Report

v. 2.0

Customer

Blockswap

Cryptography Audit

CIP

12th June 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	07.06.23	A. Zveryanskaya	Initial Draft
0.2	07.06.23	A. Zveryanskaya	Minor revision
1.0	09.06.23	A. Zveryanskaya	Release
1.1	12.06.23	A. Zveryanskaya	Title page fix
1.2	13.06.23	A. Zveryanskaya	CVF-27. Client comment updated
2.0	13.06.23	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:

- Original Code
- Code with Fixes

Files:

/		
	main.cc	
include/		
	threshold.h	utils.h
src/		
	threshold.cc	utils.cc

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 4 critical, 14 major, and a few less important issues.

Issues	Active	Fixed
Severity **Critical**	1	3
Major	6	8
Moderate	3	3
Minor (no security issues)	13	17

Fixed 31 out of 54 issues

6 Critical Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** utils.cc

Description Contrast to the spec, the encryption key is derived as <recipient pub-key>*<sender priv key> . This is insecure.

Recommendation Consider doing a proper Diffie Hellman key exchange. you use the same private key for different derived keys. You will not be able to have a cryptographic security proof if you do that.
You have a correct encryption scheme in the spec, which works differently.

```
316 Binary Hybrid::Encrypt(const Point &public_key, const Binary &
        ↪ plaintext) const noexcept {
    /// I think we have exact function for that
318     Aes aes;
        auto key = public_key * private_key_;
320     aes.SetupKey(HashOnce(key.ToBinary()));
        return aes.Encrypt(plaintext);
    }
```

CVF-2. INFO

- **Category** Flaw
- **Source** utils.cc

Recommendation The spec requires AES-GCM here while AES-CBC is used.

Client Comment *We use AES-CBC in combination with the ECDSA signature which ensures the message integrity on transit. Encryptor and signer are supposed to be the same entity in the protocol.*

```
320 return aes.Encrypt(plaintext);
```

CVF-3. FIXED

- **Category** Flaw
- **Source** utils.cc

Recommendation All alleged points should be checked to satisfy the curve equation. The proof construction is insecure otherwise.

```
348 NizkVerify(const Point &pk1, const Point &pk2, const Point &key,
      ↪ const Binary &ciphertext, const Point &w1, const Point &w2,
      ↪ const BigNumber &z) {
```

CVF-4. FIXED

- **Category** Flaw
- **Source** threshold.cc

Description The resulting generator is a power of another generator, i.e. the discrete log between them is known. This makes the scheme insecure.

Recommendation Consider hashing n into the curve directly.

Client Comment *OpenSSL lacks Hash-To-Curve support. Here we use an approach from YACL library built on top of OpenSSL https://github.com/secretflow/yacl/blob/ed095f04255fe8c4466184b46c31531270220f18/yacl/crypto/base/ecc/openssl/openssl_group.cc#L260*

```
42 base_ = Point::FromBigInteger(n);
```

7 Major Issues

CVF-5. INFO

- **Category** Unclear behavior
- **Source** threshold.h

Recommendation The DKG object should be stateful and store the current round number and the encryption shares, rather than passing them between round function calls.

Client Comment *We acknowledge that, though this code is only for demonstration purposes only. It will be never used in production. In our production code (JS + Solidity) smart contracts track the current round number, though it is out of scope for this audit.*

```
12  class KeyGenerator {
```

CVF-6. FIXED

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

Recommendation The name is confusing. It should be called 'Reencryption'

```
118 Binary Decrypt() const {
        auto shared_key = encrypt_public_key_ * shared_private_key_;
120     return hybrid_.Encrypt(recipient_public_key_, shared_key.
            ↪ ToBinary());
```

CVF-7. FIXED

- **Category** Procedural
- **Source** threshold.h

Recommendation This TODO should be resolved or removed.

```
160 shared_key_[i] = Point::FromBinary(binary.value());
    /// TODO this might be the problem
```

CVF-8. INFO

- **Category** Procedural
- **Source** threshold.h

Recommendation These variables should be initialized in a constructor rather than in separate calls.

Client Comment *Acknowledged, but won't do it since it's not relevant in prod.*

```
319 size_t old_total_;
320 size_t old_threshold_;
    size_t new_threshold_;
    size_t index_;
    std::vector<Point> old_hybrid_key_list_;
    Hybrid hybrid_;
```

CVF-9. INFO

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

Description The shared private keys \overline{sk_j} are not public knowledge.

