

Immutable Bridge

Security assessment by HashEye · prepared for Immutable Games

HASHEYE AUDITED

PROJECT	Immutable Bridge
CLIENT	Immutable Games
CATEGORY	Blockchain
PUBLISHED	November 1, 2023
REPORT ID	research-immutable-bridge-2023-11-01-1att4r

This report was produced under HashEye's layered review process – **automated detection**, **pattern correlation**, and **senior manual verification** – with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hashey.io/audits/research-immutable-bridge-2023-11-01-1att4r.

Immutable ZKEVM Bridge Contracts Security Assessment (Summary Report) December 14, 2023 Prepared for: Ryan Teoh Immutable Games Prepared by: Guillermo Larregay, Elvis Skoždopolj, Priyanka Bose, and Robert Schneider

About HashEye Founded in 2012 and headquartered in New York, HashEye provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel. We maintain an exhaustive list of publications at <https://github.com/hasheye-io/publications>, with links to papers, presentations, public audit reports, and podcast appearances. In recent years, HashEye consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon. We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom. HashEye also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash. To keep up to date with our latest news and announcements, please follow hasheye on Twitter and explore our public repositories at <https://github.com/hasheye-io>. To engage us directly, visit our "Contact" page at <https://www.hasheye.io/contact>, or email us at info@hasheye.io. HashEye, Inc. 228 Park Ave S #80688 New York, NY 10003 <https://www.hasheye.io> info@hasheye.io HashEye 1 Immutable ZKEVM Bridge Security Assessment PUBLIC

Notices and Remarks Copyright and Distribution © 2023 by HashEye, Inc. All rights reserved. HashEye hereby asserts its right to be identified as the creator of this report in the United Kingdom. This report is considered by HashEye to be public information; it is licensed to Immutable Games under the terms of the project statement of work and has been made public at Immutable Games' request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of HashEye. The sole canonical source for HashEye publications is the HashEye Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic. Test Coverage Disclaimer All activities undertaken by HashEye in association with this project were performed in accordance with a statement of work and agreed upon project plan. Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase. HashEye uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project. HashEye 2 Immutable ZKEVM Bridge Security Assessment PUBLIC

Table of Contents About HashEye 1 Notices and Remarks 2 Table of Contents 3 Project Summary 4 Executive Summary 5 Codebase Maturity Evaluation 7 A. Code Maturity Categories 10 B. Code Quality Recommendations 12 C. Incident Response Recommendations 13 D. Security Best Practices for Using Multisignature Wallets 15 E. Upgradability Checks with Slither 17 F. Fix Review Results 19 Detailed Fix Review Results 20 Code Quality Fix Review Results 21 G. Fix Review Status Categories 22 HashEye 3 Immutable ZKEVM Bridge Security Assessment PUBLIC

Project Summary Contact Information The following project manager was associated with this project: Anne Marie Barry, Project Manager annemarie.barry@hasheye.io The following engineering director was associated with this project: Josselin Feist, Engineering Director, Blockchain josselin.feist@hasheye.io The following consultants were associated with this project: Guillermo Larregay, Consultant Priyanka Bose, Consultant guillermo.larregay@hasheye.io priyanka.bose@hasheye.io Elvis Skoždopolj, Consultant Robert Schneider, Consultant elvis.skozdopolj@hasheye.io robert.schneider@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event November 21, 2023 Pre-project kickoff call December 4, 2023 Delivery of report draft December 4, 2023 Report readout meeting December 14, 2023 Delivery of summary report with fix review appendix HashEye 4 Immutable ZKEVM Bridge Security Assessment PUBLIC

