Report

v. 3.0

Customer

Blockswap

Cryptography Audit

RPBS

12th June 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	24.05.23	A. Zveryanskaya	Initial Draft
0.2	24.05.23	A. Zveryanskaya	Minor revision
1.0	25.05.23	A. Zveryanskaya	Release
1.1	02.06.23	A. Zveryanskaya	CVF-18, 19, 23, 32, 33, 35, 36, 37, 42 marked as fixed
1.2	02.06.23	A. Zveryanskaya	C++ Original link description updated
1.3	02.06.23	A. Zveryanskaya	CVF-18 code added
1.4	02.06.23	A. Zveryanskaya	CVF-19, 23, 37 comments are updated
2.0	05.06.23	A. Zveryanskaya	Release
2.1	12.06.23	A. Zveryanskaya	Company description added
2.2	12.06.23	A. Zveryanskaya	Title page fix
3.0	12.06.23	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.

Blockswap Labs is a research and development firm making blockchain technology accessible to mainstream users. As core contributors to Blockswap Network and Proof of Neutrality Network, Blockswap Labs are building a permissionless middle layer and catalyzing web3 development through credibly neutral public benefit infrastructure solutions.

3 Project scope

We were asked to review several repositories:

- Original C++ Code (as reference)
- Original JavaScript Code
- Original Solidity Code

And corresponding fixes:

- C++ Fixes (as reference)
- JavaScript Fixes
- Solidity Fixes

Files:

/			
	main.cc	partial_signature.cc	partial_signature.h
	wapper.cc	wapper.h	
/			
	curveOperations.js	rpbs.js	
/			
	RPBS.sol		

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.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 6 critical, 13 major, and a few less important issues. All identified Critical issues have been fixed.

Issues	Active	Fixed
Severity		
Critical	**0**	**6**
Major	**5**	**8**
Moderate	**2**	**3**
Minor (no security issues)	**13**	**6**

Fixed 23 out of 43 issues

6 Critical Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** wapper.cc

Description This random number generator is not considered cryptographically secure: https://stackoverflow.com/questions/30473376/can-i-generate-cryptographically-ecure-random-data-from-a-combination-of-random.

Recommendation Consider using a secure random number generator.

Client Comment *We will ensure to use a cryptographically secure generator.*

```
40 std::mt19937 rng(std::random_device{}());
```

CVF-2. FIXED

- **Category** Flaw
- **Source** RPBS.sol

Description These values are secret to the signature holder (recipient). Revealing them breaks the blindness property.

```
125 uint256 alpha;
    uint256 beta;
```

```
141     _signature.alpha,
        _signature.beta
```

CVF-3. FIXED

- **Category** Flaw
- **Source** RPBS.sol

Description Contrast to the spec, the generator is not hashed in here.

```
169 abi.encodePacked(
```

CVF-4. FIXED

- **Category** Unclear behavior
- **Source** curveOperations.js

Description The decoder assumes that an "x" length always has two digits, while the encoder doesn't guarantee this. For points with x < 10 this could cause problems.

Client Comment *We will add extra checks.*

```
63 return point.getX().toString('hex').length.toString().concat(point.
      ↪ encode('hex'))
```

```
67 const xLength = Number(point.substring(0,2));
```

CVF-5. FIXED

- **Category** Flaw
- **Source** rpbs.js

Description Here y2 is a multiple of g, so the discrete log of y2 w.r.t. g is known. This allows forging a signature by computing s2' = c2'*Hash(forged_info) + log_g a2' Instead, one should use some hash-to-curve algorithm.

```
21 const y2 = curve.getCurvePointFromHash(sha256.hex(info));
```

CVF-6. FIXED

- **Category** Unclear behavior
- **Source** rpbs.js

Description 'g' is missing in the list of arguments

```
49 const digest = sha256
```

```
161 const digest = sha256
```

7 Major Issues

CVF-7. INFO

- **Category** Flaw
- **Source** partial_signature.cc

Description Here random numbers are generated inside methods, and callers have no control on this process.

Recommendation Consider provide some way how a caller may control random number generation, e.g. move it into a separate method that could be overridden.

