Report

v. 1.0

Customer

Manta

Smart Contract Audit

Manta Network

17th July 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	17.07.23	A. Zveryanskaya	Initial Draft
0.2	17.07.23	A. Zveryanskaya	Minor revision
1.0	17.07.23	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 smart contracts structure, critical/major bugs detection and issuing the general recommendations.

After fixing the indicated issues the smart contracts should be re-audited.

3 Project scope

We were asked to review the code for the third audit and the following files:

manta-crypto/src/		
constraint.rs	accumulator.rs	
manta-crypto/src/merkle_tree/		
tree.rs	path.rs	node.rs
forest.rs		
manta-crypto/src/arkworks/		
groth16.rs	ff.rs	algebra.rs
manta-crypto/src/arkworks/constraint/		
fp.rs	mod.rs	

Then we reviewed the code for the fourth audit, including related files:

manta-pay/src/config/	
mod.rs	utxo.rs
manta-pay/src/crypto/encryption/	
aes.rs	

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 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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 1 critical, 13 major, and a few less important issues.

6 Critical Issues

CVF-1 INFO

- **Category** Overflow/Underflow
- **Source** mod.rs

Description Overflow is possible here, which allows arbitrary “remainder” value to satisfy this condition.

```
510 self.enforce_equal(&(quotient * &modulus + &remainder))
        .expect("This␣equality␣holds␣because␣of␣the␣Euclidean␣algorithm.
            ↪ ");
```

7 Major Issues

CVF-2 INFO

- **Category** Bad naming
- **Source** node.rs

Description Names “lhs” and “rhs” are confusing here, as they they could be swapped depending on the parity.

Recommendation Consider using neutral names such as “this side” and “that side”.

```
96  pub fn map<T, L, R>(self, lhs: L, rhs: R) -> T
```

```
109 pub const fn order<T>(&self, lhs: T, rhs: T) -> (T, T) {
```

```
122     lhs: &InnerDigest<C>,
        rhs: &InnerDigest<C>,
```

```
138     lhs: &LeafDigest<C>,
        rhs: &LeafDigest<C>,
```

CVF-3 INFO

- **Category** Procedural
- **Source** fp.rs

Recommendation Consider setting some bounds for the “F” type parameter.

```
56  pub struct Fp<F>(
```

CVF-4 INFO

- **Category** Suboptimal
- **Source** mod.rs

Description Here a value is converted into a binary representation of the full width, and then higher bits are enforced to be zero. This is an inefficient way to enforce a value to fit into certain number of bits. More efficient way would be to convert the value into a binary representation whose width is exactly BITS. This won't be possible unless the value actually fits into this number of bits.

```
318 let value_bits = value
        .to_bits_le()
320     .expect("Bit␣decomposition␣is␣not␣allowed␣to␣fail.");
    for bit in &value_bits[BITS..] {
        bit.enforce_equal(&Boolean::FALSE)
            .expect("Enforcing␣equality␣is␣not␣allowed␣to␣fail.");
    }
```

CVF-5 INFO

- **Category** Unclear behavior
- **Source** algebra.rs

Recommendation It should be ensured that there are sufficiently many bases to cover the bit length.

```
664 precomputed_bases: I,
```

```
678 for (bit, base) in scalar_bits.into_iter().zip(precomputed_bases.
      ↪ into_iter()) {
```

CVF-6 INFO

- **Category** Unclear behavior
- **Source** accumulator.rs

Description This function not only asserts something, but also returns a value.

Recommendation Consider reflecting this fact in the comment.

333
```
/// Asserts that `accumulator` can prove the membership of `item`
   ↪ after it is inserted.
```

335
```
pub fn assert_provable_membership<A>(accumulator: &mut A, item: &A::
   ↪ Item) -> Output<A>
```

CVF-7 INFO

- **Category** Suboptimal
- **Source** accumulator.rs

Recommendation Consider using “take(i)” instead of “skip(i + 1)” here, as “take” is usually more efficient.

375
```
for (j, y) in outputs.iter().enumerate().skip(i + 1) {
```

CVF-8 INFO

- **Category** Unclear behavior
- **Source** aes.rs

Recommendation A proper one-time-key GCM implementation should make sure that the key is never used twice, for example, by deleting the key after use.

102
```
impl<const P: usize, const C: usize> Encrypt for FixedNonceAesGcm<P,
   ↪  C> {
```

CVF-9 INFO

- **Category** Procedural
- **Source** utxo.rs

Description Using a zero domain tag is discouraged when multiple hash functions are used in order to avoid collisions and bind the context to the hash function call.

Recommendation Consider using a context-specific domain tag as recommend in the SAFE API. https://eprint.iacr.org/2023/522

```
277 Fp(0u8.into()) // FIXME: Use a real domain tag
```

```
410 Fp(0u8.into()) // FIXME: Use a real domain tag
```

```
1010 Fp(0u8.into()) // FIXME: Use a real domain tag
```

```
1131 Fp(0u8.into()) // FIXME: Use a real domain tag
```

```
1300 Fp(0u8.into()) // FIXME: Use a real domain tag
```

CVF-10 INFO

- **Category** Readability
- **Source** utxo.rs

Description Using tuples instead of structs makes code much harder to read.

