HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Gotbit
Date: 09 June, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Gotbit
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	Bridge
Platform	EVM
Language	Solidity
Methodology	Link
Changelog	22.05.2023 - Initial Review 07.06.2023 - Second Review 09.06.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Introduction **4**
System Overview **4**
Executive Summary **5**
Risks **6**
Checked Items **7**
Findings **10**
Critical 10
C01. Funds Lock; Denial of Service 10
High 10
H01. Token Supply Manipulation 10
H02. Undocumented Functionality 11
H03. Inconsistent Data 11
H04. Funds Lock 12
H05. Inconsistent Data 12
Medium 13
M01. Inconsistent Data 13
M02. Inconsistent Data 13
M03. Coarse-grained Access Control 14
M04. Undocumented Functionality 14
M05. Highly Permissive Role Access 15
M06. Denial of Service 15
Low 15
L01. Contradiction 15
L02. Contradiction 16
L03. Contradiction 16
L04. Missing Validation 17
Informational 17
I01. Inefficient Gas Model 17
I02. Typos In Natspec 17
I03. Inefficient Gas Model 18
I04. Naming Convention 18
Disclaimers **19**
Appendix 1. Severity Definitions **20**
Risk Levels 20
Impact Levels 21
Likelihood Levels 21
Informational 21
Appendix 2. Scope **22**

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Gotbit (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

The audit of the scope consists of a bridge contract for ERC20 transfers between chains. The aim is to implement a bridge that provides transfers both between EVM chains and from EVM to non-EVM chains and vice versa.

The system works using signature verification using the EIP712 standard. The transactions are signed by a relayer to be ready for fulfillment.

The files in the scope:

- **BridgeAssist.sol** - The bridge contract that records requests for sending tokens between chains, and fulfills requests if signed by the relayer.

Privileged roles

- Relayer: is the backend contract address, which is used for signing transactions to fulfill.
- DefaultAdmin: can grant/revoke all roles, can set fee, feeWallet, limitPerSend, pause/unpause contract and withdraw tokens from contract.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- Description of the development environment is given.
- NatSpec is present.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions by several users are not tested thoroughly.

Security score

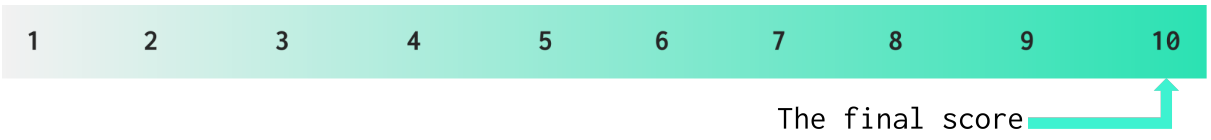
As a result of the audit, the code contains **no** issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10.**

The system users should acknowledge all the risks summed up in the risks section of the report.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
22 May 2023	4	6	5	1
07 June 2023	0	0	0	0
09 June 2023	0	0	0	0

Risks

- The project is highly centralized, with a Relayer role that controls the release of funds on each chain. If the private keys of the Relayer are lost or there is malicious intent, the protocol could be vulnerable to attack.
- The *fulfill()* function has the *whenNotPaused* modifier, if the contract is paused, users will not be able to receive their funds.
- There is a *withdraw()* function for the Admin role to withdraw any tokens from the contract, including the token that is bridged. If compromised, all tokens stored inside bridge contracts will be lost.
- The bridge fee can be changed between sending and fulfilling and has an upper limit of 99.99%. This may lead to the losses of funds during bridging.
- The chains have to be manually whitelisted by the admin and can be removed at any moment, rendering all pending transactions unclaimable.
- The bridge supports fee on transfers tokens and reflective tokens only if the bridge is excluded from the fees.
- The fee can be changed at any time by the owner, transfers that are already started, but not completely fulfilled will have the new fee value applied.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Passed	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

C01. Funds Lock; Denial of Service

Impact	High
Likelihood	High

There is no validation when setting up a new chain in the addChains() function to prevent the *exchangeRate* from being 0,

if this happens, in the *send()* function, there will be a division by 0 and the transaction will revert, causing a Denial of Service,

if this happens, in the *fulfill()* function, there will be a multiplication by 0, and the amount received by the user will be 0.

Path:
./contracts/BridgeAssist.sol : addChains()

Recommendation: Check that the *exchangeRate* is not 0.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

■■■ High

H01. Token Supply Manipulation

Impact	High
Likelihood	Medium

The default admin role can withdraw any sent ERC-20 tokens from the *BridgeAssist* contract at any time.

If the tokens are withdrawn this way after a fulfillment is confirmed from another chain, the tokens will be duplicated across chains.

This can lead to token supply manipulation.

Path:
./contracts/BridgeAssist.sol : withdraw()

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Recommendation: The tokens should not be withdrawable outside of bridge transaction fulfillment.

Found in: 2f76331

Status: Mitigated (with Customer notice: it is a centralized solution, to be used, user needs to trust it)

H02. Undocumented Functionality

Impact	**High**
Likelihood	Medium

It is not documented how the tokens that will be transferred to the users when bridged are added to the bridge contract.

This can lead to unexpected behavior, and the safety of the process cannot be validated.

Path:
./contracts/BridgeAssist.sol : fulfill()

Recommendation: Document how the tokens will be added in the destination chain contract.

Found in: 2f76331

Status: Mitigated (It is the responsibility of the project to make sure there is always enough liquidity on the bridge to sustain demand.

Customer response: "*Funds are added to the bridge by transferring them to the contract address. **If there is not enough funds on the contract, receiving funds will be impossible.** The admins are supposed to keep enough liquidity on the both end of the bridge so that this does not happen.*")

H03. Inconsistent Data

Impact	**High**
Likelihood	Medium

In the *setFees()* function, it is possible to change the fulfillment fee value after a transaction has already been sent from the originating chain.

The user will be affected by the new fee and is unable to estimate the fee that should be paid.

This can lead to a loss of trust in the bridge system.

Path:
./contracts/BridgeAssist.sol : setFees()

Recommendation: Transactions that are started before the *setFees()* call should not pay the new fee amount.

Found in: 2f76331

Status: Mitigated (Issue is documented as intended behavior. This is mentioned in the Risk section.

Customer response: "*The program should be able to take fees on sending and receiving funds. The fees should be changeable by the admin and unlimited (up to 99.99%). The recipient of the fees should be configurable by the admin as well. In case the fees are changed by the admin between sending and fulfilling a transaction, new fees should be used to make fee revenue more predictable.*")

H04. Funds Lock

Impact	**High**
Likelihood	Medium

The variable *fromChain* can be modified with the *removeChains()* function before calling the *fulfill()* function, so that a valid transaction would fail on the destination chain.

This will lead to funds being locked on the originating chain.

Path:
./contracts/BridgeAssist.sol : fulfill(), removeChain()

Recommendation: Document the behavior or implement a system of pending transactions that will not allow the default admin to remove a chain that has a pending transaction.

Found in: 2f76331

Status: Mitigated (Issue is documented as intended behavior. This is mentioned in the Risk section.)

H05. Inconsistent Data

Impact	**High**
Likelihood	Medium

the variable *exchangeRate* could be changed from when a transaction is sent to when a transaction is received, the user will not be able to estimate the received amount this way.

Path:
./contracts/BridgeAssist.sol : fulfill(), addChains(), removeChains();

Recommendation: Transactions that are started before the change in the *exchangeRate* should not use the new value.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

Medium

M01. Inconsistent Data

Impact	Low
Likelihood	High

The function performs division before multiplication during the fee calculation.

This leads to the loss of precision.

Path:
./contracts/BridgeAssist.sol : send()

Recommendation: Perform the multiplication before the division to avoid the loss of precision.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

M02. Inconsistent Data

Impact	Low
Likelihood	High

The event *SentTokens* is emitted in the *send()* function, but the provided data is not in line with the data stored in the Transaction struct. The event emits *(amount - currentFee) * exchangeRate* as amount, but the amount stored in the Transaction struct is *amount - currentFee*.

This inconsistency is not documented as desired behavior and can lead to incorrect data being emitted.

Path:
./contracts/BridgeAssist.sol : send()

Recommendation: Fix the discrepancy between the variables and the emitted event, or document this as intended.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

M03. Coarse-grained Access Control

Impact	Medium
Likelihood	Medium

The critical functions of the contract, such as token withdrawals and system property modifications, are accessible to the default admin.

Unauthorized access to these functions would be considered a security breach within the contract.

Path:
./contracts/BridgeAssist.sol

Recommendation: It is recommended to implement a multiple signature scheme for the admin role to enhance security measures. This scheme would require multiple admin approvals before any critical action can be executed, providing an extra layer of protection against unauthorized access.

Found in: 2f76331

Status: Mitigated (It is recommended in the documentation that the admin address should be a multi-sig wallet.

Customer response: "*We advise the admin role be only given to multisig wallets like Gnosis Safe for security reasons.*")

M04. Undocumented Functionality

Impact	High
Likelihood	Low

The *exchangeRate* is a variable that significantly alters how many tokens are received with the bridge, but the functionality is not documented.

Path:
./contracts/BridgeAssist.sol

Recommendation: Document the use and requirements of the *exchangeRate* variable.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

M05. Highly Permissive Role Access

Impact	High
Likelihood	Low

There are 2 fees retained from the original transaction of the user, one on the send transaction and one on the fulfilled transaction, both fees can go up to 99.99%, assuming that exchangeRate is 1:1 in both chains, the received amount could be 0.00000001% of the original amount.

