Report

v. 1.0

Customer

Composable Finance

Smart Contract Audit

zk-ed25519

28th October 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	28.10.24	A. Zveryanskaya	Initial Draft
0.2	28.10.24	A. Zveryanskaya	Minor revision
1.0	28.10.24	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:

- Original Code

Files:

curve_ed25519/		
curve_ed25519.go	curve_ ed25519circuit.go	curve_utils.go
element_f.go	element_o.go	element_q.go
element.go		
signature_verifier/		
Circuito64.go	Interface.go	SHA2-512.go
Signature.go		
curve_ed25519/		
curve_ed25519.go	curve_ed25519_test.go	curve_ed25519 _circuit_test.go
curve_ed25519 circuit.go	curve_utils.go	element.go
element_f.go	element_o.go	element_q.go
element_q_test.go	element_test.go	
signature_verifier/		
Circuito.go	Circuito16.go	Circuito32.go
Circuito48.go	Circuito64.go	Circuito_test.go
EntToEnd_test.go	Interface.go	SHA2-512.go
SHA2-512_test.go	Sample_test.go	Signature.go
measure_time_test.go	signature_test.go	signature_utils.go

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 23 critical, 32 major, and a few less important issues.

6 Critical Issues

CVF-1 INFO

- **Category** Flaw
- **Source** SHA2-512.go

Description The padding schema implemented by this function doesn't follow the standard.

```
12 func SHA2_512Circuit(uapi *uints.BinaryField[uints.U64], api
   ↪ frontend.API, input []uints.U8) [64]uints.U8 {
```

CVF-2 INFO

- **Category** Flaw
- **Source** SHA2-512.go

Description Index out of range error could happen here in case n is a factor of 128.

```
18 input8[n] = uints.NewU8(0x80)
```

CVF-3 INFO

- **Category** Flaw
- **Source** SHA2-512.go

Description Index out of range error could happen here in case n is zero.

```
20 input8[i] = uints.NewU8(uint8(L % 256))
```

CVF-4 INFO

- **Category** Flaw
- **Source** SHA2-512.go

Description An input value could be overwritten here in case n % 128 > 112.

```
20 input8[i] = uints.NewU8(uint8(L % 256))
```

CVF-5 INFO

- **Category** Flaw
- **Source** SHA2-512.go

Description The ‘ret‘ array values must be range checked.

```
55 ret, _ := api.Compiler().NewHint(HintDivMod64bits, 2, vres)
```

CVF-6 INFO

- **Category** Flaw
- **Source** Signature.go

Description The variable[] array values must be range checked.

```
37 variable, _ := api.Compiler().NewHint(frontendVariableToU8Hint, 32,
      ↪ v)
```

CVF-7 INFO

- **Category** Overflow/Underflow
- **Source** curve_utils.go

Description This check isn’t enough to ensure that res[0] = a % QC. One should also ensure that res[0] < QC and that res[1] * QC + res [0] doesn’t overflow.

```
71 api.AssertIsEqual(api.Add(api.Mul(res[1], QC), res[0]), a)
```

CVF-8 INFO

- **Category** Overflow/Underflow
- **Source** element_f.go

Recommendation It is not sufficient to check that c0+c1*2^128 equals a*b. It is necessary to check that both c0 and c1 are in the right range, i.e. c0<2^128 and c1*2^128+c0 <FieldModulus over integers

```
68 der := api.Add(
           c.V[0], api.Mul(c.V[1], FieldBaseC), api.Mul(res[2],
              ↪ FieldModulus))
70 api.AssertIsEqual(izq, der)
```

CVF-9 INFO

- **Category** Overflow/Underflow
- **Source** element_f.go

Recommendation It is not sufficient to check that c0+c1*2^128 equals a+b. It is necessary to check that both c0 and c1 are in the right range, i.e. c0<2^128 and c1*2^128+c0 <FieldModulus over integers

```
119 der := api.Add(c.V[0], api.Mul(FieldBaseC, c.V[1]), api.Mul(res[2],
       ↪ FieldModulusC))
120 api.AssertIsEqual(izq, der)
    return c
```

CVF-10 INFO

- **Category** Flaw
- **Source** element_f.go

Recommendation It should be checked that the ‘res‘ variables are either 0 or 1.

```
156 res, _ = api.Compiler().NewHint(HintBitsElementF, 256, a.V[0], a.V
       ↪ [1])
```

CVF-11 INFO

- **Category** Overflow/Underflow
- **Source** element_o.go

Recommendation It is not sufficient to check that c0+c1*2^128+res[2]*Ord equals a*b. It is necessary to check that both c0 and c1 are in the right range, i.e. c0<2^128 and c1*2^128+c0 <Ord over integers. It should be also checked that res[2] is in the right range

```
64 der := api.Add(
           c.V[0], api.Mul(c.V[1], FieldBaseC), api.Mul(res[2], OrdC))
   api.AssertIsEqual(izq, der)
```

CVF-12 INFO

- **Category** Overflow/Underflow
- **Source** element_o.go

Recommendation It is not sufficient to check that c0+c1*2^128 equals a+b. It is necessary to check that both c0 and c1 are in the right range, i.e. c0<2^128 and c1*2^128+c0 <FieldModulus over integers. It should be also checked that res[2] is in the right range

```
119
120 func AddElementsO(a []ElementO, api frontend.API) ElementO {
        var res ElementO
```

CVF-13 INFO

- **Category** Flaw
- **Source** element_o.go

Recommendation It should be checked that the ‘res‘ variables are either 0 or 1.

```
152 res, _ = api.Compiler().NewHint(HintBitsElementO, 253, a.V[0], a.V
    ↪ [1])
```

CVF-14 INFO

- **Category** Overflow/Underflow
- **Source** element_q.go

Recommendation It is not sufficient to check that c0+c1*2^128+res[2]*QC equals a*b. It is necessary to check that both c0 and c1 are in the right range, i.e. c0<2^128 and c1*2^128+c0 <QC over integers. It should be also checked that res[2] is in the right range

```
67 der := api.Add(
        c.V[0], api.Mul(c.V[1], FieldBaseC), api.Mul(res[2], QC))
api.AssertIsEqual(izq, der)
```

CVF-15 INFO

- **Category** Flaw
- **Source** element_q.go

Recommendation Both c[] and res[] array values must be appropriately range checked in order to be a well formed representation of a field element.

```
96 vd := api.Add(c.V[0], api.Mul(FieldBaseC, c.V[1]))
   vx := api.Add(res[2], api.Mul(FieldBaseC, res[3]))
```