Client Comment *This is only for demo purposes only, this code will never be used in prod.*

15
```
r1 = Scalar::random_scalar();
```

21
```
c2 = Scalar::random_scalar();
s2 = Scalar::random_scalar();
```

47
```
auto alpha = Scalar::random_scalar();
auto beta = Scalar::random_scalar();
```

51
```
u1 = Scalar::random_scalar();
v1 = Scalar::random_scalar();
```

57
```
u2 = Scalar::random_scalar();
v2 = Scalar::random_scalar();
```

CVF-8. FIXED

- **Category** Unclear behavior
- **Source** partial_signature.cc

Description In contrast to the spec, ‘g‘ is not hashed into.

63
```
sha256.update(y1)
```

105
```
auto sha256 = Sha256::create();
```

CVF-9. FIXED

- **Category** Flaw
- **Source** wapper.cc

Description The distribution is not sufficiently uniform when a 256-bit number is taken modulo a 25x-bit prime.

Recommendation Consider using 512bit numbers.

```
41 std::array<uint8_t, 32> words;
   for (int i = 0; i < 32; i++) {
```

CVF-10. FIXED

- **Category** Suboptimal
- **Source** wapper.cc

Description Generating one byte at a time is suboptimal.

Recommendation Consider generating as many bytes at a time as random number generator allows.

```
43 words[i] = rng();
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** wapper.cc

Description Consider reducing before storing to avoid ambiguous encoding.

Client Comment *In the demonstration, the ambiguity is avoided by simply selecting appropriate values. In the production scope this is not an issue since Javascript performs memory management for us.*

```
85 le::unsafe::store(&out[0], inner_);
```

CVF-12. INFO

- **Category** Suboptimal
- **Source** wapper.cc

Description Inverting via exponentiation is slow.

Recommendation Consider using the extended Euclidean algorithm.

Client Comment *Future optimisation.*

```
94 power(inner_, CURVE_ORDER - 2, mul, &result);
```

```
117 power(x, FIELD_MODULUS - 2, fq_mul, &result);
```

CVF-13. INFO

- **Category** Procedural
- **Source** wapper.cc

Recommendation This generator should be part of the Point public interface.

Client Comment *This is not a demonstration concern, the goal is to fix corectness in the C++ implementation.*

```
128 inline constexpr uint256 G1[] = {1, 2, 1};
```

CVF-14. FIXED

- **Category** Suboptimal
- **Source** RPBS.sol

Description This check is redundant, as the precompile anyway performs it.

Recommendation Consider removing this check or moving it after a failed precompile call to save gas on successful executions.

Client Comment *Good spot. We have removed this check.*

```
24 require(_isOnCurve(_point), 'Point not on the curve');
```

CVF-15. FIXED

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

Description The function returns GROUP_ORDER for 0.

Recommendation Consider returning 0 as well.

Client Comment *We will return 0 for 0 input.*

```
68 function negateScalar(uint256 _scalar) public pure returns (uint256)
   ↪  {
     return GROUP_ORDER - reduceScalar(_scalar);
70 }
```

CVF-16. FIXED

- **Category** Suboptimal
- **Source** RPBS.sol

Description This function is very inefficient.

Recommendation Consider using the approach described here: https://stackoverflow.com/questions/67893318/solidity-how-to-represent-bytes32-as-string/69266989#69266989.

```
97 function _bytesToHex(bytes memory _buffer) internal pure returns (
   ↪ string memory) {
```

CVF-17. INFO

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

Description The spec implies that the message is itself a point rather than a scalar.

Recommendation Consider clarifying.

Client Comment *We will clarify this in docs.*

```
195 Point memory messagePoint = scalarToPoint(reduceScalar(uint256(
   ↪ _messageHash)));
```

CVF-18. FIXED

- **Category** Unclear behavior
- **Source** curveOperations.js

Description Scalars under 2^256 mod CURVE_ORDER have higher probability to be generated by this function.

Recommendation Consider using an algorithm that produces all values with the same probability.

```
25 const getRandomScalar = () =>
```

New code:

```
25 const getRandomScalar = () =>
       new BN(randomHex(64).slice(2), 16).toRed(GROUP_REDUCTION)
```

CVF-19. FIXED

- **Category** Unclear behavior
- **Source** rpbs.js

Description These functions must be always called in a specific order with the Signer keeping the state of requests and not replying many times to the same message.

Recommendation Consider making the signer stateful.

Client Comment *Signer was made stateful.*

```
4   const commitToSignature = (message, info, privateKey) => {

27  const computeChallenge = (commitment, messagePoint, info, publicKey)
    ↪  => {

65  const solveChallenge = (challenge, commitment, privateKey) => {

72  const unblindCondition1 = (challenge, challengeSolution, commitment)
    ↪  => {

