hexens × Ava Labs.

APR.25

# SECURITY REVIEW REPORT FOR AVA LABS

# CONTENTS

# ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

# SCOPE

The analyzed resources are located on:

https://github.com/ava-labs/EncryptedERC/tree/circom2-circuits

The issues described in this report were fixed. Corresponding commits are mentioned in the description.

# AUDITING DETAILS

**STARTED**
03.04.2025

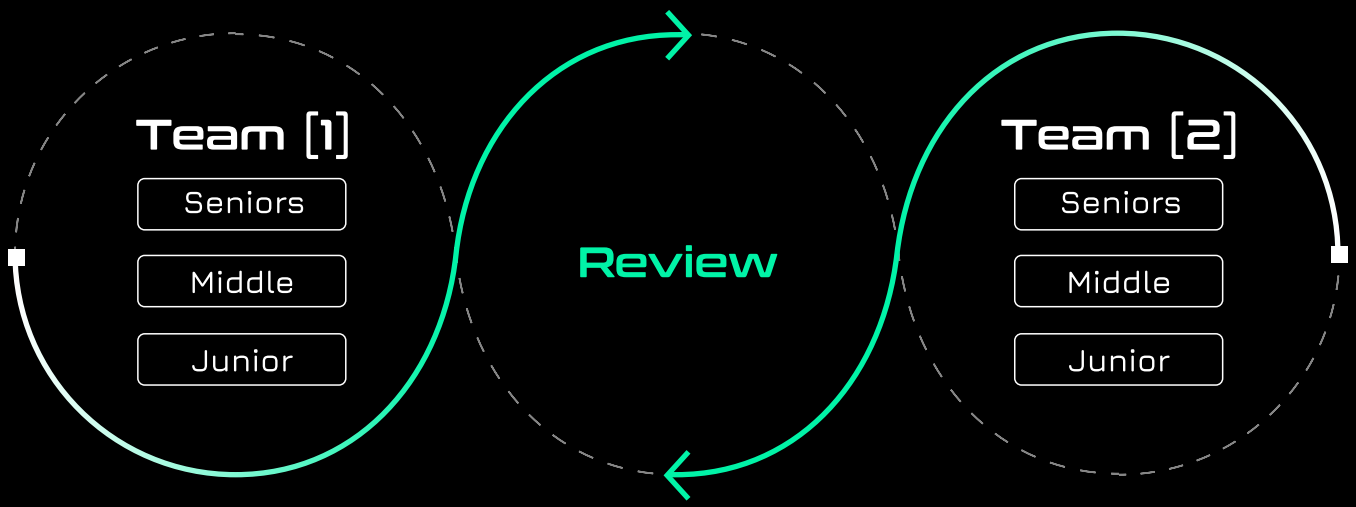
**DELIVERED**
09.04.2025

Review Led by

**HAYK ANDRIASYAN**
Senior Security Researcher | Hexens

## HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



# SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

| Impact | Probability | | | |
|---|---|---|---|---|
| | rare | unlikely | likely | very likely |
| Low/Info | Low/Info | Low/Info | Medium | Medium |
| Medium | Low/Info | Medium | Medium | High |
| High | Medium | Medium | High | Critical |
| Critical | Medium | High | Critical | Critical |

# SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

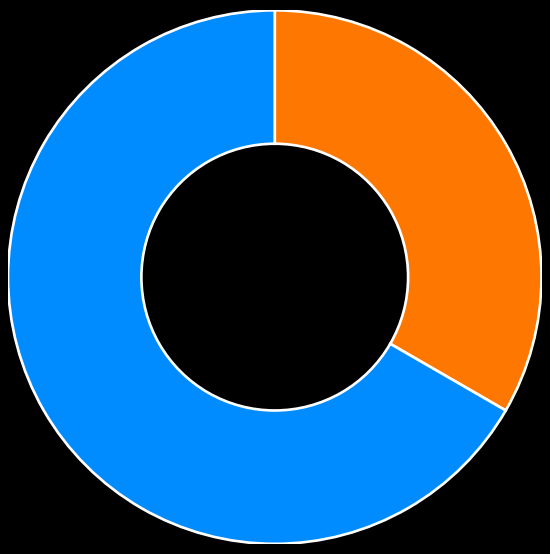
## ISSUE SYMBOLIC CODES

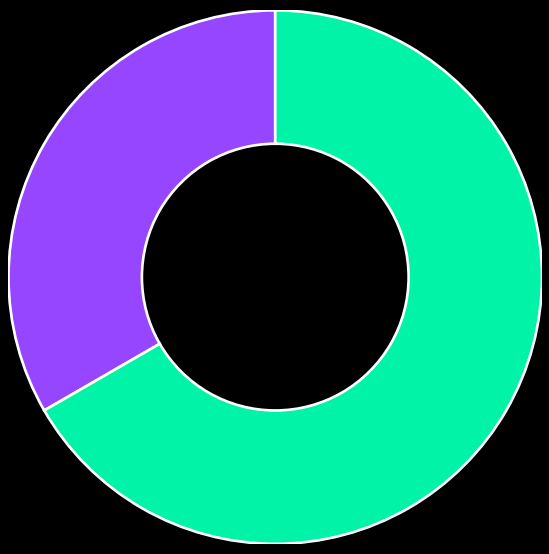
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

# FINDINGS SUMMARY

| Severity | Number of Findings |
| --- | --- |
| Critical | 0 |
| High | 0 |
| Medium | 1 |
| Low | 0 |
| Informational | 2 |

Total: 3



# WEAKNESSES

This section contains the list of discovered weaknesses.

