_node_mpt_key_nibble is unconstrained with regards to extension key elements when node is leaf

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/113

Description

`ext_node_mpt_key_nibble` sequence should be constrained to be equal to nibbles of the extension key path, which is correctly constrained by the following:

transaction/mpt/mpt_stark.rs#L681-L684:

```
>>  yield_constr.constraint(local.is_extension.mul(local.ext_node_mpt_key_nibble_fl
↪   ag - local.mpt_key_nibble_flag));
    let key_path_nibbles:&[P] = &local.key_path_nibbles.map(|b| b.into());
    let ext_node_mpt_key_u64_cal_from_nibbles =
↪   hex_array_to_u64_p(key_path_nibbles.to_vec());
    yield_constr.constraint(local.is_ext_elements_end.mul(p2_local.ext_node_mpt_key
↪   _u64 - ext_node_mpt_key_u64_cal_from_nibbles));
```

receipt/mpt/mpt_stark.rs#L774-L777:

```
>>  yield_constr.constraint(local.is_extension.mul(local.ext_node_mpt_key_nibble_fl
↪   ag - local.mpt_key_nibble_flag));
    let key_path_nibbles:&[P] = &local.key_path_nibbles.map(|b| b.into());
    let ext_node_mpt_key_u64_cal_from_nibbles =
↪   hex_array_to_u64_p(key_path_nibbles.to_vec());
    yield_constr.constraint(local.is_ext_elements_end.mul(p2_local.ext_node_mpt_key
↪   _u64 - ext_node_mpt_key_u64_cal_from_nibbles));
```

However nodes of type `is_leaf` need to be constrained exactly in the same way; the constraints need to be adapted to include leaves as well.

Recommendation

Leaves should be included in these constraints:

transaction/mpt/mpt_stark.rs#L681-L684:

```
-   yield_constr.constraint(local.is_extension.mul(local.ext_node_mpt_key_nibble_fl
↪   ag - local.mpt_key_nibble_flag));
+   yield_constr.constraint((local.is_extension +
↪   local.is_leaf).mul(local.ext_node_mpt_key_nibble_flag -
↪   local.mpt_key_nibble_flag));
    let key_path_nibbles:&[P] = &local.key_path_nibbles.map(|b| b.into());
```

```
    let ext_node_mpt_key_u64_cal_from_nibbles =
↪   hex_array_to_u64_p(key_path_nibbles.to_vec());
    yield_constr.constraint(local.is_ext_elements_end.mul(p2_local.ext_node_mpt_key
↪   _u64 - ext_node_mpt_key_u64_cal_from_nibbles));
```

receipt/mpt/mpt_stark.rs#L774-L777:

```
-   yield_constr.constraint(local.is_extension.mul(local.ext_node_mpt_key_nibble_fl
↪   ag - local.mpt_key_nibble_flag));
+   yield_constr.constraint((local.is_extension +
↪   local.is_leaf).mul(local.ext_node_mpt_key_nibble_flag -
↪   local.mpt_key_nibble_flag));
    let key_path_nibbles:&[P] = &local.key_path_nibbles.map(|b| b.into());
    let ext_node_mpt_key_u64_cal_from_nibbles =
↪   hex_array_to_u64_p(key_path_nibbles.to_vec());
    yield_constr.constraint(local.is_ext_elements_end.mul(p2_local.ext_node_mpt_key
↪   _u64 - ext_node_mpt_key_u64_cal_from_nibbles));
```

> Other constraints only enforced for extension type nodes should be enforced for leaves: account/account_stark.rs#L711-L714:
>
> ```
> // constrain extension_branch_values = local.extension_elements[-33..] (mpt hash)
> for i in 0..32 {
>     yield_constr.constraint(local.is_extension.mul(local.branch_values[i+1] -
> ↪   local.extension_elements[i]));
> }
> ```

Discussion

grrThea

The ext_node_mpt_key_nibble_flag is only applicable to extension nodes; leaf nodes do not have an mpt_key.

grrThea

Because the receipt `key_path` is the index RLP of a tx in block. the whole key can present the location of MPT leaf. there is no remaining key path need to mark by flags.

CergyK

> Because the receipt `key_path` is the index RLP of a tx in block. the whole key can present the location of MPT leaf. there is no remaining key path need to mark by flags.