Executive Summary Engagement Overview Immutable Games engaged HashEye to review the security of its Immutable ZKEVM Bridge contracts. The bridge contracts facilitate the transfer of assets between two chains, Ethereum and Immutable, using Axelar as the General Message Passing (GMP) Layer. Users can bridge their assets from Ethereum to Immutable by depositing them into the bridge contract and minting a synthetic version of the assets on the Immutable chain. Unlocking the assets on the Ethereum chain requires burning the corresponding synthetic assets on the Immutable chain, in case of ERC-20 tokens, or depositing the assets, in case of native assets (IMX). A team of four consultants conducted the review from November 27 to November 30, 2023, for a total of two engineer-weeks of effort. With full access to source code and documentation, we performed static testing of the Immutable ZKEVM Bridge contracts and the deployment scripts provided, using automated and manual processes. Observations and Impact The bridge smart contracts are divided into well-defined components with minimal code duplication, accompanied by a threat model identifying various security considerations. During our review, we evaluated each system component for missing or incorrect access controls, issues related to arithmetic operations, reentrancy risks, transaction ordering risks, common upgradeability issues, spoofing of users via meta transactions, and execution of arbitrary message payloads. We reviewed historical bridge hacks and previous security reviews to identify whether the bridge contracts are susceptible to any known issues. The Axelar integration was only briefly reviewed because the Axelar contracts were out of scope for this review. During the review, we identified a medium-severity issue related to the withdrawal of user assets failing when the user provides a non-compatible receiver. We also found three informational issues related to the front-running resistance of the contract deployment scripts, signature validation in the EIP-712 implementation, and the integration of some ERC-20 tokens, respectively. Recommendations Based on the codebase maturity evaluation and findings identified during the security review, HashEye recommends that Immutable take the following steps: • Remediate the findings disclosed in this report. These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

HashEye 5 Immutable ZKEVM Bridge Security Assessment PUBLIC

- Continue to develop system documentation. The repository contains thorough documentation on the system, its components, the expected user flows, and various security considerations. This documentation could be improved by including additional security considerations related to the Axelar GMP contracts. Ensure that user-facing documentation outlines any risks related to interacting with the system, including issues related to user error.
- Improve the front-running resistance of contract initialization. Consider using a factory pattern or adding access controls to contract initialization to mitigate front-running risks.
- Continue to improve the testing suite. Ensure that the testing suite has full coverage for all contracts of the protocol. Consider using advanced testing techniques such as fuzzing and mutation testing to help identify issues that might be difficult to find via manual analysis, and ensure the robustness of the testing suite.

HashEye 6 Immutable ZKEVM Bridge Security Assessment PUBLIC

Codebase Maturity Evaluation HashEye uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs. Category Summary Result Arithmetic All contracts are compiled using Solidity version 0.8.19, so safe arithmetic operations are built in. No complex arithmetic is present in the in-scope contracts. The contracts have a single instance of unchecked arithmetic operations, which could benefit from additional documentation. Strong Auditing Events are emitted from critical state-changing functions and from bridging operations through Axelar, making the system status easy to monitor. However, we are unaware of any off-chain monitoring system. The protocol would benefit from such a system in that the correct functioning of the protocol could be tracked and alerts could be configured in case of suspicious activity. Additionally, creating an incident response plan to define a recovery process in case of a compromise would improve the protocol's security posture (see appendix C). Moderate Authentication / Access Controls The system implements multiple access control roles that follow the principle of least privilege. The roles and their responsibilities are documented, but creating additional documentation on the nature of the accounts that hold these roles, and the circumstances under which specific role actions will be executed, would be beneficial. Best practices for using multisignature wallets should be followed (see appendix D). Satisfactory HashEye 7 Immutable ZKEVM Bridge Security Assessment PUBLIC

Complexity Management The system architecture is simple, with well-defined components, a clear naming convention, and minimal code duplication. Some minor code quality issues are present in the codebase. Satisfactory Decentralization All contracts are upgradeable, so an account with upgrade permissions can take control of the whole system, including user funds stored in the bridge contract. There are no timelocks or other measures that could allow users to move their funds out

of the bridge if they decide not to accept a new upgrade. Additionally, certain authorized roles can modify critical contract states, such as updating the expected bridge adapter address or the bridge's expected chain ID, which could prevent users from withdrawing their assets. Weak Documentation Most system actions and components are well documented in the NatSpec documentation. However, there is a lack of documentation about Axelar integration and its risks. The EIP-712 implementation could benefit from additional documentation of the code, particularly the low-level assembly blocks. Satisfactory Low-Level Manipulation The system uses very limited low-level assembly, but the assembly blocks of the EIP712MetaTransaction implementation are insufficiently documented and are lacking in test coverage. Low-level data manipulation using ABI encoding and decoding is present on the Axelar integration. Moderate Testing and Verification The test suite is thorough, includes unit and integration tests, and provides test cases for normal protocol use and some known adversarial situations. The tests are run automatically as part of the CI, and the coverage results are posted to GitHub. However, there is no full coverage for statements and branches, so some parts of the code are not tested. Notably, the coverage for the EIP-712 meta transactions contract is 0%. Moderate HashEye 8 Immutable ZKEVM Bridge Security Assessment PUBLIC

