MatterLabs - zkSync Era

Zero Knowledge Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 9th, 2023 - March 8th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/05/2023	Omar Alshaeb
0.2	Document Updates	03/08/2023	Omar Alshaeb
0.3	Draft Review	03/08/2023	Gokberk Gulgun
0.4	Draft Review	03/09/2023	Gabi Urrutia
0.5	Document Updates	04/03/2023	Omar Alshaeb
0.6	Document Draft Review	04/03/2023	Gokberk Gulgun
0.7	Document Draft Review	04/03/2023	Gabi Urrutia
1.0	Remediation Plan	04/04/2023	Omar Alshaeb
1.1	Remediation Plan Review	04/04/2023	Gokberk Gulgun
1.2	Remediation Plan Review	04/04/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Omar Alshaeb	Halborn	Omar.Alshaeb@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

MatterLabs zkSync Era is a Layer 2 blockchain protocol that eliminates Ethereum's inherent congestion with zero knowledge proofs. Matter Labs' creation is on a mission to accelerate the mass adoption of crypto for personal sovereignty. It is designed to unlock the full potential of trustless blockchain technology while scaling the core values of Ethereum.

MatterLabs engaged Halborn to conduct a security audit on their zero knowledge circuits and the verifier, beginning on January 9th, 2023 and ending on March 8th, 2023. The security assessment was scoped to the circuits provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two months for the engagement and assigned a full-time security engineer to audit the security of the zero knowledge circuits and the verifier. The security engineer is a blockchain, smart-contract and ZK security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that the circuits operate as intended.
- Identify potential security issues within the circuits.
- Ensure that the verifier operate as intended.
- Identify potential security issues within the verifier contracts.

In summary, Halborn identified some security risks that were mostly addressed by the MatterLabs team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Circuit manual code review and walkthrough.
- Graphing out functionality and circuit logic/connectivity/functions. (cargo-deps)
- Manual code review of common Rust security vulnerabilities.
- Manual code review of specific zero knowledge security vulnerabilities.
- Scanning of circuits files for unsafe Rust code usage. (cargo-geiger)
- Static Analysis of security for scoped circuits, and imported functions. (cargo-audit)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. IN-SCOPE:

The security assessment was scoped to the following zero knowledge circuits:

- /src/vm/*
- /src/glue/sort_decommitments_requests/*
- /src/glue/code_unpacker_sha256/*
- /src/glue/demux_log_queue/*
- /src/precompiles/keccak256.rs
- /src/precompiles/sha256.rs
- /src/glue/ecrecover_circuit/*
- /src/glue/ram_permutation/*
- /src/glue/storage_validity_by_grand_product/*
- /src/glue/storage_application/*
- /src/glue/pub_data_hasher/*
- /src/glue/log_sorter/*
- /src/glue/merkleize_l1_messages/*
- /src/scheduler/*
- /src/circuit_structures/*
- /src/data_structures/*
- /src/inputs/*
- /src/recursion/*
- /src/traits/*
- /src/secp256k1/*
- /src/utils.rs

Commit ID : 014e674916058e31725d6e92439fa5ff14e6677e

And the verifier:

- /zksync/Verifier.sol
- /zksync/Plonk4VerifierWithAccessToDNext.sol
- /zksync/libraries/PairingsBn254.sol
- /zksync/libraries/TranscriptLib.sol

Commit ID : fc7e86a3df404acb88d86502c944c0630a7ed288

2. REMEDIATION PR/COMMITS:

- Fix Commit ID (HAL-01): 5109e0768c7de799f87ec67bf40b6a544cca4e4e
- Fix Commit ID (HAL-02): b0a79356613655bddccaab3b89dbf1142b5483fb
- Fix Commit ID (HAL-03): 06c2e76546369fb112d8ac14fb5388154857435b

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	0	1	3

LIKELIHOOD

IMPACT

				(HAL-01)
(HAL-02)				
(HAL-03) (HAL-04) (HAL-05)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - CIRCUIT NOT PROPERLY WORKING WHEN USING SHARD ID > 0	Critical	SOLVED - 03/27/2023
HAL02 - HEAD STATE NOT ENFORCED TO BE ZERO	Low	SOLVED - 03/27/2023
HAL03 - UNUSED CIRCUIT FUNCTIONALITY	Informational	SOLVED - 03/27/2023
HAL04 - QUEUE NOT ENFORCED TO BE EMPTY RIGHT AFTER POPPING ALL ELEMENTS	Informational	ACKNOWLEDGED
HAL05 - UNNEEDED INITIALIZATION OF UINT256 VARIABLES	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) CIRCUIT NOT PROPERLY WORKING WHEN USING SHARD ID > 0 - CRITICAL

Description:

The STORAGE QUERIES FILTER circuit (storage_validity_by_grand_product) does not produce the intended output when shard ID is greater than 0. When the keys of each element of the initial queue are being packed throughout the sorting of the queues, the second linear combination is wrongly created, overlapping the bits of the address variable, due to using the incorrect coefficient.

The problem with this bug, is that as the pack_key function is returning an incorrect packed key to the sorting functionality, the final sorted queue of the circuit is not being properly generated, leading to even more issues afterward. This can be seen as a completeness bug.