@grrThea Not sure I understand this, but this is not true in the general case. Take for example the following (tx_hash, block_hash): (`0xdd1abce12a14156863f4664ce8a90ab2ec721338545f6357f4ffd6b959da79eb`, `0x6f11e9ea6df74feab3b25ab03a43a61a3e92404bd5be43aea46a3c3c22f93481`)

Its proof has the leaf:

```
["30","f9010801825208b90100000000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  00000000000000000000000000000000000000000000000000000000000000c0"]
```

meaning that the leaf key has 1 nibble equal to zero.

bytetang

@CergyK Assume there are 20 receipts in the block, the keys for these 20 receipts should be RLP_encode(index_of_tx), with the range being [0, 19]. Presented by hex is 0x00...0x14

```
0xc100
0xc101
0xc102
...
0xc10f
0xc111
0xc112
0xc113
0xc114
```

Unlike the storage/account, the keys are from hash. Since the receipts key numbers are sequential. In general, the keys will terminated at the branch and point to a leaf to store the final value.

CergyK

@bytetang Yes totally agree, there are some edge cases where the keys are not terminated on a branch however, as demonstrated in my point above.

A frequent example, is the tx at index 0 which would have 0x80 as the key, of which 8 would be on the branch and 0 as part of the leaf

CergyK

As discussed later, this does not have an impact, as it is the position of the leaf we are intersted in uniquely identifying here. Downgrading to low and won't fix

Issue L-11: PathStark is redundant with path_acc, mpt_key_u64 checks

Source: https://github.com/sherlock-audit/2025-04-brevis-network/issues/114

This issue has been acknowledged by the team but won't be fixed at this time.

Description

PathStark is introduced in the coprocessor in order to check that extension/leaf paths are correct. However it seems that the decoding correctness is already enforced for account, storage by the `path_acc` check:

account/account_stark.rs#L770-L773:

```
// constrain: slot_nibbles_rlc = key_path_acc
let address_nibble:&[P] = &local.address_nibble.map(|b| b.into());
let address_nibbles_rlc = bytes_rlc_p(&address_nibble, gamma.clone());
yield_constr.constraint(local.is_leaf_elements_end.mul(p2_local.path_acc -
↪  address_nibbles_rlc));
```

In transactions and receipt mpts, the same is enforced by a check on `mpt_key_u64`: accumulation of nibbles from the branch/extension/leaf nodes yield the `mpt_key_u64` publicly exposed input

Recommendation

It seems `path_stark.rs` and corresponding cross-table lookups can be removed to simplify overall architecture.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.

Additional analysis of the findings and design considerations:

General Health

Some issues reported during this review have been labeled as "General Health" issues and would need deeper attention (accross the whole codebase) to properly be addressed

RLP decoding strictness

Multiple instances of RLP decoding malleability have been detected and can be broadly categorized as follows:

1. Overlap of array paddings (#75)
2. Providing RLC fragments out of order (#82)
3. RLP prefixes and RLC exponents are not strongly constrained (#84)

Cross table lookups

When handling cross table lookups (CTL in short) a careful consideration must be given to multiplicities used (especially on the looking tables side). In many occurences, it has been found that important checks are lacking on the multiplicities (`filter` columns) as reported in issue #76

State transition soundness

Some starks are enforcing state transition through relationships between `local` and `next` rows of the trace. In that case the transitions must be strict enough to not enable bypassing some essential constraints.

> Due to the need of having an `enable` column to indicate non-padding rows (padding is required to have traces of length being a power of two), there is an additional 'hidden' state which should be accounted for. This yielded the finding #77.

Some columns are used as state flags to bypass constraints when some "state" is not currently active. When the flag is never constrained to become true, a prover can set the flag to zero for all rows of the trace to bypass the constraints (#79)

Code duplication

The code would benefit from unduplicating some components. Most notably the Merkle Patricia Tries for the block commitments (`transactionRoot`, `receiptsRoot`, `accountsRoot`, `storageRoot`) all have separate implementations, but are heavily similar.

Some of the reported findings (see #98), could have been prevented by having a common implementation for all MPTs.

> It is to note however that unduplicating the MPT implementation is not entirely trivial either;
>
> - (transaction, receipt) MPT have a different path format (RLP encoding of index) used than (storage, account) hash of key
> - storage MPT has a non-existence argument