Transaction Ordering We identified an issue related to the front-running resistance of upgradeable contracts during initialization. We did not identify any additional vectors that could lead to a loss of funds or a denial-of-service attack. Moderate HashEye 9 Immutable ZKEVM Bridge Security Assessment PUBLIC

A. Code Maturity Categories The following tables describe the code maturity categories and rating criteria used in this document. Code Maturity Categories Category Description Arithmetic The proper use of mathematical operations and semantics Auditing The use of event auditing and logging to support monitoring Authentication / Access Controls The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system Complexity Management The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions Decentralization The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades Documentation The presence of comprehensive and readable codebase documentation Low-Level Manipulation The justified use of inline assembly and low-level calls Testing and Verification The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage Transaction Ordering The system's resistance to transaction-ordering attacks Rating Criteria Rating Description Strong No issues were found, and the system exceeds industry standards. Satisfactory Minor issues were found, but the system is compliant with best practices. Moderate Some issues that may affect system safety were found. Weak Many issues that affect system safety were found. Missing A required component is missing, significantly affecting system safety. HashEye 10 Immutable ZKEVM Bridge Security Assessment PUBLIC

Not Applicable The category is not applicable to this review. Not Considered The category was not considered in this review. Further Investigation Required Further investigation is required to reach a meaningful conclusion. HashEye 11 Immutable ZKEVM Bridge Security Assessment PUBLIC

B. Code Quality Recommendations The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future. • Rename the `_roles` input variable of the `ChildAxelarBridgeAdaptor` and `RootAxelarBridgeAdaptor` contracts to avoid variable name shadowing from the `AccessControlUpgradeable` dependency. • Apply a consistent naming convention for function names by adding an underscore to the beginning of the `transferTokensAndEmitEvent` function of the `ChildERC20Bridge` contract. • Standardize the naming convention for functions granting and revoking roles in the `AdaptorRoles` contract. Some of the functions are lacking the `Role` suffix. • Some functions' Boolean return values are checked to add custom revert errors. However, there are functions—`ChildERC20.mint`, `ChildERC20.burn`, and `WIMX.transferFrom`—that either return `true`, or revert, causing the custom revert to never happen. • Add additional documentation to the `EIP712MetaTransaction` contract for the unchecked and assembly blocks. Evaluate the risk of overflowing the nonce via the `invalidateNext` function and how it can impact the system, document this risk, and mitigate it if necessary. • Implement the `_executeWithToken` function from the `AxelarExecutable` contract so that it explicitly reverts, instead of relying on the `validateContractCallAndMint` function of the `AxelarGateway` contract to revert if users call the `executeWithToken` function. HashEye 12 Immutable ZKEVM Bridge Security Assessment PUBLIC

C. Incident Response Recommendations This section provides recommendations on formulating an incident response plan. • Identify the parties (either specific people or roles) responsible for implementing the mitigations when an issue occurs (e.g., deploying smart contracts, pausing contracts, upgrading the front end, etc.). • Document internal processes for addressing situations in which a deployed remedy does not work or introduces a new bug. ◦ Consider documenting a plan of action for handling failed remediations. • Clearly describe the intended contract deployment

process. • Outline the circumstances under which Immutable Games will compensate users affected by an issue (if any). ◦ Issues that warrant compensation could include an individual or aggregate loss or a loss resulting from user error, a contract flaw, or a third-party contract flaw. • Document how the team plans to stay up to date on new issues that could affect the system; awareness of such issues will inform future development work and help the team secure the deployment toolchain and the external on-chain and off-chain services that the system relies on. ◦ Identify sources of vulnerability news for each language and component used in the system, and subscribe to updates from each source. Consider creating a private Discord channel in which a bot will post the latest vulnerability news; this will provide the team with a way to track all updates in one place. Lastly, consider assigning certain team members to track news about vulnerabilities in specific components of the system. • Determine when the team will seek assistance from external parties (e.g., auditors, affected users, other protocol developers, etc.) and how it will onboard them. ◦ Effective remediation of certain issues may require collaboration with external parties. • Define contract behavior that would be considered abnormal by off-chain monitoring solutions. HashEye 13 Immutable ZKEVM Bridge Security Assessment PUBLIC