Code Location:

Listing 1: storage_validity_by_grand_product/mod.rs (Line 425)

```
419 const PACKED_WIDTHS: [usize; 2] = [192, 232];
420 // now resolve a logic
421 for (item, is_trivial) in it {
422     // check if keys are equal and check a value
423     let TimestampedStorageLogRecord { record, timestamp } = item;
424
425     let packed_key = pack_key(
426         cs,
427         (record.shard_id.clone(), record.address.clone(), record.
    ↳ key),
428     )?;
429
430     // ensure sorting
431     let (keys_are_equal, previous_key_is_greater) =
432         prepacked_long_comparison(cs, &previous_packed_key, &
    ↳ packed_key, &PACKED_WIDTHS)?;
```

```

    can_not_be_false_if_flagged(cs, &previous_key_is_greater.not(), &is_trivial.not())?;

    // if keys are the same then timestamps are sorted
    let (_, previous_timestamp_is_less) = previous_timestamp.sub(cs, &timestamp)?;
    // enforce if keys are the same and not trivial
    let must_enforce = smart_and(cs, &[keys_are_equal, is_trivial.not()])?;
    can_not_be_false_if_flagged(cs, &previous_timestamp_is_less, &must_enforce)?;

    // we follow the procedure:
    // if keys are different then we finish with a previous one and update parameters
    // else we just update parameters
```

Listing 2: storage_validity_by_grand_product/mod.rs (Line 669)

```
650 pub fn pack_key<E: Engine, CS: ConstraintSystem<E>>(
651     cs: &mut CS,
652     key_tuple: (Byte<E>, UInt160<E>, UInt256<E>),
653 ) -> Result<[Num<E>; 2], SynthesisError> {
654     let shifts = compute_shifts::<E::Fr>();
655
656     // LE packing
657
658     let (shard_id, address, key) = key_tuple;
659     let mut lc_0 = LinearCombination::zero();
660     lc_0.add_assign_number_with_coeff(&key.inner[0].inner, shifts[0]);
661     lc_0.add_assign_number_with_coeff(&key.inner[1].inner, shifts[64]);
662     lc_0.add_assign_number_with_coeff(&key.inner[2].inner, shifts[128]);
663     // 192 in total
664     let value_0 = lc_0.into_num(cs)?;
665
666     let mut lc_1 = LinearCombination::zero();
667     lc_1.add_assign_number_with_coeff(&key.inner[3].inner, shifts[0]);
668     lc_1.add_assign_number_with_coeff(&address.inner, shifts[64]);
669     lc_1.add_assign_number_with_coeff(&shard_id.inner, shifts
```

```
↳ [160]);
    let value_1 = lc_1.into_num(cs)?;
    // 64 + 160 + 8 = 232 in total

    Ok([value_0, value_1])
}
```

Proof of Concept:

1. The STORAGE QUERIES FILTER circuit receives as input witness the storage access requests queue.
2. This circuit aims to order the resulting queue of all the elements in the initial queue by a generated key.
3. The circuit calls the pack_key function to generate the key for the current element of the queue.
4. A shard_id different from 0 is being used.
5. Within the generated key, the bits of the address parameter gets overlapped with the bits of the shard_id.
6. An incorrect key is returned to the main function of the circuit, thus breaking the overall functionality of the circuit.
7. The resulting queue is not ordered as expected, leading to even more issues afterward.

Risk Level:

Likelihood - 5

Impact - 5

Recommendation:

The last coefficient on line 669 needs to be shifts[224] instead of shifts[160].

Listing 3: storage_validity_by_grand_product/mod.rs (Line 669)

```
666 let mut lc_1 = LinearCombination::zero();
667 lc_1.add_assign_number_with_coeff(&key.inner[3].inner, shifts[0]);
668 lc_1.add_assign_number_with_coeff(&address.inner, shifts[64]);
669 lc_1.add_assign_number_with_coeff(&shard_id.inner, shifts[224]);
670 let value_1 = lc_1.into_num(cs)?;
```

Moreover, would be useful to check after each linear combination if the shift value is within the capacity by using:

Listing 4: storage_validity_by_grand_product/mod.rs

```
assert!(shift <= E::Fr::CAPACITY as usize);
```

Remediation Plan:

SOLVED: The MatterLabs team solved the issue by fixing the offset and adding assertion.

Commit ID : 5109e0768c7de799f87ec67bf40b6a544cca4e4e

3.2 (HAL-02) HEAD STATE NOT ENFORCED TO BE ZERO - LOW

Description:

The MESSAGES EVENTS FILTER circuit (log_sorter) is not enforcing the head state of the initial queue received as input to be zero. Even though the LOG DEMULTIPLEXER circuit gives as output all queues that were initially empty, already enforcing the head state to be zero, it is essential to ensure it is atomically checked in each circuit, regardless of where the input is coming from.

Code Location:

Listing 5: log_sorter/mod.rs

```
let structured_input_witness = project_ref!(witness, closed_form_input).cloned();
let initial_queue_witness = project_ref!(witness, initial_queue_witness).cloned();
let intermediate_sorted_queue_state =
    project_ref!(witness, intermediate_sorted_queue_state).cloned();
let sorted_queue_witness = project_ref!(witness, sorted_queue_witness).cloned();

let mut structured_input = EventsDeduplicatorInputOutput::alloc_ignoring_outputs(
    cs,
    structured_input_witness.clone(),
)?;

Boolean::enforce_equal(cs, &structured_input.start_flag, &Boolean::constant(true))?;

let mut initial_queue = StorageLogQueue::from_raw_parts(
    cs,
    structured_input
        .observable_input
        .initial_log_queue_state
```

```
            .head_state,
        structured_input
            .observable_input
            .initial_log_queue_state
            .tail_state,
        structured_input
            .observable_input
            .initial_log_queue_state
            .num_items,
)?;

// dbg!(initial_queue.clone().into_state().create_witness());

if let Some(wit) = initial_queue_witness {
    initial_queue.witness = wit;
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Adding the enforcement for the initial queue head state to be zero right after getting it from the input witness.

Listing 6: log_sorter/mod.rs (Line 79)