CVF-16 INFO

- **Category** Flaw
- **Source** element_q.go

Recommendation The res[] array values must be appropriately range checked in order to be a well formed representation of a field element.

```
146 c := ElementQ{[2]frontend.Variable{res[0], res[1]}}
```

CVF-17 INFO

- **Category** Flaw
- **Source** element_q.go

Recommendation The res[] array values must be appropriately range checked in order to be a well formed representation of a field element.

```
171 c := ElementQ{[2]frontend.Variable{res[0], res[1]}}
```

CVF-18 INFO

- **Category** Flaw
- **Source** element_q.go

Recommendation It should be checked that the ‘res‘ variables are either 0 or 1.

```
212 res, _ = api.Compiler().NewHint(HintBitsElementQ, 256, a.V[0], a.V
      ↪ [1])
```

CVF-19 INFO

- **Category** Overflow/Underflow
- **Source** element.go

Recommendation It is not sufficient to check that c0+c1*2^128+res[2]*a.M[] equals a*b. It is necessary to check that both c0 and c1 are in the right range, i.e. c0<2^128 and c1*2^128+c0 <a.M[] over integers. It should be also checked that res[2] is in the right range

```
71 der := api.Add(
           c.V[0], api.Mul(c.V[1], FieldBaseC), api.Mul(res[2], api.Add
              ↪ (a.M[0], api.Mul(a.M[1], FieldBaseC))))
   api.AssertIsEqual(izq, der)
```

CVF-20 INFO

- **Category** Flaw
- **Source** element.go

Recommendation The res[] array values must be appropriately range checked in order to be a well formed representation of a field element.

```
124 c := Element{[2]frontend.Variable{res[0], res[1]}, a.M}
```

CVF-21 INFO

- **Category** Flaw
- **Source** element.go

Recommendation It should be checked that the ‘res‘ variables are either 0 or 1.

```
165 res, _ = api.Compiler().NewHint(HintBitsElement, 256, a.V[0], a.V
      ↪ [1])
```

CVF-22 INFO

- **Category** Flaw
- **Source** curve_ed25519circuit.go

Recommendation The ‘temp‘ variables must be range checked.

```
195 temp[31] = api.Sub(temp[31], arr[4])
```

```
201         y0 = api.Add(y0, temp[i])
            y1 = api.Add(y1, temp[i+16])
```

CVF-23 INFO

- **Category** Flaw
- **Source** curve_ed25519circuit.go

Recommendation The output point is not related to the intermediate computations. Perhaps it should be computed from y0,y1.

```
204 OnCurveCircuit(res, api)
```

7 Major Issues

CVF-24 INFO

- **Category** Unclear behavior
- **Source** Circuito64.go

Description There is no length check for the argument.

Recommendation Consider adding an appropriate range check.

```
25 func (circuit *Circuit64) SetSignatures(value []Signature) {
```

CVF-25 INFO

- **Category** Unclear behavior
- **Source** SHA2-512.go

Description There are no length checks for the array arguments.

Recommendation Consider adding appropriate checks.

```
42 func HintDivMod64bits(_ *big.Int, inputs []*big.Int, result []*big.
   ↪ Int) error {
```

CVF-26 INFO

- **Category** Bad datatype
- **Source** SHA2-512.go

Recommendation This value should be a named constant, rather than an expression.

```
44 pow := big.NewInt(0).Exp(big.NewInt(2), big.NewInt(64), nil)
```

```
57 pow := frontend.Variable(big.NewInt(0).Exp(big.NewInt(2), big.NewInt
   ↪ (64), nil))
```

CVF-27 INFO

- **Category** Unclear behavior
- **Source** Interface.go (ONLY YELLOW)

Description There are no length checks for the arguments.

Recommendation Consider adding appropriate checks to ensure all arrays are of the same length.

```
46 func InputToCircuit(circuit Interface, R []curve_ed25519.Point, S
   ↪ []*big.Int, A []curve_ed25519.Point, msg [][MLAR]byte)
   ↪ Interface {
```

CVF-28 INFO

- **Category** Unclear behavior
- **Source** Interface.go (ONLY YELLOW)

Recommendation The function should validate its inputs or explain where this validation happens.

```
46 func InputToCircuit(circuit Interface, R []curve_ed25519.Point, S
   ↪ []*big.Int, A []curve_ed25519.Point, msg [][MLAR]byte)
   ↪ Interface {
```

```
68 func Define(circuit Interface, api frontend.API) error {
```

CVF-29 INFO

- **Category** Suboptimal
- **Source** Interface.go (ONLY YELLOW)

Recommendation These values should be range checked before making an alleged point out of them.

```
74 R := curve_ed25519.CompressToPointCircuit(Rc[:], api, uapi)
   A := curve_ed25519.CompressToPointCircuit(Ac[:], api, uapi)
```

CVF-30 INFO

- **Category** Suboptimal
- **Source** Signature.go

Description Making all signatures public inputs does not make sense as the Verifier can verify them without SNARKs.

Recommendation Consider either verifying off-circuit or making all signatures private inputs.

```
17  Rc    [32]uints.U8                  `gnark:",public"`
    Sc    [32]uints.U8                  `gnark:",public"`
    Ac    [32]uints.U8                  `gnark:",public"`
20  Msg   [MLAR]uints.U8                `gnark:",public"`
    Input [InputLarge]frontend.Variable `gnark:",public"`
```

CVF-31 INFO

- **Category** Suboptimal
- **Source** Signature.go

Description This variable seems to be redundant.

Recommendation Consider removing it.

```
21  Input [InputLarge]frontend.Variable `gnark:",public"`
```

CVF-32 INFO

- **Category** Suboptimal
- **Source** Signature.go

Recommendation This should be initialized with ”variable[0]”, the loop below should start at index 1, and ”ret[0]” should be assigned outside the loop. This would save one ”Add” and one ”Mul” operation.

```
38  v2 := frontend.Variable(0)
```

CVF-33 INFO

- **Category** Suboptimal
- **Source** curve_utils.go

Recommendation It would be more efficient to use the DivMod function: https://pkg.go.dev/math/big#Int.DivMod

```
63 result[0].Mod(inputs[0], inputs[1])
   result[1].Div(inputs[0], inputs[1])
```

CVF-34 INFO

- **Category** Unclear behavior
- **Source** curve_ed25519.go

Description There is no length check for ”b”.

Recommendation Consider adding an appropriate length check.

```
72 func CompressToPoint(b []byte) Point {
```

CVF-35 INFO

- **Category** Documentation
- **Source** curve_ed25519.go

Recommendation It should be documented which inputs are admissible for this function and how it behaves on the other.

```
72 func CompressToPoint(b []byte) Point {
```

CVF-36 INFO

- **Category** Unclear behavior
- **Source** curve_ed25519.go

Description There is no length check for ”b”.

Recommendation Consider adding an appropriate length check.

```
108 func BytesToPoint(b []byte) Point {
```

CVF-37 INFO

- **Category** Unclear behavior
- **Source** element_f.go

Description There are no length checks for the ”inputs” and ”result” arguments.

Recommendation Consider adding appropriate checks.

```
48 func HintProductF(_ *big.Int, inputs []*big.Int, result []*big.Int)
   ↪ error {
```

```
83 func HintInverseF(_ *big.Int, inputs []*big.Int, result []*big.Int)
   ↪ error {
```

```
103 func HintAddF(_ *big.Int, inputs []*big.Int, result []*big.Int)
   ↪ error {
```

```
138 func HintBitsElementF(_ *big.Int, inputs []*big.Int, result []*big.
   ↪ Int) error {
```

```
172 func HintElementToUint8F(_ *big.Int, inputs []*big.Int, result []*
   ↪ big.Int) error {
```

CVF-38 INFO

- **Category** Suboptimal
- **Source** element_f.go

Description Multiplying by FieldModulus always gives 0 and can be omitted

```
69 c.V[0], api.Mul(c.V[1], FieldBaseC), api.Mul(res[2], FieldModulus))
```

CVF-39 INFO

- **Category** Flaw
- **Source** element_f.go

Description This doesn’t work is ”a” is empty.

Recommendation Consider returning the identity element in such case.

```
76 res = a[0]
```

CVF-40 INFO

- **Category** Suboptimal
- **Source** element_f.go

Description Multiplying by FieldModulus always gives 0 and can be omitted

```
119 der := api.Add(c.V[0], api.Mul(FieldBaseC, c.V[1]), api.Mul(res[2],
    ↪ FieldModulusC))
```

CVF-41 INFO

- **Category** Flaw
- **Source** element_f.go

Description This doesn't work is "a" is empty.

Recommendation Consider returning the zero element in such case.

```
126 res = a[0]
```

CVF-42 INFO

- **Category** Unclear behavior
- **Source** element_o.go

Description There are no length checks for the "inputs" and "result" arguments.

Recommendation Consider adding appropriate checks.

```
44 func HintProductO(_ *big.Int, inputs []*big.Int, result []*big.Int)
    ↪ error {
```

```
79 func HintInverseO(_ *big.Int, inputs []*big.Int, result []*big.Int)
    ↪ error {
```

```
99 func HintAddO(_ *big.Int, inputs []*big.Int, result []*big.Int)
    ↪ error {
```

```
134 func HintBitsElementO(_ *big.Int, inputs []*big.Int, result []*big.
    ↪ Int) error {
```

```
167 func HintElementToUint8O(_ *big.Int, inputs []*big.Int, result []*
    ↪ big.Int) error {
```

CVF-43 INFO

- **Category** Suboptimal
- **Source** element_o.go

Description This doesn't work is "a" is empty.

Recommendation Consider returning the identity element in such case.

```
72 res = a[0]
```

CVF-44 INFO

- **Category** Suboptimal
- **Source** element_o.go

Description This doesn't work is "a" is empty.

Recommendation Consider returning the zero element in such case.

```
122 res = a[0]
```

CVF-45 INFO

- **Category** Suboptimal
- **Source** element_o.go

Description Calculating 2^i on every iteration is suboptimal.

Recommendation Consider just multiplying the previous value by 2.

```
156 izq = api.Select(res[i], api.Add(izq, frontend.Variable(big.NewInt
    ↪ (0).Exp(big.NewInt(2), big.NewInt(int64(i)), nil))), izq)
```

CVF-46 INFO

- **Category** Unclear behavior
- **Source** element_q.go

Description There are no length checks for the ”inputs” and ”result” arguments.

Recommendation Consider adding appropriate checks.

```
44 func HintProductQ(_ *big.Int, inputs []*big.Int, result []*big.Int)
   ↪ error {
```

```
73 func HintDivQ(_ *big.Int, inputs []*big.Int, result []*big.Int)
   ↪ error {
```

```
112 func HintInverseQ(_ *big.Int, inputs []*big.Int, result []*big.Int)
   ↪ error {
```

```
132 func HintAddQ(_ *big.Int, inputs []*big.Int, result []*big.Int)
   ↪ error {
```

```
153 func HintSubQ(_ *big.Int, inputs []*big.Int, result []*big.Int)
   ↪ error {
```

```
194 func HintBitsElementQ(_ *big.Int, inputs []*big.Int, result []*big.
   ↪ Int) error {
```

```
228 func HintElementToUint8Q(_ *big.Int, inputs []*big.Int, result []*
   ↪ big.Int) error {
```

CVF-47 INFO

- **Category** Suboptimal
- **Source** element_q.go

Description This doesn’t work is ”a” is empty.

Recommendation Consider returning the identity element in such case.

```
105 res = a[0]
```

CVF-48 INFO

- **Category** Suboptimal
- **Source** element_q.go

Description This doesn't work is "a" is empty.

Recommendation Consider returning the zero element in such case.

```
182 res = a[0]
```

CVF-49 INFO

- **Category** Overflow/Underflow
- **Source** element_q.go

Description This addition may overflow in the most significant byte.

Recommendation Consider forbidding it explicitly or documenting why it is okay.

```
245 check = api.Add(api.Mul(check, frontend.Variable("256")), temp[i])
```

CVF-50 INFO

- **Category** Unclear behavior
- **Source** element.go

Description There are no length checks for the "inputs" and "result" arguments.

Recommendation Consider adding appropriate checks.

```
47  func HintProduct(_ *big.Int, inputs []*big.Int, result []*big.Int)
      ↪ error {
```

```
86  func HintInverse(_ *big.Int, inputs []*big.Int, result []*big.Int)
      ↪ error {
```

```
107 func HintAdd(_ *big.Int, inputs []*big.Int, result []*big.Int) error
      ↪  {
```

```
147 func HintBitsElement(_ *big.Int, inputs []*big.Int, result []*big.
      ↪ Int) error {
```

```
181 func HintElementToUint8(_ *big.Int, inputs []*big.Int, result []*big
      ↪ .Int) error {
```

CVF-51 INFO

- **Category** Flaw
- **Source** element.go

Description This doesn't work is "a" is empty.

Recommendation Consider returning the identity element in such case.

```
79 res = a[0]
```

CVF-52 INFO

- **Category** Flaw
- **Source** element.go

Description This doesn't work is "a" is empty.

Recommendation Consider returning the zero element in such case.

```
133 res = a[0]
```

CVF-53 INFO

- **Category** Documentation
- **Source** element.go

Description This addition may overflow in the most significant byte.