76  const unblindCondition2 = (challenge, challengeSolution, commitment,
    ↪  publicKey) => {

84  const unblindCondition3 = (challenge, challengeSolution, commitment,
    ↪  message) => {

92  const unblindCondition4 = (commitment, info) => {

100 const unblindSignature = (challenge, challengeSolution, commitment,
    ↪ publicKey, message, info) => {
```

8 Moderate Issues

CVF-20. INFO

- **Category** Suboptimal
- **Source** partial_signature.h

Recommendation The object should have a state variable for the number of the current round.

Client Comment *Future optimisation.*

```
39 class Recipient {
```

CVF-21. FIXED

- **Category** Overflow/Underflow
- **Source** wapper.cc

Description Underflow is possible here.

Recommendation Consider reducing before negating.

```
56 return Scalar(CURVE_ORDER - inner_);
```

CVF-22. FIXED

- **Category** Suboptimal
- **Source** wapper.cc

Description For non-reduced values these conditions could work incorrectly.

Recommendation Consider reducing before comparing.

```
76 return inner_ == other.inner_;
```

```
80 return inner_ != other.inner_;
```

CVF-23. FIXED

- **Category** Overflow/Underflow
- **Source** wapper.cc

Description Underflow is possible here.

Recommendation Consider reducing before subtracting.

Client Comment *We have fixed fq_neg, addmod should be fine since the intx (Extended precision integer C++ library) handles it.*

```
106   return addmod(x, FIELD_MODULUS - y, FIELD_MODULUS);
```

```
109 static auto fq_neg(const uint256 &x) -> uint256 { return
      ↪ FIELD_MODULUS - x; }
```

CVF-24. INFO

- **Category** Bad naming
- **Source** rpbs.js

Description ‘Commitment‘ means different in different function signatures. Sometimes it contains signer’s secrets, sometimes not.

Recommendation Consider naming appropriately.

Client Comment *Documentation and naming considerations.*

```
27 const computeChallenge = (commitment, messagePoint, info, publicKey)
   ↪  => {
```

```
65 const solveChallenge = (challenge, commitment, privateKey) => {
```

```
72 const unblindCondition1 = (challenge, challengeSolution, commitment)
   ↪  => {
```

```
76 const unblindCondition2 = (challenge, challengeSolution, commitment,
   ↪  publicKey) => {
```

```
84 const unblindCondition3 = (challenge, challengeSolution, commitment,
   ↪  message) => {
```

```
92 const unblindCondition4 = (commitment, info) => {
```

```
100 const unblindSignature = (challenge, challengeSolution, commitment,
   ↪ publicKey, message, info) => {
```

9 Minor Issues

CVF-25. INFO

- **Category** Suboptimal
- **Source** partial_signature.cc

Recommendation Sending ‘c2‘ is redundant, consider dropping it.

Client Comment *That’s just an optimality issue, but it does not change the demonstration correctness*

```
79 if (c1 * c2 != c) {
80   return std::nullopt;
}
```

CVF-26. INFO

- **Category** Suboptimal
- **Source** partial_signature.h

Description Representing a message as a string looks odd.

Recommendation Consider representing as an array of bytes.

```
10 static auto create(const Scalar &x1, const std::string &info) {
```

```
29 auto is_valid(const Point &y1, const std::string &info,
```

```
41 static auto create(const Point &m, const Point &y1, const std::
     ↪ string &info) {
```

```
52 Recipient(const Point &m, const Point &y1, const std::string &info);
```

CVF-27. INFO

- **Category** Documentation
- **Source** partial_signature.h

Recommendation Consider explaining which kind of signature is that.

```
28 struct Signature {
```

CVF-28. INFO

- **Category** Suboptimal
- **Source** wapper.cc

Description This multiplication is performed one extra time at the very last iteration.

Recommendation Consider refactoring to avoid redundant calculations.

Client Comment *We will leave it as is.*

```
22 base = mul(base, base);
```

CVF-29. INFO

- **Category** Bad naming
- **Source** wapper.h

Recommendation The names are confusing. Words “serialize”/”deserialize” or “marshal”/”unmarshal” would be more appropriate.