Recommendation Consider either using structs or at least adding comments describing the semantics of tuple fields used in expressions, i.e. what “target.0”, “target.1”, “block[1].0” etc means here.

```
674 if target.0 && target.1.len() == 1 {
        let block = &target.1[0].0;
```

```
678             Fp(block[0].0),
                Asset::new(Fp(block[1].0), try_into_u128(block[2].0)?),
```

CVF-11 INFO

- **Category** Procedural
- **Source** utxo.rs

Recommendation 16 should be TAG_SIZE.

```
720 pub const AES_CIPHERTEXT_SIZE: usize = AES_PLAINTEXT_SIZE + 16;
```

CVF-12 INFO

- **Category** Unclear behavior
- **Source** utxo.rs

Description There are no length checks for this vector.

Recommendation Consider adding appropriate checks.

```
960 let bytes_vector = target?.0;
```

```
1663 let bytes_vector = target?.0;
```

CVF-13 INFO

- **Category** Suboptimal
- **Source** utxo.rs

Description This operation truncates some bits of the value to make it fit the field size.

Recommendation Consider using a value that always fits the field.

```
965 let utxo_randomness = Fp::<ConstraintField>::from_vec(
      ↪ utxo_randomness_bytes)
```

```
967 let asset_id = Fp::<ConstraintField>::from_vec(asset_id_bytes)
```

CVF-14 INFO

- **Category** Procedural
- **Source** utxo.rs

Recommendation 16 should be TAG_SIZE.

```
1424 pub const OUT_AES_CIPHERTEXT_SIZE: usize = OUT_AES_PLAINTEXT_SIZE +
        ↪ 16;
```

8 Moderate Issues

CVF-15 INFO

- **Category** Unclear behavior
- **Source** path.rs

Description These functions don't check right siblings to be default.

```
685 pub fn is_current(&self) -> bool
```

```
695 pub fn is_current_with(&self, default: &InnerDigest<C>) -> bool
```

CVF-16 INFO

- **Category** Procedural
- **Source** utxo.rs

Recommendation This should be resolved. The domain tag should be a type parameter or even a run-time parameter.

```
277 Fp(0u8.into()) // FIXME: Use a real domain tag
```

```
410 Fp(0u8.into()) // FIXME: Use a real domain tag
```

```
1010 Fp(0u8.into()) // FIXME: Use a real domain tag
```

```
1131 Fp(0u8.into()) // FIXME: Use a real domain tag
```

```
1300 Fp(0u8.into()) // FIXME: Use a real domain tag
```

CVF-17 INFO

- **Category** Documentation
- **Source** utxo.rs

Recommendation Consider explaining how 3 values, presumably field elements, yield 80 bytes.

```
925 target_plaintext.extend(source.utxo_commitment_randomness.to_vec());
    target_plaintext.extend(source.asset.id.to_vec());
    target_plaintext.extend(source.asset.value.to_le_bytes().to_vec());
```

CVF-18 INFO

- **Category** Procedural
- **Source** utxo.rs

Description The key is in one case converted to Fp explicitly, and the other case not.

Recommendation Consider a consistent approach.

```
1388 &Fp(proof_authorization_key.0.x),
     &Fp(proof_authorization_key.0.y),
```

```
1411 &proof_authorization_key.0.x,
     &proof_authorization_key.0.y,
```

CVF-19 INFO

- **Category** Suboptimal
- **Source** utxo.rs

Description Encryption and decryption functions should not be aware of the plaintext semantics.

Recommendation Consider passing just bytes to them.

```
1507 fn encrypt(
```

```
1521 fn decrypt(
```

CVF-20 INFO

- **Category** Procedural
- **Source** utxo.rs

Description Reducing the 256-bit hash modulo a 256-bit prime may give a non-uniform value that violates the security proof of the Schnorr signature.

Recommendation Consider using a 512-bit hash.

```
1795 Fp(EmbeddedScalarField::from_le_bytes_mod_order(&bytes))
```

9 Minor Issues

CVF-21 INFO

- **Category** Suboptimal
- **Source** node.rs

Recommendation This logic could be implemented using “Parity::map”.

191
```
match self.parity() {
    Parity::Left => Self(self.0 + 1),
    Parity::Right => Self(self.0 - 1),
}
```

203
```
match self.parity() {
    Parity::Left => (f(self), f(self + 1)),
    Parity::Right => (f(self - 1), f(self)),
}
```

220
```
match self.parity() {
    Parity::Left => *self,
    Parity::Right => Self(self.0 - 1),
}
```

231
```
match self.parity() {
    Parity::Left => Self(self.0 + 1),
    Parity::Right => *self,
}
```

CVF-22 INFO

- **Category** Suboptimal
- **Source** node.rs

Recommendation This formula is suboptimal. A more efficient approach would be to check that “lhs.0 » 1” is the same as “rhs.0 » 1”.