Recommendation Consider forbidding it explicitly or documenting why it is okay.

```
198 check = api.Add(api.Mul(check, frontend.Variable("256")), temp[i])
```

CVF-54 INFO

- **Category** Suboptimal
- **Source** curve_ed25519circuit.go

Recommendation Most of iterations work with small numbers and can be computed without modular reduction.

```
105 res = ProdElementQ(res, StringToElementQ("256"), api)
    res = AddElementQ(res, ElementQ{[2]frontend.Variable{hash[i].Val,
       ↪ frontend.Variable(0)}}, api)
```

```
119 res = ProdElementO(res, StringToElementO("256"), api)
120 res = AddElementO(res, ElementO{[2]frontend.Variable{hash[i].Val,
       ↪ frontend.Variable(0)}}, api)
```

```
133 res = ProdElement(res, StringToElement("256", mod), api)
    res = AddElement(res, Element{[2]frontend.Variable{hash[i].Val,
       ↪ frontend.Variable(0)}, res.M}, api)
```

CVF-55 INFO

- **Category** Unclear behavior
- **Source** curve_ed25519circuit.go

Description There are no length checks for arguments.

Recommendation Consider adding appropriate length checks.

```
147 func HintGetX(_ *big.Int, inputs []*big.Int, result []*big.Int)
       ↪ error {
```

8 Moderate Issues

CVF-56 INFO

- **Category** Unclear behavior
- **Source** curve_ed25519.go

Recommendation There is no handler for an error here, consider adding it.

```
96 res.X = big.NewInt(0).ModSqrt(left, Q)
```

CVF-57 INFO

- **Category** Overflow/Underflow
- **Source** element_f.go

Description Overflow is possible here which makes the binary representation non-deterministic.

Recommendation Consider using additional constraints to prevent overflow.

```
161 izq = api.Select(res[i], api.Add(izq, base), izq)
    base = api.Mul(base, frontend.Variable(2))
```

```
189 check = api.Add(api.Mul(check, frontend.Variable("256")), temp[i])
```

CVF-58 INFO

- **Category** Overflow/Underflow
- **Source** element_o.go

Description Overflow is possible here which makes the binary representation non-deterministic.

Recommendation Consider using additional constraints to prevent overflow.

```
156 izq = api.Select(res[i], api.Add(izq, frontend.Variable(big.NewInt
      ↪ (0).Exp(big.NewInt(2), big.NewInt(int64(i)), nil))), izq)
```

```
184 check = api.Add(api.Mul(check, frontend.Variable("256")), temp[i])
```

CVF-59 INFO

- **Category** Overflow/Underflow
- **Source** element_q.go

Description Overflow is possible here which makes the binary representation non-deterministic.

Recommendation Consider using additional constraints to prevent overflow.

217
```
izq = api.Select(res[i], api.Add(izq, base), izq)
base = api.Mul(base, frontend.Variable(2))
```

245
```
check = api.Add(api.Mul(check, frontend.Variable("256")), temp[i])
```

CVF-60 INFO

- **Category** Overflow/Underflow
- **Source** element.go

Description Overflow is possible here which makes the binary representation non-deterministic.

Recommendation Consider using additional constraints to prevent overflow.

170
```
izq = api.Select(res[i], api.Add(izq, base), izq)
base = api.Mul(base, frontend.Variable(2))
```

198
```
check = api.Add(api.Mul(check, frontend.Variable("256")), temp[i])
```

CVF-61 INFO

- **Category** Documentation
- **Source** curve_ed25519circuit.go

Recommendation This weird comment should be removed.

208
```
//ssh -i pub_rsa lautaro@34.118.49.208
```

9 Minor Issues

CVF-62 INFO

- **Category** Procedural
- **Source** Circuito64.go

Recommendation The file name is inconsistent with the struct name. Looks like a typo.

```
7 type Circuit64 struct {
```

CVF-63 INFO

- **Category** Suboptimal
- **Source** Circuito64.go

Description This function looks too simple.

Recommendation Consider removing it.

```
15 func NewCircuit64() *Circuit64 {
```

CVF-64 INFO

- **Category** Procedural
- **Source** SHA2-512.go

Recommendation This commented out import should be removed.

```
9 //csha3 "golang.org/x/crypto/sha2"
```

CVF-65 INFO

- **Category** Readability
- **Source** SHA2-512.go

Recommendation This could be simplified as: n8 := (n + 127) &^ 127

```
15 n8 := (n + 127) / 128 * 128
```

CVF-66 INFO

- **Category** Documentation
- **Source** Interface.go (ONLY YELLOW)

Description Both comments are confusing and hardly clarify anything.

Recommendation Consider elaborating more.

```
13 const MLAR = 115 /// d(nbConstrains)/d(MLAR) aprox 5.000
   const HSIZE = 2  /// 32 bytes hash as little endian integers
```

CVF-67 INFO

- **Category** Bad naming
- **Source** Interface.go (ONLY YELLOW)

Description The interface name is too generic.

Recommendation Consider using a more specific name.

```
16 type Interface interface {
```

CVF-68 INFO

- **Category** Procedural
- **Source** Interface.go (ONLY YELLOW)

Recommendation This commented out line should be removed.

```
70 //Msg := circuit.GetMsg()
```

CVF-69 INFO

- **Category** Unclear behavior
- **Source** Interface.go (ONLY YELLOW)

Description This function always returns "nil".

Recommendation Consider returning nothing.

```
89 return nil
```

CVF-70 INFO

- **Category** Flaw
- **Source** Signature.go

Description There are no length checks for the array arguments.

Recommendation Consider adding appropriate checks.

```
28 func frontendVariableToU8Hint(_ *big.Int, inputs []*big.Int, result
   ↪ []*big.Int) error {
```

CVF-71 INFO

- **Category** Procedural
- **Source** Signature.go

Recommendation These commented out lines should be removed.

```
63 /*Rc = [32]uints.U8(output[0:32])
   Sc = [32]uints.U8(output[32:64])
   Ac = [32]uints.U8(output[64:96])
   Msg = [MLAR]uints.U8(output[96 : 96+MLAR])*/
```

```
96 /*Rc = sig.Rc
   Sc = sig.Sc
   Ac = sig.Ac
   Msg = sig.Msg*/
```

CVF-72 INFO

- **Category** Procedural
- **Source** curve_utils.go

Recommendation This import should be grouped with other ”gnark” imports.

```
10 "github.com/consensys/gnark/frontend"
```

CVF-73 INFO

- **Category** Procedural
- **Source** curve_utils.go

Recommendation These commented out lines should be removed.

```
14 //fmt.Println("Iniciando...")
```

```
32 /*BX.SetString
   ("151122213495354007725011514095885315114540126930418572060461132839498477622
   02", 10)
   //BX.Mul(BU, big.NewInt(0).ModInverse(BV, Q))
   BY.Mul(big.NewInt(1).Add(big.NewInt(-1), BU),
           big.NewInt(0).ModInverse(big.NewInt(0).Add(big.NewInt(1), BU
              ↪ ), Q))
```

```
37 BX.Mod(BX, Q)
   BY.Mod(BY, Q)*/
```

CVF-74 INFO

- **Category** Suboptimal
- **Source** curve_utils.go

Description Calculating Q instead of hardcoding it as a constant looks weird.

