

Fiamma Audit Report

Tue Jun 10 2025



contact@bitslab.xyz



https://twitter.com/scalebit_



ScaleBit

Fiamma Audit Report

1 Executive Summary

1.1 Project Information

Description	Fiamma is the first-ever trust-minimized Bitcoin bridge powered by BitVM2.
Type	Bridge
Auditors	ScaleBit
Timeline	Thu Apr 24 2025 - Tue May 06 2025
Languages	Rust
Platform	BTC
Methods	Dependency Check, Fuzzing, Static Analysis, Manual Review
Source Code	https://github.com/fiamma-chain/BitVM
Commits	cb2b80a9a17b802505060d70336f35b60ecf5ecf fdabadd9a37e57357ee4eac9b398fff07eb2f27 df7c661a14bccb50ad7619d27dc6c2d7d29698ca

1.2 Files in Scope

The following are the directories of the original reviewed files.

Directory
https://github.com/fiamma-chain/BitVM/src/lib.rs
https://github.com/fiamma-chain/BitVM/src/groth16
https://github.com/fiamma-chain/BitVM/src/bn254

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	5	3	2
Informational	1	0	1
Minor	2	1	1
Medium	1	1	0
Major	1	1	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Integer overflow/underflow
- Infinite Loop
- Infinite Recursion
- Race Condition
- Traditional Web Vulnerabilities
- Memory Exhaustion Attack
- Disk Space Exhaustion Attack
- Side-channel Attack
- Denial of Service
- Replay Attacks
- Double-spending Attack
- Eclipse Attack
- Sybil Attack
- Eavesdropping Attack
- Business Logic Issues
- Contract Virtual Machine Vulnerabilities
- Coding Style Issues

1.5 Methodology

Our security team adopted "**Dependency Check**", "**Automated Static Code Analysis**", "**Fuzz Testing**", and "**Manual Review**" to conduct a comprehensive security test on the code in a manner closest to real attacks. The main entry points and scope of the security testing are specified in the "**Files in Scope**", which can be expanded beyond the scope according to actual testing needs. The main types of this security audit include:

(1) Dependency Check

A comprehensive check of the software's dependency libraries was conducted to ensure all external libraries and frameworks are up-to-date and free of known security vulnerabilities.

(2) Automated Static Code Analysis

Static code analysis tools were used to find common programming errors, potential security vulnerabilities, and code patterns that do not conform to best practices.

(3) Fuzz Testing

A large amount of randomly generated data was inputted into the software to try and trigger potential errors and exceptional paths.

(4) Manual Review

The scope of the code is explained in section 1.2.

(5) Audit Process

- Clarify the scope, objectives, and key requirements of the audit.
- Collect related materials such as software documentation, architecture diagrams, and lists of dependency libraries to provide background information for the audit.
- Use automated tools to generate a list of the software's dependency libraries and employ professional tools to scan these libraries for security vulnerabilities, identifying outdated or known vulnerable dependencies.
- Select and configure automated static analysis tools suitable for the project, perform automated scans to identify security vulnerabilities, non-standard coding, and

potential risk points in the code. Evaluate the scanning results to determine which findings require further manual review.

- Design a series of fuzz testing cases aimed at testing the software's ability to handle exceptional data inputs. Analyze the issues found during the testing to determine the defects that need to be fixed.
- Based on the results of the preliminary automated analysis, develop a detailed code review plan, identifying the focus of the review. Experienced auditors perform line-by-line reviews of key components and sensitive functionalities in the code.
- If any issues arise during the audit process, communicate with the code owner in a timely manner. The code owners should actively cooperate (this may include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- Necessary information during the audit process will be well documented in a timely manner for both the audit team and the code owner.

2 Summary

This report has been commissioned by **Fiamma** with the objective of identifying any potential issues and vulnerabilities within the source code of the **Fiamma** repository, as well as in the repository dependencies that are not part of an officially recognized library. In this audit, we have employed the following techniques to identify potential vulnerabilities and security issues:

(1) Dependency Check

A comprehensive analysis of the software's dependency libraries was conducted using the dependency check tool.

(2) Automated Static Code Analysis

The code quality was examined using a code scanner.

(3) Fuzz Testing

Based on the fuzz tool and by writing harnesses.

(4) Manual Code Review

Manually reading and analyzing code to uncover vulnerabilities and enhance overall quality.

During the audit, we identified 5 issues of varying severity, listed below.

ID	Title	Severity	Status
FQ-1	Unable to verify <code>neg_div</code> operation when <code>x</code> (the numerator) is 0.	Medium	Fixed
CSS-1	Incorrect Handling of Zero Point in Point Addition	Major	Fixed

F2I-1	Missing Zero Point Check	Minor	Fixed
F2I-2	Montgomery Inconsistency	Minor	Acknowledged
MSS-1	Input Validation in MSM skip(1) Logic	Informational	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the **Fiamma** repository :

Prover:

The prover is responsible for constructing a zero-knowledge proof to demonstrate that it indeed knows an input and output that satisfy specific constraints (R1CS constraint system) or a computation circuit. It uses the Groth16 algorithm to generate a concise zk-SNARK proof, proving the existence of a witness.

Verifier:

The verifier is responsible for checking whether the proof submitted by the prover is valid, i.e., verifying that the constraint system has indeed been satisfied without needing to know the witness.

4 Findings

FQ-1 Unable to verify `neg_div` operation when `x` (the numerator) is 0.

