Report

v. 2.0

Customer

Mystiko

Smart Contract Audit

Solidity and ZoKrates

15th March 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	06.03.23	A. Zveryanskaya	Initial Draft
0.2	06.03.23	A. Zveryanskaya	Minor revision
1.0	06.03.23	A. Zveryanskaya	Release
1.1	15.03.23	A. Zveryanskaya	CFV-11 typo fixed
2.0	15.03.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the contract and circuit structure, critical/major bugs detection and issuing the general recommendations.

Mystiko.Network is the base layer of web3 that provides both connectivity and confidentiality to all blockchain data, transactions and applications.

3 Project scope

We were asked to review:

- New functionality as a diff to the code
- After-audit fixes

Solidity files:

core/commitment/			
	CommitmentPool Main.sol	CommitmentPool.sol	CommitmentPool ERC20.sol
core/deposit/base/			
	MystikoV2Loop.sol		
core/deposit/loop/			
	MystikoV2Loop ERC20.sol	MystikoV2LoopMain.sol	
core/rule/			
	Sanctions.sol		
interface/			
	ICommitmentPool.sol	IHasher3.sol	IMystikoLoop.sol
	IVerifier.sol		
libs/asset/			
	AssetPool.sol	ERC20AssetPool.sol	IERC20Metadata.sol
	MainAssetPool.sol		
libs/common/			
	DataTypes.sol	CustomErrors.sol	

ZoKrates files:

/			
	Commitment.zok	ECIES.zok	JoinSplit.zok
	KeccakBatch.zok	MerkleTree.zok	MerkleTreeBatch Updater.zok
	MerkleTreeBuilder.zok	MerkleTreeUpdater.zok	Ownership.zok
	Rollup1.zok	Rollup4.zok	Rollup16.zok
	Rollup64.zok	Rollup256.zok	SecretSharing.zok
	SerialNumber.zok	Sha256Batch.zok	SignatureHash.zok
	Transaction1×0.zok	Transaction1×1.zok	Transaction1×2.zok
	Transaction2×0.zok	Transaction2×1.zok	Transaction2×2.zok

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract and circuit functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 5 major, and a few less important issues. All identified Major issues have been fixed.

	Active	Fixed
Major	0	5
Minor	16	9

Fixed 14 out of 30 issues

6 Major Issues

CVF-1. FIXED

- **Category** Suboptimal
- **Source** MystikoV2Loop.sol

Description Currently, min and max amounts could be set separately, but not both at once. Also, the contract doesn't allow "minAmount" to exceed "maxAmount". Thus it would be problematic to move the min-max amount range in case the new and the old ranges don't overlap.

Recommendation Consider implementing an ability to set both amounts at once.

Client Comment *New function updateDepositAmountLimits.*

```
+if (_minAmount > maxAmount) revert CustomErrors.
   ↪ MinAmountGreaterThanMaxAmount();
```

```
+if (_maxAmount < minAmount) revert CustomErrors.
   ↪ MaxAmountLessThanMinAmount();
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** JoinSplit.zok

Description This code is executed for each i whereas it should be executed only once.

Recommendation Consider refactoring.

Client Comment *Refactoring.*

```
+field[2] auditorPublicKey = [auditorPublicKeyXs[j],
   ↪ auditorPublicKeyYs[j]];
+assert(isOnCurve(auditorPublicKey, context));
+assert((auditorPublicKeyXs[j] > HALF_FIELD) ==
   ↪ auditorPublicKeyXSigns[j]);
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** JoinSplit.zok

Description This function every time checks that ‘randomSecretKey‘ is a DLOG of ‘randomPublicKey‘, thus making I*N total scalar multiplications. This is a huge overhead.

Recommendation Consider refactoring.

Client Comment *Refactoring.*

```
+assert(checkEncryption(commitmentShares[i][j], \
```

CVF-10. FIXED

- **Category** Unclear behavior
- **Source** CommitmentPool.sol

Description Here zero is silently returned for an invalid auditor index. Such behavior could hide errors.

Recommendation Consider reverting on invalid indexes.

Client Comment *Revert with AuditorIndexError.*

```
+return 0;
```

CVF-11. FIXED

- **Category** Suboptimal
- **Source** CommitmentPool.sol

Description Emitting events in a loop is usually a bad idea.

Recommendation Consider emitting a single event with array parameter.