```
77
78 // it must be trivial
79 initial_queue.head_state.enforce_equal(cs, &Num::zero())?;
80
81 // dbg!(initial_queue.clone().into_state().create_witness());
82
83 if let Some(wit) = initial_queue_witness {
84     initial_queue.witness = wit;
85 }
```

Remediation Plan:

SOLVED: The MatterLabs team solved the issue by enforcing the initial queue head state to zero.

Commit ID : b0a79356613655bddccaab3b89dbf1142b5483fb

3.3 (HAL-03) UNUSED CIRCUIT FUNCTIONALITY - INFORMATIONAL

Description:

Within the LOG DEMULTIPLEXER circuit (demux_log_queue) the demultiplex_storage_logs_inner function is declared but never used. The demultiplex_storage_logs_inner_optimized function is used instead.

Code Location:

Listing 7: demux_log_queue/mod.rs

```
pub fn demultiplex_storage_logs_inner<
    E: Engine,
    CS: ConstraintSystem<E>,
    R: CircuitArithmeticRoundFunction<E, 2, 3, StateElement = Num<E>>,
>(
    cs: &mut CS,
    mut storage_log_queue: StorageLogQueue<E>,
    round_function: &R,
    limit: usize,
) -> Result<[StorageLogQueue<E>; NUM_SEPARATE_QUEUES], SynthesisError> {
    assert!(limit <= u32::MAX as usize);

    let mut optimizer = SpongeOptimizer::new(round_function.clone(), 3);

    let mut rollup_storage_queue = StorageLogQueue::empty();
    // let mut porter_storage_queue = StorageLogQueue::empty();
    let mut events_queue = StorageLogQueue::empty();
    let mut l1_messages_queue = StorageLogQueue::empty();
    let mut keccak_calls_queue = StorageLogQueue::empty();
    let mut sha256_calls_queue = StorageLogQueue::empty();
    let mut ecdsa_calls_queue = StorageLogQueue::empty();

    const SYSTEM_CONTRACTS_OFFSET_ADDRESS: u16 = 1 << 15;
```

```

    const KECCAK256_ROUND_FUNCTION_PRECOMPILE_ADDRESS: u16 = SYSTEM_CONTRACTS_OFFSET_ADDRESS + 0x10;
    const SHA256_ROUND_FUNCTION_PRECOMPILE_ADDRESS: u16 = 0x02; // as in Ethereum
    const ECRECOVER_INNER_FUNCTION_PRECOMPILE_ADDRESS: u16 = 0x01; // as in Ethereum

    let keccak_precompile_address = UInt160::from_uint(u160::from_u64(
        KECCAK256_ROUND_FUNCTION_PRECOMPILE_ADDRESS as u64,
    ));
    let sha256_precompile_address = UInt160::from_uint(u160::from_u64(
        SHA256_ROUND_FUNCTION_PRECOMPILE_ADDRESS as u64,
    ));
    let ecrecover_precompile_address = UInt160::from_uint(u160::from_u64(
        ECRECOVER_INNER_FUNCTION_PRECOMPILE_ADDRESS as u64,
    ));

    for _ in 0..limit {
        let execute = storage_log_queue.is_empty(cs)?.not();

        // let n = cs.get_current_step_number();
        let popped = storage_log_queue.pop_first(cs, &execute, round_function)?;
        // dbg!(cs.get_current_step_number() - n); // 291

        let is_storage_aux_byte =
            Num::equals(cs, &aux_byte_for_storage().inner, &popped.aux_byte.inner)?;
        let is_event_aux_byte =
            Num::equals(cs, &aux_byte_for_event().inner, &popped.aux_byte.inner)?;
        let is_l1_message_aux_byte =
            Num::equals(cs, &aux_byte_for_l1_message().inner, &popped.aux_byte.inner)?;
        let is_precompile_aux_byte = Num::equals(
            cs,
            &aux_byte_for_precompile_call().inner,
            &popped.aux_byte.inner,
        )?;

```

```
        let is_keccak_address = UInt160::equals(cs, &keccak_precompile_address, &popped.address)?;
        let is_sha256_address = UInt160::equals(cs, &sha256_precompile_address, &popped.address)?;
        let is_ecrecover_address =
            UInt160::equals(cs, &ecrecover_precompile_address, &popped.address)?;

        let is_rollup_shard = popped.shard_id.inner.is_zero(cs)?;

        let execute_rollup_storage =
            smart_and(cs, &[is_storage_aux_byte, is_rollup_shard, execute])?;
        let execute_porter_storage =
            smart_and(cs, &[is_storage_aux_byte, is_rollup_shard.not(), execute])?;
        Boolean::enforce_equal(cs, &execute_porter_storage, &Boolean::constant(false))?;
        let execute_event = smart_and(cs, &[is_event_aux_byte, execute])?;
        let execute_l1_message = smart_and(cs, &[is_l1_message_aux_byte, execute])?;
        let execute_keccak_call =
            smart_and(cs, &[is_precompile_aux_byte, is_keccak_address, execute])?;
        let execute_sha256_call =
            smart_and(cs, &[is_precompile_aux_byte, is_sha256_address, execute])?;
        let execute_ecrecover_call =
            smart_and(cs, &[is_precompile_aux_byte, is_ecrecover_address, execute])?;

        // let n = cs.get_current_step_number();
        rollup_storage_queue.push_with_optimizer(
            cs,
            LogType::RollupStorage as u64,
            &popped,
            &execute_rollup_storage,
            &mut optimizer,
        )?;
        // porter_storage_queue.push_with_optimizer(cs, LogType::PorterStorage as u64, &popped, &execute_porter_storage, &mut optimizer)?;
        events_queue.push_with_optimizer(
```

```
            cs,
            LogType::Events as u64,
            &popped,
            &execute_event,
            &mut optimizer,
        )?;
        l1_messages_queue.push_with_optimizer(
            cs,
            LogType::L1Messages as u64,
            &popped,
            &execute_l1_message,
            &mut optimizer,
        )?;
        keccak_calls_queue.push_with_optimizer(
            cs,
            LogType::KeccakCalls as u64,
            &popped,
            &execute_keccak_call,
            &mut optimizer,
        )?;
        sha256_calls_queue.push_with_optimizer(
            cs,
            LogType::Sha256Calls as u64,
            &popped,
            &execute_sha256_call,
            &mut optimizer,
        )?;
        ecdsa_calls_queue.push_with_optimizer(
            cs,
            LogType::ECRecoverCalls as u64,
            &popped,
            &execute_ecrecover_call,
            &mut optimizer,
        )?;
        // dbg!(cs.get_current_step_number() - n); // 96

        // let n = cs.get_current_step_number();
        optimizer.enforce(cs)?;
        // dbg!(cs.get_current_step_number() - n); // 338

        let expected_bitmask_bits = [
            is_storage_aux_byte,
            is_event_aux_byte,
            is_l1_message_aux_byte,
```

```
                is_precompile_aux_byte,
            ];

            let (is_bitmask, all_flags_are_false) =
                check_if_bitmask_and_if_empty(cs, &expected_bitmask_bits)?;
            can_not_be_false_if_flagged(cs, &is_bitmask, &Boolean::Constant(true))?;
            can_not_be_false_if_flagged(cs, &all_flags_are_false.not(), &execute)?;
        }

        storage_log_queue.enforce_to_be_empty(cs)?;

        let all_queues = [
            rollup_storage_queue,
            events_queue,
            l1_messages_queue,
            keccak_calls_queue,
            sha256_calls_queue,
            ecdsa_calls_queue,
        ];

        Ok(all_queues)
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Any unused code is recommended to be removed for better readability of the overall code and optimization.

Remediation Plan:

SOLVED: The MatterLabs team solved the issue by removing the unused function.

Commit ID : 06c2e76546369fb112d8ac14fb5388154857435b

3.4 (HAL-04) QUEUE NOT ENFORCED TO BE EMPTY RIGHT AFTER POPPING ALL ELEMENTS - INFORMATIONAL

Description:

The MERKLEIZER circuit (merkleize_l1_messages) does not enforce the initial queue to be empty right after popping all elements. Even though it does it at the end of the circuit, as this circuit is resource-consuming while computing the linear hash and the Merkle tree hash, it would be useful to enforce it before all the hashing functionality for better readability of the overall code and optimization.

Code Location:

Listing 8: merkleize_l1_messages/merkleize.rs (Line 244)

```
for chunk in linear_hash_input[4..].chunks_exact_mut(MESSAGE_SERIALIZATION_BYTES) {
    let can_pop = initial_queue.is_empty(cs)?.not();
    let item = initial_queue.pop_first(cs, &can_pop, round_function)?;
    let serialized = item.serialize(cs)?;
    assert_eq!(chunk.len(), serialized.len());
    for (dst, src) in chunk.iter_mut().zip(serialized.iter()) {
        *dst = Byte::conditionally_select(cs, &can_pop, src, dst)?;
    }
}

let linear_hash = if output_linear_hash {
    println!(
        "Computing linear hash over {} bytes",
        linear_hash_input.len()
    );
    let pubdata_hash = tree_hasher.hash(cs, &linear_hash_input)?;
    let pubdata_hash_as_bytes32 = Bytes32::from_bytes_array(&pubdata_hash);

```

```
    pubdata_hash_as_bytes32
} else {
    Bytes32::empty()
};

// a little bit tricky: unsafe cast, but we checked the length, and ABI wise it's guaranteed
// later on we can use split_array_ref

let leafs_only_bytes = &linear_hash_input[4..];
assert!(leafs_only_bytes.len() % MESSAGE_SERIALIZATION_BYTES == 0);

let mut leafs = vec![];
for chunk in leafs_only_bytes.chunks_exact(MESSAGE_SERIALIZATION_BYTES) {
    let leaf_encoding: [_; MESSAGE_SERIALIZATION_BYTES] = chunk.to_vec().try_into().unwrap();
    leafs.push(leaf_encoding);
}

println!("Computing tree over {} leafs", leafs.len());

let calculated_merkle_root =
    circuit_compute_merkle_root_from_leafs_generic::<_, _, H, ARITY>(cs, &leafs, tree_hasher)?;

initial_queue.enforce_to_be_empty(cs)?;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Enforce the initial queue to be empty right after popping all elements for better readability of the overall code and optimization.

Remediation Plan:

ACKNOWLEDGED: The MatterLabs team acknowledged this issue. It will be addressed while moving to the new proof system.

3.5 (HAL-05) UNNEEDED INITIALIZATION OF UINT256 VARIABLES - INFORMATIONAL

Description:

As `i` is an `uint256`, it is already initialized to 0. `uint256 i = 0` reassigns the 0 to `i` which wastes gas.

Code Location:

Verifier.sol
- Line 134: `for (uint256 i = 0; i < public_inputs.length; i = i.uncheckedInc()){`
- Line 139: `for (uint256 i = 0; i < STATE_WIDTH; i = i.uncheckedInc()){`
- Line 164: `for (uint256 i = 0; i < proof.quotient_poly_parts_commitments.length; i = i.uncheckedInc()){`
- Line 172: `for (uint256 i = 0; i < proof.state_polys_openings_at_z.length; i = i.uncheckedInc()){`
- Line 178: `for (uint256 i = 0; i < proof.state_polys_openings_at_z_omega.length; i = i.uncheckedInc()){`
- Line 183: `for (uint256 i = 0; i < proof.gate_selectors_openings_at_z.length; i = i.uncheckedInc()){`
- Line 188: `for (uint256 i = 0; i < proof.copy_permutation_polys_openings_at_z.length; i = i.uncheckedInc()){`

Plonk4VerifierWithAccessToDNext.sol
- Line 144: `for (uint256 i = 0; i < vk.num_inputs; i = i.uncheckedInc()){`
- Line 148: `for (uint256 i = 0; i < STATE_WIDTH; i = i.uncheckedInc()){`
- Line 164: `for (uint256 i = 0; i < proof.quotient_poly_parts_commitments.length; i = i.uncheckedInc()){`
- Line 171: `for (uint256 i = 0; i < proof.state_polys_openings_at_z.length; i = i.uncheckedInc()){`
- Line 175: `for (uint256 i = 0; i < proof.state_polys_openings_at_z_omega`

.length; i = i.uncheckedInc()){
- Line 178: for (uint256 i = 0; i < proof.gate_selectors_openings_at_z.length; i = i.uncheckedInc()){
- Line 181: for (uint256 i = 0; i < proof.copy_permutation_polys_openings_at_z.length; i = i.uncheckedInc()){
- Line 301: for (uint256 i = 0; i < lagrange_poly_numbers.length; i = i.uncheckedInc()){
- Line 307: for (uint256 i = 0; i < vk.num_inputs; i = i.uncheckedInc()){
- Line 324: for (uint256 i = 0; i < proof.copy_permutation_polys_openings_at_z.length; i = i.uncheckedInc()){
- Line 448: for (uint256 i = 0; i < proof.state_polys_openings_at_z.length;){
- Line 472: for (uint256 i = 0; i < STATE_WIDTH - 1; i = i.uncheckedInc()){
- Line 558: for (uint256 i = 0; i < STATE_WIDTH - 1; i = i.uncheckedInc()){
- Line 613: for (uint256 i = 0; i < STATE_WIDTH; i = i.uncheckedInc()){
- Line 622: for (uint256 i = 0; i < STATE_WIDTH - 1; i = i.uncheckedInc()){

PairingsBn254.sol
- Line 240: for (uint256 i = 0; i < elements;){

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended not to initialize uint256 variables to 0 to reduce the gas costs. For example, use instead:

```
for (uint256 i; i < length; ++i){
```

Remediation Plan:

ACKNOWLEDGED: The MatterLabs team acknowledged this issue.

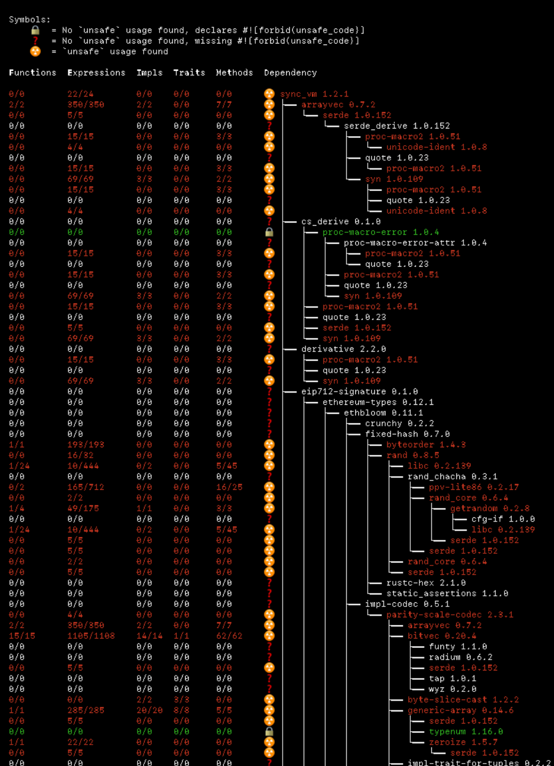
AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

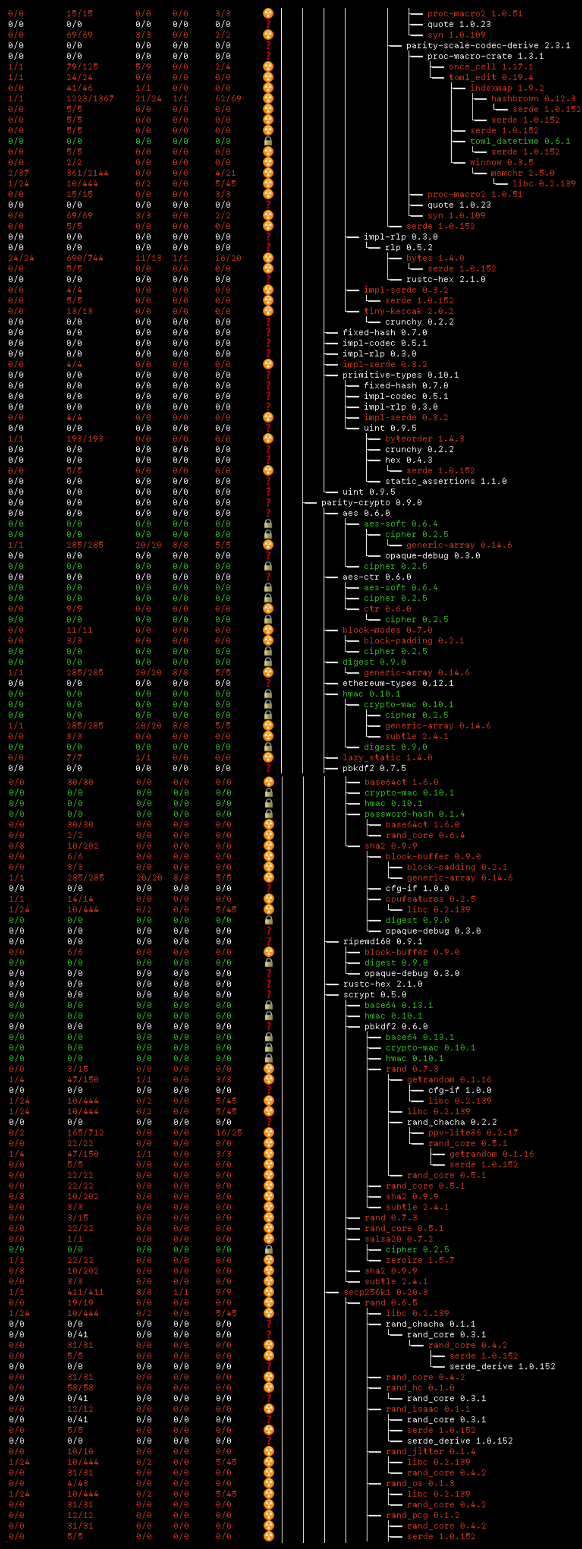
Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped circuits. Among the tools used was cargo geiger, a tool that lists statistics related to the usage of unsafe Rust code in a Rust crate and all its dependencies. After Halborn verified all the circuits in the repository and was able to compile them correctly, cargo geiger was run on the all-scoped circuits.

cargo geiger results:

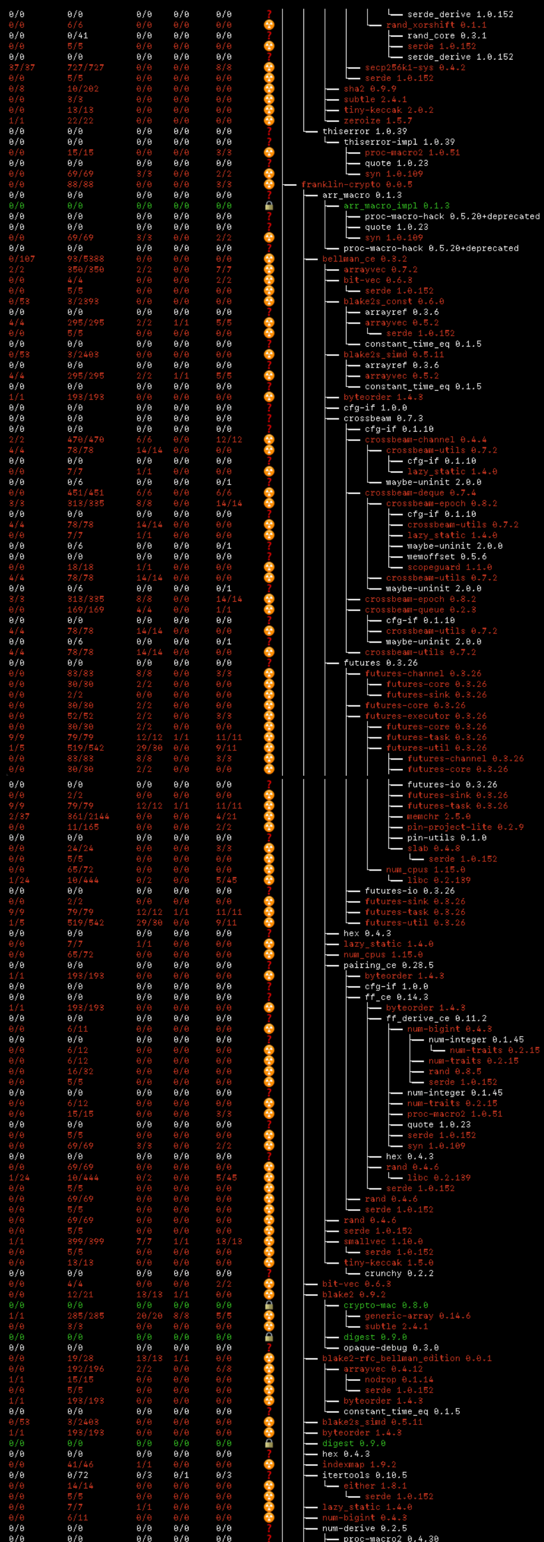