Recommendation Consider storing them in KeyGenerator objects

Client Comment *Not relevant in prod.*

```
126 std::tie(shared_private_key[i], shared_public_key[i], public_key[i])
       ↪  = generator[i].Round5(a_table);
```

CVF-10. INFO

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

Description Plain AES-CBC does not protect against adversaries that modify the ciphertext: the result will still decrypt to some valid plaintext. Moreover, only certain plaintext bits will be affected.

Recommendation Consider using an authenticated encryption mode such as GCM, OCB, or others.

Client Comment *We use AES-CBC in combination with the ECDSA signature which ensures the message integrity on transit. Encryptor and signer are supposed to be the same entity in the protocol.*

```
30 Binary Aes::Encrypt(const byte *key, const byte *iv, const Binary &
      ↪ plaintext) {
```

CVF-11. FIXED

- **Category** Bad naming
- **Source** utils.cc

Description Contrary to the name, the proof does not ensure that the ciphertext was created on a certain key. Instead, the proof only guarantees that 'key' is obtained from 'public_key' using the 'private_key_'

Client Comment *We've changed the function signature, do not use public key here according to spec.*

```
330 std::tuple<Point, Point, Point, BigNumber> Hybrid::NizkProof(const
      ↪ Point &public_key, const Binary &ciphertext) const noexcept {
```

CVF-12. FIXED

- **Category** Procedural
- **Source** utils.cc

Description The order of arguments is different in the spec: ciphertext is hashed before key=c2

```
336 hash.Update(Point::FromBigInteger(private_key_).ToBinary());
    hash.Update(public_key.ToBinary());
    hash.Update(key.ToBinary());
    hash.Update(w1.ToBinary());
340 hash.Update(w2.ToBinary());
    hash.Update(ciphertext);
```

CVF-13. FIXED

- **Category** Procedural
- **Source** threshold.cc

Description This function calculates a polynomial modulo the curve order, while the function signature gives absolutely no clue regarding this fact.

Recommendation Consider either passing the curve order as an additional argument or renaming the function.

Client Comment *Renamed to 'EvaluatePolynomial' as you suggested.*

```
3 BigNumber Polynomial(const std::vector<BigNumber> &coefficient,
    ↪ size_t x) {
```

CVF-14. INFO

- **Category** Procedural
- **Source** threshold.cc

Recommendation Modulo here would be redundant, if both, "item" and "power" would be below the curve order.

Client Comment *Not critical, acknowleded.*

```
7 result += item * power % Curve::GetOrder();
```

CVF-15. FIXED

- **Category** Flaw
- **Source** threshold.cc

Description No range check is made for the argument.

```
23 void KeyGenerator::SetIndex(size_t index) noexcept {
```

```
36 void KeyGenerator::SetThreshold(size_t threshold) noexcept {
```

CVF-16. FIXED

- **Category** Flaw
- **Source** threshold.cc

Description There is no check to ensure that ”hybrid_key_list” and ”hybrid_key_list_” are of the same length.

```
49 void KeyGenerator::SetHybridKeyList(const std::vector<Binary> &
   ↪ hybrid_key_list) {
```

CVF-17. INFO

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

Description It is unclear if the argument can be trusted to be of the right form. By default, all inputs should be validated, which does not hold here.

Client Comment *We acknowledge that, though this code is only for demonstration purposes only. It will be never used in production. In production we validate inputs (though it is out of scope for this audit) and in this PoC there is no way to pass an invalid input.*

```
90  std::vector<std::pair<size_t, Binary>> KeyGenerator::Round2(
      const std::vector<std::vector<Binary>> &e, const std::vector<
        ↪ Binary> &e_hat
```

```
136 bool KeyGenerator::Round2Check(
```

```
139   const std::vector<Binary> &e,
140   const Binary &e_hat,
      const Binary &evidence
```

```
179 std::vector<std::pair<size_t, Binary>> KeyGenerator::Round4(const
      ↪ std::vector<std::vector<Binary>> &a) {
```

```
203 bool KeyGenerator::Round4Check(size_t i, size_t j, const std::vector
      ↪ <Binary> &a, const Binary &e_hat, const Binary &evidence) {
```

```
237 std::tuple<Binary, Binary, Binary> KeyGenerator::Round5(const std::
      ↪ vector<std::vector<Binary>> &a) {
```

```
267 KeyGenerator::Round6(size_t new_total, size_t new_threshold, std::
      ↪ vector<Binary> &new_hybrid_key_list) {
```

CVF-18. FIXED

- **Category** Flaw
- **Source** utils.h

Description This class has no destructor which should erase the key from the memory.