Recommendation Consider just defining a constant for Q.

```
16 Q.Exp(big.NewInt(2), big.NewInt(255), nil)
   Q.Sub(Q, big.NewInt(19))
```

CVF-75 INFO

- **Category** Suboptimal
- **Source** curve_utils.go

Description Calculating A instead of hardcoding it as a constant looks weird.

Recommendation Consider just defining a constant for A.

```
19 A.Sub(Q, big.NewInt(1))
```

CVF-76 INFO

- **Category** Suboptimal
- **Source** curve_utils.go

Description Calculating Ord instead of hardcoding it as a constant looks weird.

Recommendation Consider just defining a constant for Ord.

```
22 Ord.Exp(big.NewInt(2), big.NewInt(252), nil)
   Ord.Add(Ord, temp)
```

CVF-77 INFO

- **Category** Procedural
- **Source** curve_utils.go

Recommendation The multiplications don't need to be calculated for the first loop iteration.

```
79 a = api.Add(api.Mul(a, frontend.Variable(256)), frontend.Variable(
      ↪ input[i].Val))
80 b = api.Add(api.Mul(b, frontend.Variable(256)), frontend.Variable(
      ↪ input[i+16].Val))
```

CVF-78 INFO

- **Category** Procedural
- **Source** curve_ed25519.go

Recommendation These commented out imports should be removed.

```
4  //      "fmt"

6  //      "github.com/consensys/gnark-crypto/ecc"
   //      "github.com/consensys/gnark-crypto/ecc/bn254"

9  //"github.com/consensys/gnark/backend"
10 //"github.com/consensys/gnark/frontend"
   //      "github.com/consensys/gnark/test"
   //"github.com/rs/zerolog"

14 //"github.com/consensys/gnark/std/algebra/fields_bls12377"

16 //      crand "crypto/rand"
   //      "testing"

19 //      fr "github.com/consensys/gnark-crypto/ecc/bn254/fr"
20 //      tbn254 "github.com/consensys/gnark-crypto/ecc/bn254/
      ↪ twistededwards"
   //      td "github.com/consensys/gnark/std/algebra/native/
      ↪ twistededwards"
   //      sha3 "golang.org/x/crypto/sha3"
```

CVF-79 INFO

- **Category** Bad naming
- **Source** curve_ed25519.go

Recommendation The name is confusing, as this function actually uncompresses a compressed point.

```
72 func CompressToPoint(b []byte) Point {
```

CVF-80 INFO

- **Category** Suboptimal
- **Source** curve_ed25519.go

Recommendation This operation is redundant as there will be modular reductions later on.

```
85 num = big.NewInt(0).Mod(num, Q)
```

CVF-81 INFO

- **Category** Suboptimal
- **Source** curve_ed25519.go

Recommendation Bitwise AND here is redundant, as right shift will anyway drop lower bits.

```
98 if res.X.Bit(0) != uint(b[31]&0x80)>>7 {
```

CVF-82 INFO

- **Category** Suboptimal
- **Source** curve_ed25519.go

Recommendation It would be more efficient to use ”Rsh” instead of ”Div” here.

```
150 for ; S.Cmp(big.NewInt(0)) > 0; S.Div(S, big.NewInt(2)) {
```

CVF-83 INFO

- **Category** Procedural
- **Source** curve_ed25519.go

Recommendation These commented lines should be removed.

```
160 //fmt.Println("On Curve")
```

```
168 //fmt.Println(ladoIzq)
    //fmt.Println(ladoDer)
```

CVF-84 INFO

- **Category** Procedural
- **Source** element_f.go

Recommendation These commented out imports should be removed.

```
5  //"github.com/consensys/gnark/backend"
   //"github.com/consensys/gnark/frontend"

8  //"github.com/rs/zerolog"

10 //"github.com/consensys/gnark/std/algebra/fields_bls12377"

12 //"github.com/consensys/gnark-crypto/ecc/bls12-377/fptower"
```

CVF-85 INFO

- **Category** Documentation
- **Source** element_f.go

Recommendation Consider documenting what this number (2^128) means

```
25 FieldBase, _ = big.NewInt(0).SetString("
      ↪ 340282366920938463463374607431768211456", 10)
```

CVF-86 INFO

- **Category** Suboptimal
- **Source** element_f.go

Description Here an argument is used as a local variable, which is a bad practice that makes code harder to read.

Recommendation Consider using a separate local variable instead.

```
39 a = big.NewInt(0).Mod(a, FieldModulus)
```

CVF-87 INFO

- **Category** Suboptimal
- **Source** element_f.go

Recommendation It would be more efficient to use the ”DivMod” function here: https://pkg.go.dev/math/big#Int.DivMod

```
40 return ElementF{[2]frontend.Variable{frontend.Variable(big.NewInt(0)
      ↪ .Mod(a, FieldBase)), frontend.Variable(big.NewInt(0).Div(a,
      ↪ FieldBase))}}
```

```
52 Co := big.NewInt(0).Div(C, FieldModulus)
   R := big.NewInt(0).Mod(C, FieldModulus)
   result[0] = big.NewInt(0).Mod(R, FieldBase)
   result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
86 C := big.NewInt(0).Div(Res, FieldModulus)
   R := big.NewInt(0).Mod(Res, FieldModulus)
   result[0] = big.NewInt(0).Mod(R, FieldBase)
   result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
107 Co := big.NewInt(0).Div(C, FieldModulus)
    R := big.NewInt(0).Mod(C, FieldModulus)
    result[0] = big.NewInt(0).Mod(R, FieldBase)
110 result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
142         result[i].Mod(x0, big.NewInt(2))
            x0.Div(x0, big.NewInt(2))
```

```
146         result[i].Mod(x1, big.NewInt(2))
            x1.Div(x1, big.NewInt(2))
```

```
176         result[31-i].Mod(x, big.NewInt(256))
            x.Div(x, big.NewInt(256))
```

CVF-88 INFO

- **Category** Flaw
- **Source** element_f.go

Description Error is ignored here.

Recommendation Consider handling it

```
44 b, _ := big.NewInt(0).SetString(a, 10)
```

CVF-89 INFO

- **Category** Suboptimal
- **Source** element_f.go

Recommendation It would be more efficient to use ”Bit” function here: https://pkg.go.dev/math/big#Int.Bit

```
142 result[i].Mod(x0, big.NewInt(2))
    x0.Div(x0, big.NewInt(2))
```

```
146 result[i].Mod(x1, big.NewInt(2))
    x1.Div(x1, big.NewInt(2))
```

CVF-90 INFO

- **Category** Suboptimal
- **Source** element_f.go

Recommendation It would be more efficient to use the ”Rsh” function here: https://pkg.go.dev/math/big#Int.Rsh

```
143 x0.Div(x0, big.NewInt(2))
```

```
147 x1.Div(x1, big.NewInt(2))
```

```
177 x.Div(x, big.NewInt(256))
```

CVF-91 INFO

- **Category** Suboptimal
- **Source** element_f.go

Description This function always returns ”nil’.

Recommendation Consider returning nothing.

```
57  return nil
```

```
112 return nil
```

```
150 return nil
```

```
179 return nil
```

CVF-92 INFO

- **Category** Procedural
- **Source** element_f.go

Recommendation These commented out lines should be removed.

```
155 //api.Println("a : ", a.V[0], a.V[1])
```

```
165 //api.Println(res...)
    //api.Println(a.V[0], a.V[1])
```

```
174 //fmt.Println(FieldBase)
```

```
186 //for i := 31; i >= 0; i-- {
```

CVF-93 INFO

- **Category** Suboptimal
- **Source** element_f.go

Recommendation This loop could be simplified using the ”Bytes” function: https://pkg.go.dev/math/big#Int.Bytes

```
175 for i := 0; i < 32; i++ {
        result[31-i].Mod(x, big.NewInt(256))
        x.Div(x, big.NewInt(256))
}
```

CVF-94 INFO

- **Category** Procedural
- **Source** element_o.go

Recommendation These commented out imports should be removed.

```
5  //"github.com/consensys/gnark/backend"
   //"github.com/consensys/gnark/frontend"
```

```
8  //"github.com/rs/zerolog"
```

```
10 //"github.com/consensys/gnark/std/algebra/fields_bls12377"
```

```
12 //"github.com/consensys/gnark-crypto/ecc/bls12-377/fptower"
```

CVF-95 INFO

- **Category** Suboptimal
- **Source** element_o.go

Recommendation It would be more efficient to use the "DivMod" function.

```
36 return ElementO{[2]frontend.Variable{frontend.Variable(big.NewInt(0)
   ↪ .Mod(a, FieldBase)), frontend.Variable(big.NewInt(0).Div(a,
   ↪ FieldBase))}}
```

```
48 Co := big.NewInt(0).Div(C, Ord)
   R := big.NewInt(0).Mod(C, Ord)
50 result[0] = big.NewInt(0).Mod(R, FieldBase)
   result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
82 C := big.NewInt(0).Div(Res, Ord)
   R := big.NewInt(0).Mod(Res, Ord)
   result[0] = big.NewInt(0).Mod(R, FieldBase)
   result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
103 Co := big.NewInt(0).Div(C, Ord)
    R := big.NewInt(0).Mod(C, Ord)
    result[0] = big.NewInt(0).Mod(R, FieldBase)
    result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
138         result[i].Mod(x0, big.NewInt(2))
            x0.Div(x0, big.NewInt(2))
```

```
142         result[i].Mod(x1, big.NewInt(2))
            x1.Div(x1, big.NewInt(2))
```

```
171         result[31-i].Mod(x, big.NewInt(256))
            x.Div(x, big.NewInt(256))
```

CVF-96 INFO

- **Category** Suboptimal
- **Source** element_o.go

Description This function always returns ”nil’.

Recommendation Consider returning nothing.

```
53 return nil
```

```
87 return nil
```

```
108 return nil
```

```
146 return nil
```

```
174 return nil
```

CVF-97 INFO

- **Category** Suboptimal
- **Source** element_o.go

Recommendation It would be more efficient to use ”Bit” function here: https://pkg.go.dev/math/big#Int.Bit

```
138 result[i].Mod(x0, big.NewInt(2))
    x0.Div(x0, big.NewInt(2))
```

```
142 result[i].Mod(x1, big.NewInt(2))
    x1.Div(x1, big.NewInt(2))
```

CVF-98 INFO

- **Category** Suboptimal
- **Source** element_o.go

Recommendation It would be more efficient to use the ”Rsh” function here: https://pkg.go.dev/math/big#Int.Rsh

```
139 x0.Div(x0, big.NewInt(2))
```

```
143 x1.Div(x1, big.NewInt(2))
```

```
172 x.Div(x, big.NewInt(256))
```

CVF-99 INFO

- **Category** Procedural
- **Source** element_o.go

Recommendation There commented out lines should be removed.

```
151 //api.Println("a : ", a.V[0], a.V[1])
```

```
157         //base = api.Mul(base, frontend.Variable(2))
```

```
160 //api.Println(res...)
    //api.Println(a.V[0], a.V[1])
```

```
169 //fmt.Println(FieldBase)
```

```
181 //for i := 31; i >= 0; i-- {
```

CVF-100 INFO

- **Category** Suboptimal
- **Source** element_o.go

Recommendation This loop could be simplified using the ”Bytes” function: https://pkg.go.dev/math/big#Int.Bytes

```
170 for i := 0; i < 32; i++ {
        result[31-i].Mod(x, big.NewInt(256))
        x.Div(x, big.NewInt(256))
    }
```

CVF-101 INFO

- **Category** Procedural
- **Source** element_q.go

Recommendation These commented out imports should be removed.

```
5  //"github.com/consensys/gnark/backend"
   //"github.com/consensys/gnark/frontend"
```

```
8  //"github.com/rs/zerolog"
```

```
10 //"github.com/consensys/gnark/std/algebra/fields_bls12377"
```

```
12 //"github.com/consensys/gnark-crypto/ecc/bls12-377/fptower"
```

CVF-102 INFO

- **Category** Suboptimal
- **Source** element_q.go

Recommendation It would be more efficient to use the "DivMod" function.

```
36 return ElementQ{[2]frontend.Variable{frontend.Variable(big.NewInt(0)
   ↪ .Mod(a, FieldBase)), frontend.Variable(big.NewInt(0).Div(a,
   ↪ FieldBase))}}
```

```
48 Co := big.NewInt(0).Div(C, Q)
   R := big.NewInt(0).Mod(C, Q)
50 result[0] = big.NewInt(0).Mod(R, FieldBase)
   result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
82 result[0] = big.NewInt(0).Mod(d, FieldBase)
   result[1] = big.NewInt(0).Div(d, FieldBase)
   result[2] = big.NewInt(0).Mod(x, FieldBase)
   result[3] = big.NewInt(0).Div(x, FieldBase)
```

```
115 C := big.NewInt(0).Div(Res, Q)
    R := big.NewInt(0).Mod(Res, Q)
    result[0] = big.NewInt(0).Mod(R, FieldBase)
    result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
136 Co := big.NewInt(0).Div(C, Q)
    R := big.NewInt(0).Mod(C, Q)
    result[0] = big.NewInt(0).Mod(R, FieldBase)
    result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
161 Co := big.NewInt(0).Div(C, Q)
    R := big.NewInt(0).Mod(C, Q)
    result[0] = big.NewInt(0).Mod(R, FieldBase)
    result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
198         result[i].Mod(x0, big.NewInt(2))
            x0.Div(x0, big.NewInt(2))
```

```
202         result[i].Mod(x1, big.NewInt(2))
            x1.Div(x1, big.NewInt(2))
```

```
232         result[31-i].Mod(x, big.NewInt(256))
            x.Div(x, big.NewInt(256))
```

CVF-103 INFO

- **Category** Suboptimal
- **Source** element_q.go

Description This function always returns ”nil’.

Recommendation Consider returning nothing.

```
53  return nil
```

```
86  return nil
```

```
120 return nil
```

```
141 return nil
```

```
166 return nil
```

```
206 return nil
```

```
235 return nil
```

CVF-104 INFO

- **Category** Procedural
- **Source** element_q.go

Recommendation There commented out lines should be removed.

```
64  //izq := api.Add(
    //              api.Mul(a.V[0], b.V[0]), api.Mul(FieldBaseC,
    //                      api.Add(api.Mul(a.V[1], b.V[0]), api.Mul(a.V
        ↪ [0], b.V[1])), api.Mul(FieldBaseC, a.V[1], b.V[1])))
```

```
80  /// d * b = a + Q * x
```

```
157 //C = big.NewInt(0).Add(C, Q)
    /*fmt.Println(A)
    fmt.Println(B)
160 fmt.Println(C)*/
```

```
211 //api.Println("a : ", a.V[0], a.V[1])
```

```
221 //api.Println(res...)
    //api.Println(a.V[0], a.V[1])
```

```
230 //fmt.Println(FieldBase)
```

```
242 //for i := 31; i >= 0; i-- {
```

CVF-105 INFO

- **Category** Suboptimal
- **Source** element_q.go

Description This code largely duplicates the addition method.

Recommendation Consider implementing the negation function instead.

```
169 func SubElementQ(a, b ElementQ, api frontend.API) ElementQ {
```

CVF-106 INFO

- **Category** Suboptimal
- **Source** element_q.go

Recommendation It would be more efficient to use ”Bit” function here: https://pkg.go.dev/math/big#Int.Bit

198
```
result[i].Mod(x0, big.NewInt(2))
x0.Div(x0, big.NewInt(2))
```

202
```
result[i].Mod(x1, big.NewInt(2))
x1.Div(x1, big.NewInt(2))
```

CVF-107 INFO

- **Category** Suboptimal
- **Source** element_q.go

Recommendation It would be more efficient to use the ”Rsh” function here: https://pkg.go.dev/math/big#Int.Rsh

199
```
x0.Div(x0, big.NewInt(2))
```

203
```
x1.Div(x1, big.NewInt(2))
```

233
```
x.Div(x, big.NewInt(256))
```

CVF-108 INFO

- **Category** Suboptimal
- **Source** element_q.go

Recommendation This loop could be simplified using the ”Bytes” function: https://pkg.go.dev/math/big#Int.Bytes

231
```
for i := 0; i < 32; i++ {
        result[31-i].Mod(x, big.NewInt(256))
        x.Div(x, big.NewInt(256))
}
```

CVF-109 INFO

- **Category** Documentation
- **Source** element_q.go

Description It is not clear if the ByteValueOf function guarantees the range of the output.

Recommendation Consider documenting.

```
244 res[i] = uapi.ByteValueOf(temp[i])
```

CVF-110 INFO

- **Category** Procedural
- **Source** element.go

Recommendation These commented out imports should be removed.

```
5  //"github.com/consensys/gnark/backend"
   //"github.com/consensys/gnark/frontend"
```

```
8  //"github.com/rs/zerolog"
```

```
10 //"github.com/consensys/gnark/std/algebra/fields_bls12377"
```

```
12 //"github.com/consensys/gnark-crypto/ecc/bls12-377/fptower"
```

CVF-111 INFO

- **Category** Documentation
- **Source** element.go

Description Semantics of this data structure is unclear.

Recommendation Consider documenting

```
29 type Element struct {
```

CVF-112 INFO

- **Category** Suboptimal
- **Source** element.go

Recommendation It would be more efficient to use the "DivMod" function.

```
36 return Element{[2]frontend.Variable{frontend.Variable(big.NewInt(0).
      ↪ Mod(a, FieldBase)), frontend.Variable(big.NewInt(0).Div(a,
      ↪ FieldBase))},
           [2]frontend.Variable{frontend.Variable(big.NewInt(0).Mod(mod
              ↪ , FieldBase)), frontend.Variable(big.NewInt(0).Div(mod
              ↪ , FieldBase))},
```

```
52 Co := big.NewInt(0).Div(C, Mod)
   R := big.NewInt(0).Mod(C, Mod)
   result[0] = big.NewInt(0).Mod(R, FieldBase)
   result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
90 C := big.NewInt(0).Div(Res, Mod)
   R := big.NewInt(0).Mod(Res, Mod)
   result[0] = big.NewInt(0).Mod(R, FieldBase)
   result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
112 Co := big.NewInt(0).Div(C, Mod)
    R := big.NewInt(0).Mod(C, Mod)
    result[0] = big.NewInt(0).Mod(R, FieldBase)
    result[1] = big.NewInt(0).Div(R, FieldBase)
```

```
151         result[i].Mod(x0, big.NewInt(2))
            x0.Div(x0, big.NewInt(2))
```

```
155         result[i].Mod(x1, big.NewInt(2))
            x1.Div(x1, big.NewInt(2))
```

```
185         result[31-i].Mod(x, big.NewInt(256))
            x.Div(x, big.NewInt(256))
```

CVF-113 INFO

- **Category** Suboptimal
- **Source** element.go

Description This function always returns ”nil’.