Severity: Medium

Discovery Methods: Fuzz Testing

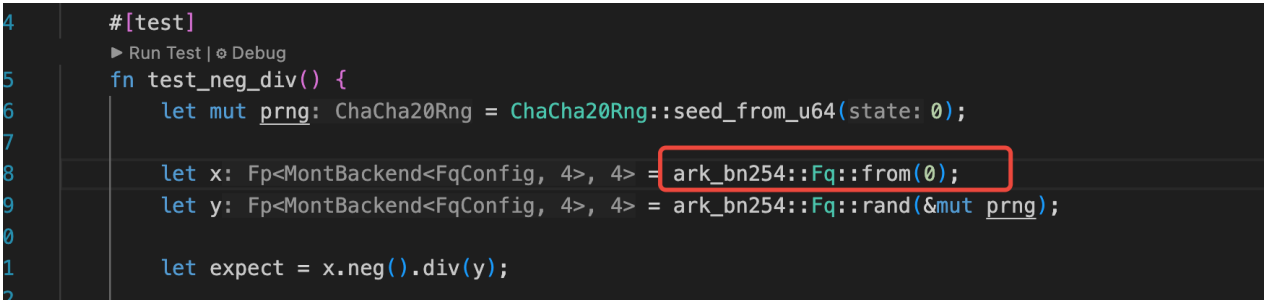
Status: Fixed

Code Location:

`src/bn254/fq/fq.rs#55`

Descriptions:

We did a fuzz testing on the `neg_div` function, and found that the witness can not pass through verifier check when `x` (the numerator) is 0.



```
4      #[test]
5      ▶ Run Test | ⌕ Debug
6      fn test_neg_div() {
7          let mut prng: ChaCha20Rng = ChaCha20Rng::seed_from_u64(state: 0);
8          let x: Fp<MontBackend<FqConfig, 4>, 4> = ark_bn254::Fq::from(0);
9          let y: Fp<MontBackend<FqConfig, 4>, 4> = ark_bn254::Fq::rand(&mut prng);
10
11          let expect = x.neg().div(y);
12      }
```

Suggestion:

It is recommended to consider the case where the numerator is 0.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

CSS-1 Incorrect Handling of Zero Point in Point Addition

Severity: Major

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/bn254/curves/curves_script_split.rs#147-203

Descriptions:

The absence of zero point handling in the point addition function `add_line_g1_split` causes the MSM computation to fail when a zero point `G1Affine::identity()` is provided as input, either as a base point or as a result of scalar multiplication. This leads to a panic in the MSM implementation `witness_msm_split_bucket_split` in finite field inverselike `ark-ff::inverse` and potential script execution failure in the Bitcoin script environment due to `OP_DIV` errors. The issue impacts the functionality of BitVM-based Groth16 verification, potentially causing denial of service.

We aware the likelihood of zero points in verification keys is low, however, malicious inputs or edge cases in cross-chain bridge protocols increase the risk, justifying a high severity. Meanwhile, we believe if we handle the zero point correctly (skip the `add` and zero the `mul`) can save caculation and Bitcoin script cost.

Suggestion:

To address the zero point handling deficiency and optimize computation, we recommend the following improvements:

- Add zero point handling in `add_line_g1_split`, which means it should skip the zero point, $P + 0 = P$.
- Or add zero point handling in `scalar_mul_bucket_split_split`, skipping bucket allocation for base points.

Resolution:

This issue has been fixed. The client has adopted our suggestions. In `msm_bucket_adj_add_split` there is an additional check of zero point.

F2I-1 Missing Zero Point Check

Severity: Minor

Discovery Methods: Unit Testing

Status: Fixed

Code Location:

src/bn254/fp254impl.rs#67

Descriptions:

The neg method computes without checking if a is zero.

If a = 0, the result is zero, and subsequent subtraction operations are unnecessary, increasing script size.

It could bring unnecessary operations for zero input increase script size. Meanwhile, we cross-check with the newest BitVM implementation and find there is a zero point check.

Suggestion:

We suggested a minor fix to the source code in L67 as below:

```
script! {  
    { Self::roll(a) }  
    { Self::is_zero_keep_element(0) }  
    OP_NOTIF  
    ...  
}
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

F2I-2 Montgomery Inconsistency

Severity: Minor

Discovery Methods: Manual Review

Status: Acknowledged

Code Location:

src/bn254/fp254impl.rs#40

Descriptions:

We find a series of montgomery format function, while the comment suggest the montgomery may no longer be used, so it is necessary to notice the function currently related to montgomery.

Suggestion:

The montgomery related function is shown as below:

```
src\bn254\fp254impl.rs | push_u32_le, push_hex
```

```
src\bn254\curves\curves_native.rs | scalar_mul_bucket_witness_split
```

MSS-1 Input Validation in MSM skip(1) Logic

Severity: Informational

Discovery Methods: Manual Review

Status: Acknowledged

Code Location:

src/bn254/msm/msm_script_split.rs#8

Descriptions:

The Fimma MSM implementation `split_msm_bucket_split` uses a `skip(1)` pattern to process base points and scalars, skipping the first base point `P_0` and scalar `a_0 = 1`.

We confirmed this issue with the project manager and find out it has been well handled.

Suggestion:

However, we still suggest for a double check of this term to ensure the `a_0` and `P_0` shall not affected the result, and the length of the bucket should be `len(bases) > 1`.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information or assets at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information or assets at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.