```
14 static auto load(std::string in) -> Scalar;
   auto save() const -> std::string;
```

```
41 static auto load(std::string in) -> Point;
   auto save() const -> std::string;
```

CVF-30. INFO

- **Category** Procedural
- **Source** wapper.h

Recommendation Consider naming the point class after the point representation: affine, projective, edwards, etc.

Client Comment *Point taken, we can implement that in the production scope.*

```
37 class Point {
```

CVF-31. INFO

- **Category** Suboptimal
- **Source** wapper.h

Description This method shouldn't be a part of the "Point" class, as different hash algorithms could be used.

Recommendation Consider implementing in an external hasher class.

Client Comment *For this signature scheme we are fixing the hash function to be sha256.*

```
46 static auto hash(std::string s) -> Point;
```

CVF-32. FIXED

- **Category** Procedural
- **Source** RPBS.sol

Recommendation This contract should be moved to a separate file named "Curve.sol".

```
6 abstract contract Curve {
```

CVF-33. FIXED

- **Category** Bad datatype
- **Source** RPBS.sol

Recommendation Precompile codes should be named constants.

```
30 if iszero(staticcall(gas(), 0x07, input, 0x60, output, 0x40)) {
```

```
52 if iszero(staticcall(gas(), 0x06, input, 0x80, output, 0x40)) {
```

CVF-34. INFO

- **Category** Bad datatype
- **Source** RPBS.sol

Recommendation The generator coordinates should be named constants.

```
41 return multiplyPointByScalar(Point({x: 1, y: 2}), _k);
```

CVF-35. FIXED

- **Category** Suboptimal
- **Source** RPBS.sol

Recommendation The ”reduceScalar” call here is redundant, as the ”scalarToPoint” function is able to deal with non-reduced scalars.

```
144 Point memory y2 = scalarToPoint(reduceScalar(uint256(_infoHash)));
```

```
195 Point memory messagePoint = scalarToPoint(reduceScalar(uint256(
    ↪ _messageHash)));
```

```
198     scalarToPoint(reduceScalar(_beta)),
```

CVF-36. FIXED

- **Category** Suboptimal
- **Source** RPBS.sol

Recommendation A common practice is to use ”assert” for sanity checks instead of ”require”.

```
146 /// Sanity check
    require(_isOnCurve(y2), 'y2 point not on the curve');
```

CVF-37. FIXED

- **Category** Suboptimal
- **Source** RPBS.sol

Recommendation It would be more efficient to concatenate binary images of the points and then convert convert the concatenation to hexadecimal representation.
Client comment We don’t have ‘encodePointHex‘ function now, we use ‘encodePoint‘.

```
169 abi.encodePacked(
170     encodePointHex(_publicKey),
    encodePointHex(y2),
    encodePointHex(m1_hat),
    encodePointHex(_signature.z1_hat),
    encodePointHex(input5),
    encodePointHex(input6),
    encodePointHex(input7)
)
```

CVF-38. INFO

- **Category** Suboptimal
- **Source** main.cc

Description This function looks like a test, rather than a production code.

Recommendation Consider either turning it into a normal test, or into a normal command-ine tool able to sign an verify signatures. In the current form it looks immature.

```
8 int main() {
```

CVF-39. INFO

- **Category** Suboptimal
- **Source** main.cc

Recommendation It would be better to use byte arrays instead. String are commonly used for something that could be printed.

```
11 // You can use `Scalar::save() -> std::string` method to serialize a
   ↪  Scalar
   // object into a std::string object. The std::string object is just
   ↪ a byte
   // buffer, don't try to use std::cout to print it.
```

CVF-40. INFO

- **Category** Suboptimal
- **Source** main.cc

Description These checks look weird.

Recommendation Consider removing them or moving into the “Scalar” constructor.

Client Comment *This is supposed to serve as a scalar conversion test, should not be used in the wild.*

```
25 assert(Scalar::load(x1.save()) == x1);
```

```
40 assert(Point::load(y1.save()) == y1);
```

CVF-41. INFO

- **Category** Procedural
- **Source** curveOperations.js

Description We didn't review these files.

```
1 const EC = require('elliptic');
  const BN = require('bn.js');
```

CVF-42. FIXED

- **Category** Suboptimal
- **Source** curveOperations.js

Description This doesn't allow the "0X" prefix.

Recommendation Consider allowing it .

```
38 let sliced = hash.startsWith('0x') ? hash.slice(2) : hash;
```

CVF-43. INFO

- **Category** Procedural
- **Source** rpbs.js

Description We didn't review this file.

```
2 const sha256 = require('js-sha256');
```

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