ETRP-16

## MISSING CURVE VALIDATION IN BABYJUBJUB CIRCOM IMPLEMENTATION

SEVERITY: Medium

PATH:

circom/components.circom, circom/circomlib/babyjub.circom

REMEDIATION:

Integrate curve validation in all BabyJubjub operations by adding BabyCheck() calls at the beginning of templates that handle curve points.

STATUS: Fixed

DESCRIPTION:

The Circom implementation of BabyJubjub curve operations does not validate that points lie on the curve before performing operations. In Circom (**babyjub.circom**), while a **BabyCheck()** template exists that can verify curve membership, it's not used in operations like **BabyAdd()**, **BabyDbl()**, or **BabyPbk()**, potentially allowing operations with invalid points.

```
template CheckValue() {
    signal input value;
    signal input privKey;
    signal input valueC1[2];
    signal input valueC2[2];

    // Verify the value is less than the base order
    assert(value <
2736030358979909402780800718157159386076813972158567259200215660948447373041
);

    component checkValue = ElGamalDecrypt();
    checkValue.c1[0] <== valueC1[0];
    checkValue.c1[1] <== valueC1[1];
    checkValue.c2[0] <== valueC2[0];
    checkValue.c2[1] <== valueC2[1];
    checkValue.privKey <== privKey;

    component valueToPoint = BabyPbk();
    valueToPoint.in <== value;

    valueToPoint.Ax === checkValue.outx;
    valueToPoint.Ay === checkValue.outy;
}
```

```
template BabyCheck() {
    signal input x;
    signal input y;

    signal x2;
    signal y2;

    var a = 168700;
    var d = 168696;

    x2 <== x*x;
    y2 <== y*y;

    a*x2 + y2 === 1 + d*x2*y2;
}
```

ETRP-17

# OPTIMIZATION OF LESS THAN CHECKS IN CIRCUIT

SEVERITY: Informational

PATH:

circom/components.circom, circom/mint.circom, circom/transfer.circom, circom/withdraw.circom

REMEDIATION:

Apply the optimizations in PoseidonDecrypt, CheckPublicKey, CheckValue, CheckPCT, MintCircuit, TransferCircuit, WithdrawCircuit components.

STATUS: Acknowledged

DESCRIPTION:

Less Than comparison circuits ensure values are below the BabyJubJub subgroup order and validate that spending does not exceed the sender's balance, preserving arithmetic integrity and transactional correctness. There can be applied some optimizations in the comparison:

```
    var baseOrder =
2736030358979909402780800718157159386076813972158567259200215660948447373041
;

    component bitCheck1 = Num2Bits(252);
    bitCheck1.in <== random;

    component bitCheck2 = Num2Bits(252);
    bitCheck2.in <== baseOrder;

    component lt = LessThan(252);
    lt.in[0] <== random;
    lt.in[1] <== baseOrder;
    lt.out === 1;
```

Since **baseOrder** is a constant, the **bitCheck2** operation is redundant because the constraints are already preserved.

```
    component bitCheck3 = Num2Bits(252);
    bitCheck3.in <== SenderBalance + 1;

    component checkValue = LessThan(252);
    checkValue.in[0] <== ValueToTransfer;
    checkValue.in[1] <== SenderBalance + 1;
    checkValue.out === 1;
```

In the value comparison section, the **LessThan** component can be replaced with **LessEqThan**, and **SenderBalance + 1** should be updated to **SenderBalance**.

ETRP-18

# MISSING SUBGROUP ORDER CHECK IN CHECKRECEIVERVALUE COMPONENT

SEVERITY: Informational

PATH:

circom/components.circom

REMEDIATION:

Validate the receiverRandom to be less than the curve's subgroup order.

STATUS: Fixed

DESCRIPTION:

**CheckReceiverValue** component validates receiver's encrypted amount equals to the actual receiving amount.

```
template CheckReceiverValue() {
    signal input receiverValue;
    signal input receiverPublicKey[2];
    signal input receiverRandom;
    signal input receiverValueC1[2];
    signal input receiverValueC2[2];

    component receiverValueToPoint = BabyPbk();
    receiverValueToPoint.in <== receiverValue;

    component receiverValueEncrypt = ElGamalEncrypt();
    receiverValueEncrypt.random <== receiverRandom;
    receiverValueEncrypt.pk[0] <== receiverPublicKey[0];
    receiverValueEncrypt.pk[1] <== receiverPublicKey[1];
    receiverValueEncrypt.msg[0] <== receiverValueToPoint.Ax;
    receiverValueEncrypt.msg[1] <== receiverValueToPoint.Ay;
```

```
    receiverValueEncrypt.encryptedC1X === receiverValueC1[0];
    receiverValueEncrypt.encryptedC1Y === receiverValueC1[1];
    receiverValueEncrypt.encryptedC2X === receiverValueC2[0];
    receiverValueEncrypt.encryptedC2Y === receiverValueC2[1];
}
```

The **receiverRandom** value misses the check to be less than the BabyJubJub subgroup order.


hexens
×
Ava
Labs.