It is best practice to perform periodic dry runs of scenarios outlined in the incident response plan to find omissions and opportunities for improvement and to develop “muscle memory.” Additionally, document the frequency with which the team should perform dry runs of various scenarios, and perform dry runs of more likely scenarios more regularly. Create a template to be filled out with descriptions of any necessary improvements after each dry run. HashEye 14 Immutable ZKEVM Bridge Security Assessment PUBLIC

D. Security Best Practices for Using Multisignature Wallets Consensus requirements for sensitive actions, such as spending funds from a wallet, are meant to mitigate the risks of the following: • Any one person overruling the judgment of others • Failures caused by any one person’s mistake • Failures caused by the compromise of any one person’s credentials For example, in a 2-of-3 multisignature wallet, the authority to execute an onlyRole setter function would require a consensus of two individuals in possession of two of the wallet’s three private keys. For this model to be useful, the following conditions are required : 1. The private keys must be stored or held separately, and access to each one must be limited to a unique individual. 2. If the keys are physically held by third-party custodians (e.g., a bank), multiple keys should not be stored with the same custodian. (Doing so would violate requirement #1.) 3. The person asked to provide the second and final signature on a transaction (i.e., the co-signer) should refer to a pre-established policy specifying the conditions for approving the transaction by signing it with his or her key. 4. The co-signer should also verify that the half-signed transaction was generated willingly by the intended holder of the first signature’s key. Requirement #3 prevents the co-signer from becoming merely a “deputy” acting on behalf of the first signer (forfeiting the decision-making responsibility to the first signer and defeating the security model). If the co-signer can refuse to approve the transaction for any reason, the due-diligence conditions for approval may be unclear. That is why a policy for validating transactions is needed. A verification policy could include the following: • A protocol for handling a request to co-sign a transaction (e.g., a half-signed transaction will be accepted only via an approved channel) • An allowlist of specific addresses allowed to be the payee of a transaction • A limit on the amount of funds spent in a single transaction or in a single day HashEye 15 Immutable ZKEVM Bridge Security Assessment PUBLIC

Requirement #4 mitigates the risks associated with a single stolen key. For example, say that an attacker somehow acquired the unlocked Ledger Nano S of one of the signatories. A voice call from the co-signer to the initiating signatory to confirm the transaction would reveal that the key had been stolen and that the transaction should not be co-signed. If the signatory were under an active threat of violence, he or she could use a duress code (a code word, a phrase, or another signal agreed upon in advance) to covertly alert the others that the transaction had not been initiated willingly, without alerting the attacker. HashEye 16 Immutable ZKEVM Bridge Security Assessment PUBLIC

E. Upgradability Checks with Slither HashEye developed the slither-check-upgradability tool to aid in the development of secure proxies; it performs safety checks relevant to both upgradeable and immutable delegatecall proxies. Consider using this tool during the development of the Immutable smart contracts’ codebase. • We recommend using the slither-check-upgradeability tool with each code update to avoid introducing bugs. For the best results, consider integrating it directly into the CI workflow. To run this tool on the code, use the following command: slither-check-upgradeability <path> <contract_name> [--new-contract-name <new_contract_name>] [--new-contract-filename <new_contract_filename>] [--proxy-name <proxy_name>] [--proxy-file-name <proxy_file_name>] • <path> is the base directory of the project if using a compilation framework (such as Hardhat or Foundry). If no framework is used, <path> is the file where <contract_name> is defined. • <contract_name> is the name of the former contract that will be upgraded. The tool will try to

automatically find a new version of the contract. Arguments surrounded by square brackets are optional and can be used in cases where the tool fails to find the necessary files:

- `--new-contract-name <new_contract_name>` is used to specify the contract name of the new implementation if it is different from the previous one.
- `--new-contract-filename <new_contract_filename>` is used to indicate the tool where the new contract implementation is if it does not find it automatically.
- `--proxy-name <proxy_name>` indicates the name of the proxy contract.
- `--proxy-file-name <proxy_file_name>` indicates the file location of the proxy contract.

slither-check-upgradeability . ContractV1 --new-contract-name ContractV2

Figure E.1: An example of how to use slither-check-upgradeability

HashEye 17 Immutable ZKEVM Bridge Security Assessment PUBLIC

For example, if a variable `a` is incorrectly added in the new implementation before other storage variables, slither-check-upgradeability will issue a warning like the one shown in figure E.2. ...

```
INFO:Slither: Different variables between ContractV1 (contracts/versions/ContractV1.sol#9-54) and ContractV2 (contracts/versions/ContractV2.sol#11-133)
ContractV1.__gap (contracts/versions/ContractV1.sol#15)
ContractV2.a (contracts/versions/ContractV2.sol#20)
Reference: https://github.com/crytic/slither/wiki/Upgradeability-Checks#incorrect-variables-w-ith-the-v2 ..
```