Recommendation Consider adding one.

Client Comment *Added, now the keys are nulled.*

```
20 class Aes {
```

8 Moderate Issues

CVF-19. FIXED

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

Recommendation The name ‘shared_key‘ is confusing: it should be something related to encrypted private key shares.

```
148 std::vector<size_t> SetSharedKey(const std::vector<Binary> &
        ↪ shared_key) {
```

CVF-20. FIXED

- **Category** Suboptimal
- **Source** utils.cc

Recommendation The assignment is redundant as the variable will be overwritten at the next operation.

```
37 int out_len1 = ciphertext.size();
```

```
42 int out_len2 = ciphertext.size() - out_len1;
```

```
58 int out_len1 = recovered_text.size();
```

```
64 int out_len2 = recovered_text.size() - out_len1;
```

CVF-21. FIXED

- **Category** Suboptimal
- **Source** utils.cc

Recommendation Here it should be asserted that the ciphertext does not grow.

```
46 ciphertext.resize(out_len1 + out_len2);
```

CVF-22. INFO

- **Category** Procedural
- **Source** threshold.cc

Recommendation This multiplication should be calculated modulo the curve order. Otherwise ”power” could become really big for a large number of coefficients making the whole calculation very slow.

Client Comment *Type conversions are difficult and this is only for optimality reasons so leaving for now + Big Number limitations are very high.*

```
8 power *= x;
```

CVF-23. INFO

- **Category** Documentation
- **Source** utils.h

Recommendation Consider explaining which statement is proven.

Client Comment *Explained in the spec and paper.*

```
258 std::tuple<Point, Point, Point, BigNumber> NizkProof(const Point &
      ↪ public_key, const Binary &ciphertext) const noexcept;
```

CVF-24. INFO

- **Category** Documentation
- **Source** utils.h

Recommendation Consider explaining which statement is verified.

Client Comment *Explained in the spec and paper.*

```
261 NizkVerify(const Point &pk1, const Point &pk2, const Point &key,
      ↪ const Binary &ciphertext, const Point &w1, const Point &w2,
      ↪ const BigNumber &z);
```

9 Minor Issues

CVF-25. INFO

- **Category** Procedural
- **Source** threshold.h

Recommendation These two functions could be merged into a single template function.

Client Comment *Does not impact functionality.*

```
8  BigNumber Polynomial(const std::vector<BigNumber> &coefficient,
       ↪ size_t x);
```

```
10 Point Polynomial(const std::vector<Point> &coefficient, size_t x);
```

CVF-26. FIXED

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

Recommendation It is called ‘numerator‘.

Client Comment *Renamed.*

```
255 auto member = BigNumber::FromUInt64(1);
```

CVF-27. INFO

- **Category** Procedural
- **Source** threshold.h

Recommendation These multiplications should be done mod q to prevent blowup.

Client Comment *Overload for multiplication operation calls openssl function under the hood https://linux.die.net/man/3/bn_mul_word. This guarantees us to produce a valid big number without blowup.*

```
259 member *= j + 1;
260 denominator *= i - j;
```

```
263 member *= j + 1;
    denominator *= j - i;
```

CVF-28. FIXED

- **Category** Suboptimal
- **Source** threshold.h

Recommendation This is redundant.

Client Comment *Removed.*

```
280 assert(new_shared_public_key_ == Point::FromBigInteger(
       ↪ new_shared_private_key_));
```

CVF-29. INFO

- **Category** Procedural
- **Source** threshold.h

Recommendation This round should be merged with Round 1.

Client Comment *Irrelevant W.R.T prod and functionality.*

```
284 std::pair<Binary, Binary> Round2() {
        return std::make_pair(
          new_shared_private_key_.ToBinary(),
          new_shared_public_key_.ToBinary()
        );
    }
```

CVF-30. INFO

- **Category** Procedural
- **Source** main.cc

Recommendation This function implements a demo of the protocol and doesn't look like a production code.

Client Comment *That's the point, just for demo purposes only.*

```
14 int main() {
```

CVF-31. FIXED

- **Category** Documentation
- **Source** main.cc

Recommendation This check is not NIZK.

Client Comment *Removed.*

```
104 /// Perform NIZK checkfor the proof
```

CVF-32. FIXED

- **Category** Procedural
- **Source** main.cc

Recommendation This comment should be resolved or removed.

Client Comment *Removed.*

```
183 auto ciphertext = encryptor.Encrypt(plaintext);
    /// TODO looked here
```

CVF-33. FIXED

- **Category** Procedural
- **Source** utils.cc

Recommendation These commented lines should be removed or replaced with useful comments.

Client Comment *Removed.*

```
34 //    ThrowIf(rc != 1, "EVP_EncryptInit_ex error!");
```

```
40 //    ThrowIf(rc != 1, "EVP_EncryptUpdate error!");
```

```
44 //    ThrowIf(rc != 1, "EVP_EncryptFinal_ex error!");
```

```
54 //    ThrowIf(rc != 1, "EVP_DecryptInit_ex error!");
```

```
61 //    ThrowIf(rc != 1, "EVP_DecryptUpdate error!");
```

```
66 //    ThrowIf(rc != 1, "EVP_DecryptFinal_ex error!");
```

CVF-34. FIXED

- **Category** Documentation
- **Source** utils.cc

Recommendation Consider mentioning which endianness is used.

Client Comment *Mentioned in header file.*

```
178 BigNumber BigNumber::FromBinary(const std::array<byte, N> &bin) {
```

```
184 BigNumber BigNumber::FromBinary(const Binary &bin) {
```

```
190 BigNumber BigNumber::FromUInt64(uint64_t x) {
```

CVF-35. FIXED

- **Category** Procedural
- **Source** utils.cc

Description 'public_key' is c1 in the spec.

Recommendation Consider using consistent notation.

Client Comment *Renamed.*

```
330 std::tuple<Point, Point, Point, BigNumber> Hybrid::NizkProof(const
    ↪ Point &public_key, const Binary &ciphertext) const noexcept {
```

CVF-36. FIXED

- **Category** Procedural
- **Source** utils.cc

Description This is 'c2' in the spec.

Recommendation Consider using consistent notation.

Client Comment *Renamed.*

```
334 auto key = public_key * private_key_;
```

CVF-37. FIXED

- **Category** Bad naming
- **Source** threshold.cc

Recommendation It should be called ”EvaluatePolynomial”.

Client Comment *Renamed.*

```
13 Point Polynomial(const std::vector<Point> &coefficient, size_t x) {
```

CVF-38. INFO

- **Category** Suboptimal
- **Source** threshold.cc

Recommendation ‘+1‘ is not really necessary.

Client Comment *Acknowledged, but this just for demo purposes.*

```
73 s1_list_[index_] = Polynomial(a1_, index_ + 1);
   /// Evaluate polynomialfor your own index
```

```
78     auto s1 = Polynomial(a1_, i + 1);
   /// Evaluate polynomialfor every index except oneself
80     auto s2 = Polynomial(a2, i + 1);
```

CVF-39. FIXED

- **Category** Procedural
- **Source** threshold.cc

Description This index is j in the spec.

Recommendation Consider unifying.

Client Comment *Renamed.*

```
95 for (size_t i = 0; i < total_; i++) {
   /// Go over all indices secept oneself
```

CVF-40. FIXED

- **Category** Procedural
- **Source** threshold.cc

Description This index is I in the spec.

Recommendation Consider unifying.

Client Comment *Renamed.*

103
```
for (size_t j = 0; j < threshold_; j++) {
```

CVF-41. FIXED

- **Category** Procedural
- **Source** threshold.cc

Recommendation This commented line should be removed or replaced with a useful comment.

Client Comment *Removed.*

104
```
/// Reminder e[i][j] = Point::Create(a1_[i],
   base_,a2[i]).ToBinary();
```

CVF-42. INFO

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

Description This should be computed only if decryption fails.

Client Comment *For demonstration purposes we shall keep it and in general this takes milliseconds.*

111
```
OStream out;
auto[key, w1, w2, z] = hybrid_.NizkProof(public_key, e_hat[i]);
out << key << w1 << w2 << z;
evidence_[i] = out.ToBinary();
```

CVF-43. INFO

- **Category** Bad naming
- **Source** threshold.cc

Recommendation Consider renaming to ‘Handover()‘

Client Comment *We don’t even have this in prod, this is for demonstration purposes only.*

```
266 std::pair<std::vector<Binary>, std::vector<Binary>>
    KeyGenerator::Round6(size_t new_total, size_t new_threshold, std::
       ↪ vector<Binary> &new_hybrid_key_list) {
```

CVF-44. INFO

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

Description It is unclear what kind of AES is meant here.