212
```
lhs.sibling().0 == rhs.0
```

CVF-23 INFO

- **Category** Suboptimal
- **Source** path.rs

Description Extracting inner path as an abstraction separate from full path looks like over-engineering.

Recommendation Consider merging into “Path”.

66
```
pub struct InnerPath<C>
```

CVF-24 INFO

- **Category** Procedural
- **Source** path.rs

Recommendation These functions should be members of the “Path” structure, rather than “InnerPath” as de-facto they operate with full paths.

127
```
pub fn root(
```

143
```
pub fn verify_digest(
```

CVF-25 INFO

- **Category** Procedural
- **Source** tree.rs

Recommendation As this type is used not only as an output, but also as an intermediary input, consider renaming into “InnerDigest”.

115
```
type Output;
```

CVF-26 INFO

- **Category** Readability
- **Source** mod.rs

Description It is confusing, that the error messages tell about variable allocations, while constants are created here.

Recommendation Consider rephrasing.

```
228 .expect("Variable␣allocation␣is␣not␣allowed␣to␣fail.")
```

```
337 .expect("Variable␣allocation␣is␣not␣allowed␣to␣fail.")
```

CVF-27 INFO

- **Category** Readability
- **Source** mod.rs

Description It is confusing that the messages tell about bitwise operations, while logical operations are performed on boolean values.

Recommendation Consider rephrasing.

```
279 self.and(&rhs).expect("Bitwise␣AND␣is␣not␣allowed␣to␣fail.")
```

```
292 self.or(&rhs).expect("Bitwise␣OR␣is␣not␣allowed␣to␣fail.")
```

CVF-28 INFO

- **Category** Readability
- **Source** algebra.rs

Description Returned value is not a bit representation (not a sequence of bits) but rather a big integer.

Recommendation Consider rephrasing.

```
68 /// Converts `scalar` to the bit representation of `0`.
```

CVF-29 INFO

- **Category** Suboptimal
- **Source** algebra.rs

Description This function is basically a compile-time constant. Calculating is at run time is suboptimal.

Recommendation Consider refactoring to calculate at compile time.

```
80 pub fn modulus_is_smaller<A, B>() -> bool
```

CVF-30 INFO

- **Category** Readability
- **Source** algebra.rs

Recommendation This message is confusing, as it tells about variable allocation, while a constant is created here.

```
590 .expect("Variable␣allocation␣is␣not␣allowed␣to␣fail."),
```

CVF-31 INFO

- **Category** Procedural
- **Source** accumulator.rs

Description “j” is not used.

Recommendation Consider using underscore (“_”).

```
375 for (j, y) in outputs.iter().enumerate().skip(i + 1) {
```

CVF-32 INFO

- **Category** Suboptimal
- **Source** aes.rs

Description This nonce is not random, but rather constant.

Recommendation Consider changing the value to not look like this joke: https://xkcd.com/221/

```
64 const NONCE: &'static␣[u8]␣=␣b"random␣nonce";
   ␣␣␣␣␣␣
```

CVF-33 INFO

- **Category** Suboptimal
- **Source** aes.rs

Description AES-256 is an overkill for an application that has at most 128 bits of security.

Recommendation Consider using AES-128.

```
113 Aes256Gcm::new_from_slice(encryption_key)
```

CVF-34 INFO

- **Category** Suboptimal
- **Source** utxo.rs

Recommendation Consider commenting that this type fits into Fp without loss.

```
364 asset_id,
```

```
393 asset_id,
```

```
633     source.asset.id,
```

CVF-35 INFO

- **Category** Bad datatype
- **Source** utxo.rs

Recommendation The type sizes should be named constants.

```
961 let utxo_randomness_bytes = bytes_vector[0..32].to_vec();
    let asset_id_bytes = bytes_vector[32..64].to_vec();
```

```
964     manta_util::Array::<u8, 16>::from_vec(bytes_vector[64..80].
           ↪ to_vec()).0;
```

CVF-36 INFO

- **Category** Procedural
- **Source** utxo.rs

Recommendation These values should be defined in configuration, rather then hardcoded.

```
1222 const HEIGHT: usize = 20;
```

```
1226 const HEIGHT: usize = 20;
```

CVF-37 INFO

- **Category** Bad datatype
- **Source** utxo.rs

Recommendation This string should be a named constant.

```
1251 hasher.update(b"manta-v1.0.0/merkle-tree-shard-function");
```

```
1713 hasher.update(b"manta-v1.0.0/address-partition-function");
```

```
1784 Digest::update(&mut hasher, b"manta-pay/1.0.0/Schnorr-hash");
```

CVF-38 INFO

- **Category** Bad datatype
- **Source** utxo.rs

Recommendation The type lengths should be named constants.

```
1664 let asset_id_bytes = bytes_vector[0..32].to_vec();
     let asset_value_bytes = manta_util::Array::<u8, 16>::from_vec(
```

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