```
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ❓  = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢️  = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        22/24        0/0    0/0     0/0      ☢️ sync_vm 1.2.1
2/2        350/350      2/2    0/0     7/7      ☢️ ├── arrayvec 0.7.2
0/0        5/5          0/0    0/0     0/0      ☢️ │   └── serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ❓ │       └── serde_derive 1.0.152
0/0        15/15        0/0    0/0     3/3      ☢️ │           ├── proc-macro2 1.0.51
0/0        4/4          0/0    0/0     0/0      ☢️ │           │   └── unicode-ident 1.0.8
0/0        0/0          0/0    0/0     0/0      ❓ │           ├── quote 1.0.23
0/0        15/15        0/0    0/0     3/3      ☢️ │           │   └── proc-macro2 1.0.51
0/0        69/69        3/3    0/0     2/2      ☢️ │           └── syn 1.0.109
0/0        15/15        0/0    0/0     3/3      ☢️ │               ├── proc-macro2 1.0.51
0/0        0/0          0/0    0/0     0/0      ❓ │               ├── quote 1.0.23
0/0        4/4          0/0    0/0     0/0      ☢️ │               └── unicode-ident 1.0.8
0/0        0/0          0/0    0/0     0/0      ❓ ├── cs_derive 0.1.0
0/0        0/0          0/0    0/0     0/0      🔒 │   ├── proc-macro-error 1.0.4
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── proc-macro-error-attr 1.0.4
0/0        15/15        0/0    0/0     3/3      ☢️ │   │   │   ├── proc-macro2 1.0.51
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   └── quote 1.0.23
0/0        15/15        0/0    0/0     3/3      ☢️ │   │   ├── proc-macro2 1.0.51
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── quote 1.0.23
0/0        69/69        3/3    0/0     2/2      ☢️ │   │   └── syn 1.0.109
0/0        15/15        0/0    0/0     3/3      ☢️ │   ├── proc-macro2 1.0.51
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── quote 1.0.23
0/0        5/5          0/0    0/0     0/0      ☢️ │   ├── serde 1.0.152
0/0        69/69        3/3    0/0     2/2      ☢️ │   └── syn 1.0.109
0/0        0/0          0/0    0/0     0/0      ❓ ├── derivative 2.2.0
0/0        15/15        0/0    0/0     3/3      ☢️ │   ├── proc-macro2 1.0.51
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── quote 1.0.23
0/0        69/69        3/3    0/0     2/2      ☢️ │   └── syn 1.0.109
0/0        0/0          0/0    0/0     0/0      ❓ ├── eip712-signature 0.1.0
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── ethereum-types 0.12.1
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── ethbloom 0.11.1
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   ├── crunchy 0.2.2
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   ├── fixed-hash 0.7.0
1/1        193/193      0/0    0/0     0/0      ☢️ │   │   │   │   ├── byteorder 1.4.3
0/0        16/32        0/0    0/0     0/0      ☢️ │   │   │   │   ├── rand 0.8.5
1/24       10/444       0/2    0/0     5/45     ☢️ │   │   │   │   │   ├── libc 0.2.139
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   ├── rand_chacha 0.3.1
0/2        165/712      0/0    0/0     16/25    ☢️ │   │   │   │   │   │   ├── ppv-lite86 0.2.17
0/0        2/2          0/0    0/0     0/0      ☢️ │   │   │   │   │   │   ├── rand_core 0.6.4
1/4        49/175       1/1    0/0     3/3      ☢️ │   │   │   │   │   │   │   ├── getrandom 0.2.8
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   │   │   │   │   ├── cfg-if 1.0.0
1/24       10/444       0/2    0/0     5/45     ☢️ │   │   │   │   │   │   │   │   └── libc 0.2.139
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │   │   │   │   │   └── serde 1.0.152
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │   │   │   │   └── serde 1.0.152
0/0        2/2          0/0    0/0     0/0      ☢️ │   │   │   │   │   ├── rand_core 0.6.4
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │   │   │   └── serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   ├── rustc-hex 2.1.0
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │   └── static_assertions 1.1.0
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   ├── impl-codec 0.5.1
0/0        4/4          0/0    0/0     0/0      ☢️ │   │   │   │   └── parity-scale-codec 2.3.1
2/2        350/350      2/2    0/0     7/7      ☢️ │   │   │   │       ├── arrayvec 0.7.2
15/15      1105/1108    14/14  1/1     62/62    ☢️ │   │   │   │       ├── bitvec 0.20.4
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │       │   ├── funty 1.1.0
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │       │   ├── radium 0.6.2
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │   │       │   ├── serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │       │   ├── tap 1.0.1
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │       │   └── wyz 0.2.0
0/0        0/0          2/2    3/3     0/0      ☢️ │   │   │   │       ├── byte-slice-cast 1.2.2
1/1        285/285      20/20  8/8     5/5      ☢️ │   │   │   │       ├── generic-array 0.14.6
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │   │       │   ├── serde 1.0.152
0/0        0/0          0/0    0/0     0/0      🔒 │   │   │   │       │   ├── typenum 1.16.0
1/1        22/22        0/0    0/0     0/0      ☢️ │   │   │   │       │   └── zeroize 1.5.7
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │   │       │       └── serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │       ├── impl-trait-for-tuples 0.2.2
```

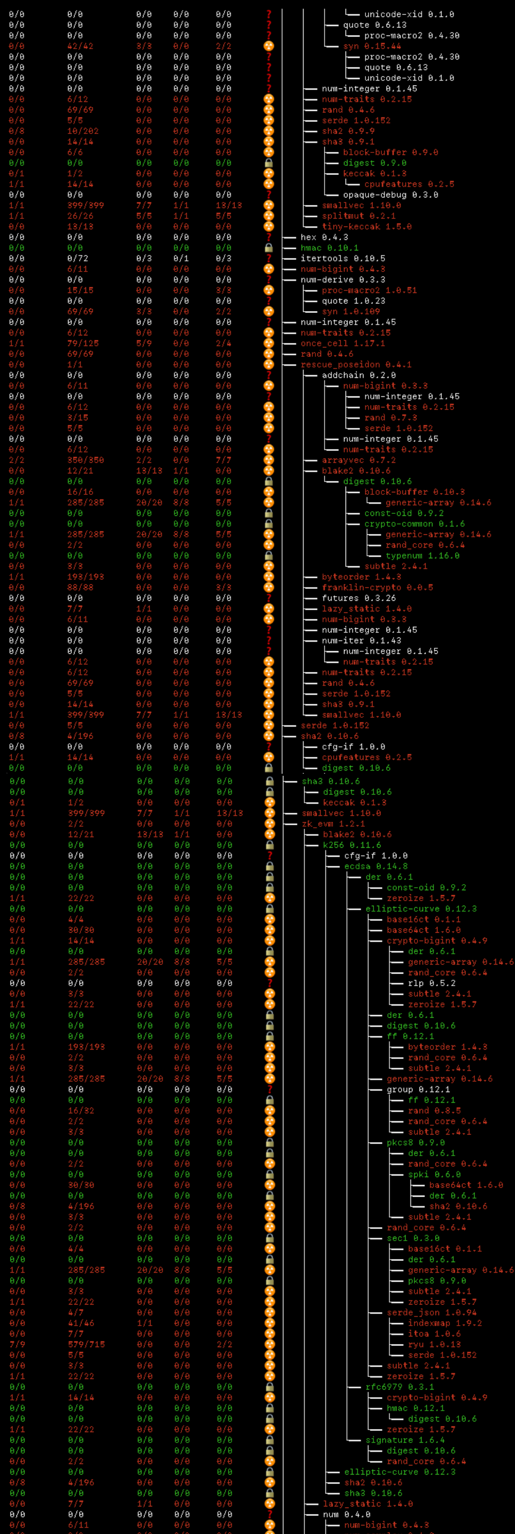
AUTOMATED TESTING

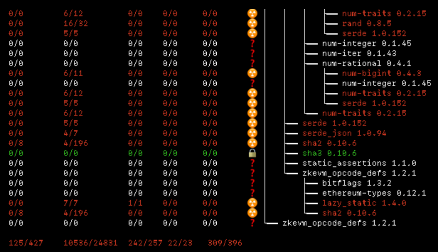


AUTOMATED TESTING



AUTOMATED TESTING





```
0/0        6/12         0/0     0/0     0/0     ☢  │    │    │    ├── num-traits 0.2.15
0/0        16/32        0/0     0/0     0/0     ☢  │    │    │    ├── rand 0.8.5
0/0        5/5          0/0     0/0     0/0     ☢  │    │    │    └── serde 1.0.152
0/0        0/0          0/0     0/0     0/0     ?  │    │    ├── num-integer 0.1.45
0/0        0/0          0/0     0/0     0/0     ?  │    │    ├── num-iter 0.1.43
0/0        0/0          0/0     0/0     0/0     ?  │    │    ├── num-rational 0.4.1
0/0        6/11         0/0     0/0     0/0     ☢  │    │    │    ├── num-bigint 0.4.3
0/0        0/0          0/0     0/0     0/0     ?  │    │    │    ├── num-integer 0.1.45
0/0        6/12         0/0     0/0     0/0     ☢  │    │    │    ├── num-traits 0.2.15
0/0        5/5          0/0     0/0     0/0     ☢  │    │    │    └── serde 1.0.152
0/0        6/12         0/0     0/0     0/0     ☢  │    │    └── num-traits 0.2.15
0/0        5/5          0/0     0/0     0/0     ☢  │    ├── serde 1.0.152
0/0        4/7          0/0     0/0     0/0     ☢  │    ├── serde_json 1.0.94
0/8        4/196        0/0     0/0     0/0     ☢  │    ├── sha2 0.10.6
0/0        0/0          0/0     0/0     0/0     🔒 │    ├── sha3 0.10.6
0/0        0/0          0/0     0/0     0/0     ?  │    ├── static_assertions 1.1.0
0/0        0/0          0/0     0/0     0/0     ?  │    └── zkevm_opcode_defs 1.2.1
0/0        0/0          0/0     0/0     0/0     ?  │         ├── bitflags 1.3.2
0/0        0/0          0/0     0/0     0/0     ?  │         ├── ethereum-types 0.12.1
0/0        7/7          1/1     0/0     0/0     ☢  │         ├── lazy_static 1.4.0
0/8        4/196        0/0     0/0     0/0     ☢  │         └── sha2 0.10.6
0/0        0/0          0/0     0/0     0/0     ?  └── zkevm_opcode_defs 1.2.1

125/427    10536/24831  242/257 22/23   309/396
```

- As a result of the tests carried out with the cargo geiger tool, the results were obtained and reviewed by Halborn. Based on the results reviewed, all warnings were determined to not pose a security issue.

AUTOMATED TESTING

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was cargo audit, a command-line utility which inspects Cargo.lock files and compares them against the RustSec Advisory Database, a community database of security vulnerabilities maintained by the Rust Secure Code Working Group. Cargo audit performed a scan on all the circuits.

cargo audit results:

```
      Loaded 516 security advisories (from /Users/omar/.cargo/advisory-db)
    Scanning Cargo.lock for vulnerabilities (206 crate dependencies)
Crate:     aes-ctr
Version:   0.6.0
Warning:   unmaintained
Title:     `aes-ctr` has been merged into the `aes` crate
Date:      2021-04-29
ID:        RUSTSEC-2021-0061
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0061
Dependency tree:
aes-ctr 0.6.0
└── parity-crypto 0.9.0
    └── eip712-signature 0.1.0
        └── sync_vm 1.2.1

Crate:     aes-soft
Version:   0.6.4
Warning:   unmaintained
Title:     `aes-soft` has been merged into the `aes` crate
Date:      2021-04-29
ID:        RUSTSEC-2021-0060
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0060
Dependency tree:
aes-soft 0.6.4
├── aes-ctr 0.6.0
│   └── parity-crypto 0.9.0
│       └── eip712-signature 0.1.0
│           └── sync_vm 1.2.1
└── aes 0.6.0
    └── parity-crypto 0.9.0

Crate:     aesni
Version:   0.10.0
Warning:   unmaintained
Title:     `aesni` has been merged into the `aes` crate
Date:      2021-04-29
ID:        RUSTSEC-2021-0059
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0059
Dependency tree:
aesni 0.10.0
├── aes-ctr 0.6.0
│   └── parity-crypto 0.9.0
│       └── eip712-signature 0.1.0
│           └── sync_vm 1.2.1
└── aes 0.6.0
    └── parity-crypto 0.9.0

warning: 3 allowed warnings found
```

- No major issues found by cargo audit.

AUTOMATED TESTING

THANK YOU FOR CHOOSING

HALBORN