Recommendation Consider naming for a particular mode of operation.

```
20 class Aes {
```

CVF-45. INFO

- **Category** Documentation
- **Source** utils.h

Recommendation Consider documenting what these functions are doing.

```
24    void SetupKey(const Block &key) noexcept;
```

```
26    Binary Encrypt(const Binary &plaintext) const;
```

```
28    std::optional<Binary> Decrypt(const Binary &ciphertext) const;
```

```
33    static Binary Encrypt(const byte *key, const byte *iv, const
         ↪ Binary &plaintext);
```

```
35    static std::optional<Binary> Decrypt(const byte *key, const byte
         ↪  *iv, const Binary &ciphertext);
```

```
99    BigNumber MulInvert(const BigNumber &mod) const noexcept;
```

```
172   Point(const Point &) = delete;
```

```
208   static Point Create(const BigNumber &n, const Point &q, const
         ↪ BigNumber &m);
```

```
215 Block HashOnce(const void *data, size_t length);
```

```
217 Block HashOnce(const Binary &binary);
```

```
250   void SetKey(const BigNumber &key) noexcept;
```

```
254   Binary Encrypt(const Point &public_key, const Binary &plaintext)
         ↪  const noexcept;
```

```
256   std::optional<Binary> Decrypt(const Point &public_key, const
         ↪ Binary &ciphertext) const noexcept;
```

```
258   std::tuple<Point, Point, Point, BigNumber> NizkProof(const Point
         ↪  &public_key, const Binary &ciphertext) const noexcept;
```

```
261   NizkVerify(const Point &pk1, const Point &pk2, const Point &key,
         ↪  const Binary &ciphertext, const Point &w1, const Point &
         ↪ w2, const BigNumber &z);
```

CVF-46. INFO

- **Category** Suboptimal
- **Source** utils.h

Recommendation This declaration misses some operators, such as ”-=”, ”/=”, ”>”, ”<” etc.

Client Comment *We don’t use them so we don’t need them.*

```
61 class BigNumber {
```

CVF-47. INFO

- **Category** Unclear behavior
- **Source** utils.h

Description This class should work with integers modulo q and named accordingly.

Client Comment *Just naming, clear from context.*

```
61 class BigNumber {
```

CVF-48. FIXED

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

Recommendation The name is confusing, as this function does modular exponentiation rather than simple exponentiation.

Client Comment *Resolved.*

```
97 BigNumber Power(const BigNumber &n, const BigNumber &mod) const
     ↪ noexcept;
```

CVF-49. FIXED

- **Category** Documentation
- **Source** utils.h

Description The particular binary format is unclear.

Recommendation Consider documenting.

Client Comment *Resolved*.

```
101 Binary ToBinary() const;
```

```
118 static BigNumber FromBinary(const std::array<byte, N> &bin);
```

```
120 static BigNumber FromBinary(const Binary &bin);
```

CVF-50. FIXED

- **Category** Documentation
- **Source** utils.h

Description The particular binary format is unclear.

Recommendation Consider documenting.

Client Comment *Resolved*.

```
200 Binary ToBinary() const;
```

```
204 static Point FromBinary(const Binary &bin);
```

CVF-51. FIXED

- **Category** Documentation
- **Source** utils.h

Description This function constructs a point [n]G + [m]q.

Recommendation Consider documenting.

Client Comment *Resolved*.

```
208 static Point Create(const BigNumber &n, const Point &q, const
    ↪ BigNumber &m);
```

CVF-52. INFO

- **Category** Documentation
- **Source** utils.h

Recommendation Consider documenting the input and output format of this primitive.

Client Comment *Looks like it is clear from the types these functions return.*

```
219 class Hash {
```

CVF-53. INFO

- **Category** Documentation
- **Source** utils.h

Recommendation Consider documenting which kind of encryption is used. Concretely, it should be AES-GCM with a secret key derived from ‘public_key‘.

Client Comment *We are currently using AES-CBC as stated previously, though the mode of operation can be seen in Encrypt function.*

```
254 Binary Encrypt(const Point &public_key, const Binary &plaintext)
      ↪ const noexcept;
```

```
256 std::optional<Binary> Decrypt(const Point &public_key, const Binary
      ↪ &ciphertext) const noexcept;
```

CVF-54. FIXED

- **Category** Procedural
- **Source** utils.h

Description The function always returns ”*this”.

Recommendation Consider coding this once at the very end of the function.

```
297     return *this;
```

```
304 return *this;
```

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