Figure E.2: An example slither-check-upgradeability output

- Use `slither-read-storage` with the new implementation to manually check that the expected storage layout matches the previous implementation. Running the command shown in figure E.3 for the previous and new implementations will list the storage locations for variables in both versions. Make sure that variables in the previous implementation have the same slot number in the upgraded version.

slither-read-storage . - -contract ContractV1 --table

Figure E.3: An example of how to use slither-read-storage

- If gaps are used in parent contracts, check the storage layout with `slither-read-storage` after adding new functionality and decrease the gap size accordingly. Make sure that storage is not overwritten or shifted in child contracts.
- Simulate the upgrade with a local network fork, verify that all storage variables have the expected values, and run the tests on the new implementation.

HashEye 18 Immutable ZKEVM Bridge Security Assessment PUBLIC

F. Fix Review Results

When undertaking a fix review, HashEye reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system. On December 12, 2023, HashEye reviewed the fixes and mitigations implemented by the Immutable team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue. In summary, the Immutable team has resolved all three issues and partially resolved one issue described in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Initialization of system components is vulnerable to front running	Resolved
2	Use of <code>ecrecover</code> is susceptible to signature malleability	Partially Resolved
3	Optional ERC-20 functions can cause reverts when mapping a new token	Resolved
4	Assets transferred with Axelar can be lost if the destination transaction cannot be executed	Resolved

HashEye 19 Immutable ZKEVM Bridge Security Assessment PUBLIC

Detailed Fix Review Results

TOB-IMM-1: Initialization of system components is vulnerable to front running Resolved in PR #74 by adding access control checks in the initialize functions of the following contracts: `ChildAxelarBridgeAdaptor` , `ChildERC20Bridge` , `RootAxelarBridgeAdaptor` , and `RootERC20Bridge` . These checks ensure that the address added during the deployment of the implementation contracts can only initialize them.

TOB-IMM-2: Use of `ecrecover` is susceptible to signature malleability Partially resolved in PR #75 through inline documentation. The Immutable team stated the following: Because the implementation is not practically susceptible to replay attacks, the team has chosen to mitigate concerns relating to this issue through additional comments to `EIP712MetaTransaction` .

TOB-IMM-3: Optional ERC-20 functions can cause reverts when mapping a new token Resolved in PR #71 . The updated contract code now checks whether a mapped ERC-20 token supports the optional public getter functions. If it does not, the call reverts and provides a more informative, custom error message. More explicit inline documentation has also been added to the code for this requirement.

TOB-IMM-4: Assets transferred with Axelar can be lost if the destination transaction cannot be executed Resolved in PR #73 , which includes inline documentation and the implementation of necessary checks at the front end of the Immutable bridge. However, the addition of these checks fell outside the scope of this fix review. In regard to these changes, the Immutable team stated the following: Due to the risks associated with the proposed technical solutions to mitigate this at the smart contract level, the team has implemented the following two measures:

1. The front end of Immutable Bridge now conducts necessary checks to verify that if the recipient on the destination side is a contract, it has a `receive` or a `fallback` function. If it does not, the user is blocked from starting the bridging transaction. It is noted that this check does not account for possible scenarios where the execution of those functions could fail on the destination chain for other reasons.
2. The risk of this scenario has been documented in the smart contracts.

HashEye 20 Immutable ZKEVM Bridge Security Assessment PUBLIC

Code Quality Fix Review Results We reviewed the list of fixes that address the Code Quality Recommendations identified in this report. All of these recommendations are resolved either through explicit code changes or through documentation. HashEye 21 Immutable ZKEVM Bridge Security Assessment PUBLIC

6. Fix Review Status Categories The following table describes the statuses used to indicate whether an issue has been sufficiently addressed. Fix Status Status Description Undetermined The status of the issue was not determined during this engagement. Unresolved The issue persists and has not been resolved. Partially Resolved The issue persists but has been partially resolved. Resolved The issue has been sufficiently resolved. HashEye 22 Immutable ZKEVM Bridge Security Assessment PUBLIC