Client Comment *Emit event EncryptedAuditorNotes with array parameter.*

```
+emit EncryptedAuditorNote(
```

7 Minor Issues

CVF-12. INFO

- **Category** Suboptimal
- **Source** Transaction1×0.zok

Recommendation It is unlikely that this array will be compressed by the compiler, so it can be easier and less error prone to just pass field elements.

Client Comment *Leave this as the boolean type, in case Zokrates optimize its compiler in the future version.*

```
+bool[NUM_AUDITORS] auditorPublicKeyXSigns, \
```

CVF-13. INFO

- **Category** Suboptimal
- **Source** Transaction1×1.zok

Recommendation It is unlikely that this array will be compressed by the compiler, so it can be easier and less error prone to just pass field elements.

Client Comment *Leave this as the boolean type, in case Zokrates optimize its compiler in the future version.*

```
+bool randomPublicKeyXSign, \
```

CVF-14. INFO

- **Category** Suboptimal
- **Source** Transaction1×2.zok

Recommendation It is unlikely that this array will be compressed by the compiler, so it can be easier and less error prone to just pass field elements.

Client Comment *Leave this as the boolean type, in case Zokrates optimize its compiler in the future version.*

```
+bool[NUM_AUDITORS] auditorPublicKeyXSigns, \
```

CVF-15. INFO

- **Category** Suboptimal
- **Source** Transaction2×0.zok

Recommendation It is unlikely that this array will be compressed by the compiler, so it can be easier and less error prone to just pass field elements.

Client Comment *Leave this as the boolean type, in case Zokrates optimize its compiler in the future version.*

```
+bool[NUM_AUDITORS] auditorPublicKeyXSigns, \
```

CVF-16. INFO

- **Category** Suboptimal
- **Source** Transaction2×1.zok

Recommendation It is unlikely that this array will be compressed by the compiler, so it can be easier and less error prone to just pass field elements.

Client Comment *Leave this as the boolean type, in case Zokrates optimize its compiler in the future version.*

```
+field[NUM_AUDITORS] auditorPublicKeyYs, \
```

CVF-17. INFO

- **Category** Suboptimal
- **Source** Transaction2×2.zok

Recommendation It is unlikely that this array will be compressed by the compiler, so it can be easier and less error prone to just pass field elements.

Client Comment *Leave this as the boolean type, in case Zokrates optimize its compiler in the future version.*

```
+bool[NUM_AUDITORS] auditorPublicKeyXSigns, \
```

CVF-18. INFO

- **Category** Procedural
- **Source** JoinSplit.zok

Description We didn't review this file.

```
+import "ecc/edwardsOnCurve" as isOnCurve;
```

CVF-19. INFO

- **Category** Suboptimal
- **Source** JoinSplit.zok

Description These functions always return true.

Recommendation Consider returning nothing.

Client Comment *To call the function of zok, you need to use a variable to receive the return value. If the return value is an empty tuple, can't define the variable type.*

```
+return true;
```

```
+return true;
```

```
+return true;
```

```
+return true;
```

```
+return true;
```

```
+return true;
```

CVF-20. INFO

- **Category** Suboptimal
- **Source** JoinSplit.zok

Recommendation Passing signs is needed only when the points are compressed. Seems they are not.

Client Comment *Passing uncompress key because compressed key may be great than field.*

```
+bool[N] auditorPublicKeyXSigns, \
```

```
+bool randomPublicKeyXSign, \
```

```
+field randomPublicKeyY, \
```

```
+field[N] auditorPublicKeyYs, \
```

CVF-21. INFO

- **Category** Procedural
- **Source** ECIES.zok

Description We did not review these files

```
+from "ecc/babyjubjubParams" import BabyJubJubParams;
+import "ecc/edwardsScalarMult" as scalarMult;
+import "hashes/poseidon/poseidon" as poseidon;
+import "utils/pack/bool/unpack256" as unpack256;
```

CVF-22. FIXED

- **Category** Bad naming
- **Source** ECIES.zok

Description Name is bad as the value is not shared: it is an ephemeral scalar in the DiffieHellman protocol.

Recommendation Consider renaming.

Client Comment *Change commonSecretKey to ephemeralScalar.*

```
+field commonSecretKey, \
```

CVF-23. INFO

- **Category** Suboptimal
- **Source** ECIES.zok

Description This check looks weird.