Path:
./contracts/BridgeAssist.sol : setFee()

Recommendation: Put a reasonable boundary to the fees.

Found in: 2f76331

Status: Mitigated (Issue is documented as intended behavior. This is mentioned in the Risk section.)

M06. Denial of Service

Impact	High
Likelihood	Low

There is a possibility that the token of the bridge could have fees on transfer implemented in its contract.

This may result in Denial of Service when bridging and data inconsistency.

Path:
./contracts/BridgeAssist.sol

Recommendation: Ensure that the token used for bridging does not have fee-on-transfers or reflection tokens.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

Low

L01. Contradiction

Impact	Low
Likelihood	Medium

The variable *limitPerSend* and its usage is never documented.

Path:
./contracts/BridgeAssist.sol

Recommendation: Remove the functionality or mention it in documentation.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

L02. Contradiction

Impact	Low
Likelihood	Medium

The usage of the type string for variables that are referring to addresses is never documented.

Path:
./contracts/BridgeAssist.sol

Recommendation: Remove the functionality or mention it in documentation.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

L03. Contradiction

Impact	Low
Likelihood	Low

According to the NatSpec of the functions *getUserTransactions()* and *getUserTransactionsAmount()* they should return the transactions and amount of bridge transactions sent by `*user*` that have been sent from or fulfilled on the current chain.

However, in the implementation, only the sent chain is returned, not the fulfilled chain.

Path:
./contracts/BridgeAssist.sol : getUserTransactions(), getUserTransactionsAmount()

Recommendation: Fix the mismatch.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

L04. Missing Validation

Impact	Low
Likelihood	Medium

In the function *addChains()* two arrays are passed as arguments; if the length of both arrays is not the same, the transaction will revert.

Path:
./contracts/BridgeAssist.sol : addChains()

Recommendation: Check that the two arrays passed have the same length.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

Informational

I01. Inefficient Gas Model

In the *send()* function it is performed a division *amount / exchangeRate* and then it was performed a multiplication *amount * exchangeRate*.

Path:
./contracts/BridgeAssist.sol : send()

Recommendation: By doing the division after the multiplication line it is possible to avoid doing the multiplication altogether.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

I02. Typos In Natspec

There is a typo on line 335 in the NatSpec. It says *standart*, however, it should be *standard*.

Path:
./contracts/BridgeAssist.sol

Recommendation: Fix typos.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

I03. Inefficient Gas Model

In the *send()* function *block.timestamp* it is saved both in the event emitted and in the *transactions* mapping, this is a waste of Gas since this information is available in the block information and can be retrieved off-chain.

In the *fulfill()* function *block.timestamp* is saved in the event emitted.

Path:
./contracts/BridgeAssist.sol : send(), fulfill()

Recommendation: Remove the block.timestamp variable.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

I04. Naming Convention

The *TRANSACTION_TYPE_HASH* variable is misleading and reduces code readability, the proper naming convention when working with the EIP712 is to name them with function name + TYPEHASH for example FulfillTx should be FULFIL_TX_TYPEHASH

Path:
./contracts/BridgeAssist.sol

Recommendation: Change the name of the variable.

Found in: 2f76331

Status: Fixed (Revised commit: f205b8a)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/GotBit/bridge
Commit	2f7633185cf94c002480c0700942992551492dfe
Functional Requirements	https://github.com/GotBit/bridge/blob/main/contracts/README.md
Contracts	File: BridgeAssist.sol SHA3: 68c2636439c26eb5521d7913e0ce23dc0a050da2c4b31b6b97660af7ce231786

Second review scope

Repository	https://github.com/GotBit/bridge
Commit	f205b8a2df5f1fdebc5291a1a61f99689d428374
Functional Requirements	https://github.com/GotBit/bridge/blob/main/contracts/README.md
Technical Requirements	https://github.com/GotBit/bridge/blob/main/contracts/README.md Gotbit Ethereum Bridge Technical Requirements.pdf SHA3: e4ca7a85468033289468bc6069060fa2d8323cff3cd0d043883420daedcc8eb3
Contracts	File: contracts/BridgeAssist.sol SHA3: fad27bfe6252199b85a0f269f75f5ae580968a02dd26da7d096537f1e3d0a68e

Third review scope

Repository	https://github.com/GotBit/bridge
Commit	82eb229f24dbb0d9a30203886420d6bc62cedf95
Functional Requirements	https://github.com/GotBit/bridge/blob/main/contracts/README.md
Technical Requirements	https://github.com/GotBit/bridge/blob/main/contracts/README.md Gotbit Ethereum Bridge Technical Requirements.pdf SHA3: e4ca7a85468033289468bc6069060fa2d8323cff3cd0d043883420daedcc8eb3
Contracts	File: contracts/BridgeAssist.sol SHA3: de0ec97b064c9f0e5ee334e10cd6d7475418ff5740434c13116df0b0ffd61c16