Recommendation Consider returning nothing.

```
57  return nil
```

```
95  return nil
```

```
117 return nil
```

```
159 return nil
```

```
188 return nil
```

CVF-114 INFO

- **Category** Suboptimal
- **Source** element.go

Recommendation It would be more efficient to use ”Bit” function here: https://pkg.go.dev/math/big#Int.Bit

```
151 result[i].Mod(x0, big.NewInt(2))
    x0.Div(x0, big.NewInt(2))
```

```
155 result[i].Mod(x1, big.NewInt(2))
    x1.Div(x1, big.NewInt(2))
```

CVF-115 INFO

- **Category** Suboptimal
- **Source** element.go

Recommendation It would be more efficient to use the ”Rsh” function here: https://pkg.go.dev/math/big#Int.Rsh

152
```
x0.Div(x0, big.NewInt(2))
```

156
```
x1.Div(x1, big.NewInt(2))
```

186
```
x.Div(x, big.NewInt(256))
```

CVF-116 INFO

- **Category** Procedural
- **Source** element.go

Recommendation There commented out lines should be removed.

164
```
//api.Println("a : ", a.V[0], a.V[1])
```

174
```
//api.Println(res...)
//api.Println(a.V[0], a.V[1])
```

183
```
//fmt.Println(FieldBase)
```

195
```
//for i := 31; i >= 0; i-- {
```

CVF-117 INFO

- **Category** Suboptimal
- **Source** element.go

Recommendation This loop could be simplified using the ”Bytes” function: https://pkg.go.dev/math/big#Int.Bytes

184
```
for i := 0; i < 32; i++ {
        result[31-i].Mod(x, big.NewInt(256))
        x.Div(x, big.NewInt(256))
}
```

CVF-118 INFO

- **Category** Documentation
- **Source** element.go

Description It is not clear if the ByteValueOf function guarantees the range of the output.

Recommendation Consider documenting.

```
197 res[i] = uapi.ByteValueOf(temp[i])
```

CVF-119 INFO

- **Category** Bad datatype
- **Source** curve_ed25519circuit.go

Recommendation The value ”253” should be a named constant.

```
33 func GetBaseCircuitPows() [253]PointCircuit {
       var res [253]PointCircuit
36     for i := 1; i < 253; i++ {
62     for i := 0; i < 253; i++ {
75 func MulByScalarCircuitWithPows(p PointCircuit, s ElementO, pows
   ↪ [253]PointCircuit, api frontend.API) PointCircuit {
79     for i := 0; i < 253; i++ {
```

CVF-120 INFO

- **Category** Suboptimal
- **Source** curve_ed25519circuit.go

Description Here each point is calculated separately, which is suboptimal.

Recommendation Consider calculating each point by doubling the previous one.

```
37 res[i] = PointToCircuit(IntToPoint(big.NewInt(0).Exp(big.NewInt(2),
   ↪ big.NewInt(int64(i)), nil)))
```

CVF-121 INFO

- **Category** Suboptimal
- **Source** curve_ed25519circuit.go

Recommendation It would be more efficient to replace ”Exp” with ”Lsh” here.

```
37 res[i] = PointToCircuit(IntToPoint(big.NewInt(0).Exp(big.NewInt(2),
   ↪ big.NewInt(int64(i)), nil)))
```

CVF-122 INFO

- **Category** Procedural
- **Source** curve_ed25519circuit.go

Recommendation Consider referencing the curve addition algorithm here for consistency

```
43 func AddCircuit(p1, p2 PointCircuit, api frontend.API) PointCircuit
   ↪ {
```

CVF-123 INFO

- **Category** Procedural
- **Source** curve_ed25519circuit.go

Recommendation This commented out code should be removed.

```
49                  //SubElementQ(ProdElementQ(AddElementQ(p1.X, p1.Y,
                       ↪ api), AddElementQ(p2.X, p2.Y, api), api),
                       ↪ AddElementQ(XX, YY, api), api),
```

```
59          //exp := BitsElement(AddElement(StringToElement
               ↪ ("57896044618658097711785492504343953926634992332820282
60  019728792003956564819948",␣OrdC),␣s,␣api),␣api)
␣␣␣␣␣␣␣
```

```
76          //exp := BitsElement(AddElement(StringToElement
               ↪ ("57896044618658097711785492504343953926634992332820282
    019728792003956564819948",␣OrdC),␣s,␣api),␣api)
␣␣␣␣␣␣␣
```

```
103         //      api.Println("RES 0 : ", res.V[0], " ", res.V[1])
```

```
107                 //      api.Println("RES ", res.V[0], " ", res.V[1])
                    //res = api.Mul(res, frontend.Variable(256))
                    //res = api.Add(res, hash[i].Val)
```

```
117         //      api.Println("RES 0 : ", res.V[0], " ", res.V[1])
```

```
121                 //      api.Println("RES ", res.V[0], " ", res.V[1])
                    //res = api.Mul(res, frontend.Variable(256))
                    //res = api.Add(res, hash[i].Val)
```

```
131         //      api.Println("RES 0 : ", res.V[0], " ", res.V[1])
```

```
135                 //      api.Println("RES ", res.V[0], " ", res.V[1])
                    //res = api.Mul(res, frontend.Variable(256))
                    //res = api.Add(res, hash[i].Val)
```

```
208 //ssh -i pub_rsa lautaro@34.118.49.208
```

CVF-124 INFO

- **Category** Suboptimal
- **Source** curve_ed25519circuit.go

Recommendation This line shouldn't be executed for the first loop iteration.

```
105 res = ProdElementQ(res, StringToElementQ("256"), api)
```

CVF-125 INFO

- **Category** Suboptimal
- **Source** curve_ed25519circuit.go

Recommendation This line shouldn't be executed for the first loop iteration.

```
119 res = ProdElementO(res, StringToElementO("256"), api)
```

CVF-126 INFO

- **Category** Suboptimal
- **Source** curve_ed25519circuit.go

Recommendation This line shouldn't be executed for the first loop iteration.

```
133 res = ProdElement(res, StringToElement("256", mod), api)
```

CVF-127 INFO

- **Category** Suboptimal
- **Source** curve_ed25519circuit.go

Recommendation This line shouldn't be executed for the first loop iteration.

```
152 Y.Mul(Y, big.NewInt(256))
```

CVF-128 INFO

- **Category** Suboptimal
- **Source** curve_ed25519circuit.go

Recommendation These lines shouldn't be executed for the first loop iteration.

198
```
y0 = api.Mul(y0, "256")
y1 = api.Mul(y1, "256")
```

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