Recommendation Consider returning the check result or returning nothing.

Client Comment *To call the function of zok, you need to use a variable to receive the return value. If the return value is an empty tuple, can't define the variable type.*

```
+assert(encryptedMsg == expectedEncryptedMsg);
+return true;
```

CVF-24. INFO

- **Category** Procedural
- **Source** MerkleTreeUpdater.zok

Description We didn’t review this file.

```
+import "utils/pack/bool/unpack.zok" as unpack;
```

CVF-25. INFO

- **Category** Procedural
- **Source** KeccakBatch.zok

Description We didn’t review these files.

```
+import "hashes/keccak/256bit.zok" as keccak;
+import "utils/casts/u64_from_bits.zok" as u64_from_bits;
+import "utils/casts/u64_to_bits.zok" as u64_to_bits;
+import "utils/pack/bool/unpack256.zok" as unpack256;
+import "utils/pack/bool/pack256" as pack256;
```

CVF-26. INFO

- **Category** Procedural
- **Source** DataTypes.sol

Recommendation Consider specifying as “^0.8.0” unless there is something special about this particular version.

Client Comment *Maintain consistency with other files.*

```
+pragma solidity ^0.8.7;
```

CVF-27. FIXED

- **Category** Suboptimal
- **Source** CommitmentPool.sol

Recommendation The “id” parameter should be indexed.

Client Comment *emit event EncryptedAuditorNotes wiht array parameter.*

```
+event EncryptedAuditorNote(uint64 id, uint256 auditorPublicKey,
   ↪ uint256 encryptedAuditorNote);
```

CVF-28. FIXED

- **Category** Bad naming
- **Source** CommitmentPool.sol

Recommendation Events are usually named via nouns, such as “AuditorPublicKey”.

Client Comment *Change name to AuditorPublicKey.*

```
+event AuditorPublicKeyChanged(uint256 indexed index, uint256
   ↪ publicKey);
```

CVF-29. FIXED

- **Category** Suboptimal
- **Source** CommitmentPool.sol

Recommendation This condition could be simplified as “_index >= auditorCount”.

Client Comment *do “_index >= auditorCount” check*

```
+if (_index + 1 > auditorCount) revert CustomErrors.
   ↪ AuditorIndexError();
```

```
+if (_index + 1 > auditorCount) {
```

CVF-30. FIXED

- **Category** Procedural
- **Source** CommitmentPool.sol

Description The expression “previousIndex + 2” is calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

Client Comment *Refactoring.*

```
+inputs[previousIndex + 2 + i] = unpackedAuditorPublicKey.xSign;
```

CVF-31. FIXED

- **Category** Procedural
- **Source** CommitmentPool.sol

Description The expression “previousIndex + 2 + auditorCount” is calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

Client Comment *Refactoring.*

```
+inputs[previousIndex + 2 + auditorCount + i] =
   ↪ unpackedAuditorPublicKey.y;
```

CVF-32. FIXED

- **Category** Procedural
- **Source** CommitmentPool.sol

Description The expression “previousIndex + 2 + 2 * auditorCount” is calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

Client Comment *Refactoring.*

```
+inputs[previousIndex + 2 + 2 * auditorCount + i] = _request.
   ↪ encryptedAuditorNotes[i];
```

CVF-33. FIXED

- **Category** Bad naming
- **Source** Ownership.zok

Description Here ‘pk‘ is the x-coordinate of some point.

Recommendation Consider renaming.

Client Comment *Change pk to publicKeyX.*

```
+def main(field pk, field sk, BabyJubJubParams context) -> bool {
```

CVF-34. INFO

- **Category** Procedural
- **Source** SecretSharing.zok

Description We didn't review this file.

```
+import "utils/casts/u32_to_field" as u32_to_field;
```

CVF-35. FIXED

- **Category** Procedural
- **Source** SecretSharing.zok

Description The expression "u32_to_field(i + 1)" is calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

Client Comment *Refactoring.*

```
+accum = accum * u32_to_field(i + 1) + coefficients[K - 1 - j];
```

CVF-36. INFO

- **Category** Suboptimal
- **Source** SecretSharing.zok

Description The function always returns true.

Recommendation Consider returning nothing.

Client Comment *To call the function of zok, you need to use a variable to receive the return value. If the return value is an empty tuple, can't define the variable type.*

```
+return true;
```

ABDK Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting