Audit of BLS-12 381

Coinbase

10 September 2021
Version: 0.2

Presented by:
Kudelski Security Research Team
Kudelski Security - Nagravision SA

Corporate Headquarters
Route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland

Final

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY **5**
1.1 Engagement Scope 5
1.2 Engagement Analysis 5
1.3 Issue Summary List 6

2 METHODOLOGY **9**
2.1 Kickoff 9
2.2 Ramp-up 9
2.3 Review 10
2.4 Reporting 11
2.5 Verify 12
2.6 Additional Note 12

3 TECHNICAL DETAILS OF SECURITY FINDINGS **13**
3.1 KS-SBCF-F-01: Field Exponentiation functions leak the exponent 13
3.2 KS-SBCF-F-02: Scalar multiplication in G1 and G2 leak the scalar value . . 14
3.3 KS-SBCF-F-03: Non-constant time Field inversion 14
3.4 KS-SBCF-F-04: Null (nil) pointer dereference in G1 and G2 IsZero function 15
3.5 KS-SBCF-F-05: Avoid OOM in certain functions where the input length is not validated 17
3.6 KS-SBCF-F-06: G1 and G2 arithmetic fail to catch negative big Int values in scalar input parameters 18
3.7 KS-SBCF-F-07: Prime order subgroup membership not enforced 19
3.8 KS-SBCF-F-08: Coefficient array size in millerLoop can cause OOB 20
3.9 KS-SBCF-F-09: Error handling negative number in field conversion 21
3.10 KS-SBCF-F-10: Non-constant time comparisons 22
3.11 KS-SBCF-F-11: Non-constant time fallback implementation 23

3.12 KS-SBCF-F-12: wNAF scalar multiplication is not constant time 23
3.13 KS-SBCF-F-13: Serialize functions do not strictly follow zcash serialization checks for invalid input 24
3.14 KS-SBCF-F-14: No error check in integer conversion function 24
3.15 KS-SBCF-F-15: Missing mod reduction in doublingStep and additionStep . 25

4 OTHER OBSERVATIONS 26

4.1 KS-SBCF-O-01: Faster subgroup check can be implemented 26
4.2 KS-SBCF-O-02: Confusion in error messages 26
4.3 KS-SBCF-O-03: Dead link to serialization rules for uncompressed points . 27
4.4 KS-SBCF-O-04: Redundant code in Add functions 27
4.5 KS-SBCF-O-05: Error in comment in Affine function 27
4.6 KS-SBCF-O-06: Dead link to Jacobian coordinates addition formula in Add 28
4.7 KS-SBCF-O-07: Speed-up: Implement point addition formulas for special cases 28
4.8 KS-SBCF-O-08: Dead link to Jacobian coordinates addition formulae . . . 29
4.9 KS-SBCF-O-09: Speed-up: Implement doubling formula for special case Z = 1 29
4.10 KS-SBCF-O-10: Set call in Double is not needed 29
4.11 KS-SBCF-O-11: Different algorithms are used to clear the cofactor 30
4.12 KS-SBCF-O-12: Many Field tests does not initialize value randomly 30
4.13 KS-SBCF-O-13: Lack of Field tests with hardcoded values 31
4.14 KS-SBCF-O-14: Speed-up: Faster cofactor clearing 32
4.15 KS-SBCF-O-15: FromBytes in g1.go and g2.go allows inputs bigger than 96 and 192 bytes 32
4.16 KS-SBCF-O-16: Reference to Formula 3 in doublingStep function does not exist in paper 33
4.17 KS-SBCF-O-17: Unused ladd, ldouble function. 33
4.18 KS-SBCF-O-18: Confusing parameter names. 33

4.19 KS-SBCF-O-19: Test vectors from Pairing-Friendly Curves ietf draft (July 2021) produce a different result 34
4.20 KS-SBCF-O-20: Unused parameters in bls12-381 34
4.21 KS-SBCF-O-21: Unused arithmetic functions in the extensions 35
4.22 KS-SBCF-O-22: Redundant return statement in fp.go 35

5 APPENDIX A: ABOUT KUDELSKI SECURITY 37

6 APPENDIX B: DOCUMENT HISTORY 38

7 APPENDIX C: SEVERITY RATING DEFINITIONS 39

REFERENCES 40

1 EXECUTIVE SUMMARY

Kudelski Security ("Kudelski," "we"), the cybersecurity division of the Kudelski Group, was engaged by Coinbase ("the Client") to conduct an external security assessment in the form of a code audit of the BLS12-381 cryptographic library ("the Product"). The assessment was conducted remotely by the Kudelski Security Research Team. The audit took place from July 30, 2021 to August 13, 2021. The audit focused on the following objectives:

- To provide a professional opinion on the maturity, adequacy, and efficiency of the software solution in exam.
- To check compliance with existing standards.
- To identify potential security or interoperability issues and include improvement recommendations based on the result of our analysis.

This report summarizes the analysis performed and findings. It also contains detailed descriptions of the discovered vulnerabilities and recommendations for remediation.

1.1 Engagement Scope

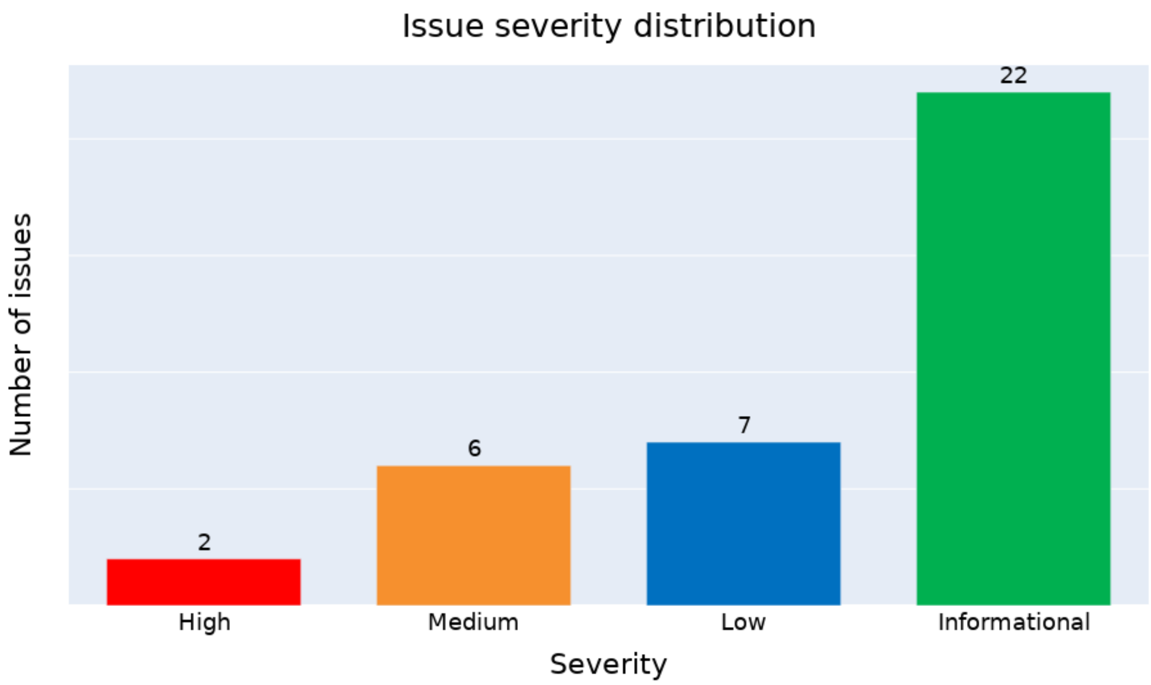
The scope of the audit was a code audit of the Product written in Go, with a particular attention to safe implementation of hashing, randomness generation, protocol verification, and potential for misuse and leakage of secrets.

The target of the audit was the cryptographic code provided by the Client within the archive `bls12-381-main-2021-05-21.zip` with SHA256 checksum: `0xb165a4fb7489662c972afe34ea463dbbd8a0103b4ee273e9f7a7b3676c8b97f9`. Particular attention was given to the correct implementation of pairing and field operations and as well constant timeness. Secure erasure of secret data from memory were not considered in scope.

1.2 Engagement Analysis

The engagement consisted of a ramp-up phase where the necessary documentation about the technological standards and design of the solution in exam was acquired, followed by a manual inspection of the code provided by Coinbase and the drafting of this report.

As a result of our work, we have identified **2 High**, **6 Medium**, **7 Low** and **22 Informational** findings.



1.3 Issue Summary List

The following security issues were found:

ID	Severity	Finding	Status
KS-SBCF-F-01	**High**	Field Exponentiation functions leak the exponent	**Remediated**
KS-SBCF-F-02	**High**	Scalar multiplication in G1 and G2 leak the scalar value	**Remediated**
KS-SBCF-F-03	**Medium**	Non-constant time Field inversion	**Remediated**
KS-SBCF-F-04	**Medium**	Null (nil) pointer dereference in G1 and G2 IsZero function	**Open**

ID	Severity	Finding	Status
KS-SBCF-F-05	Medium	Avoid OOM in certain functions where the input length is not validated	Remediated
KS-SBCF-F-06	Medium	G1 and G2 arithmetic fail to catch negative big Int values in scalar input parameters	Remediated
KS-SBCF-F-07	Medium	Prime order subgroup membership not enforced	Open
KS-SBCF-F-08	Medium	Coefficient array size in millerLoop can cause OOB	Open
KS-SBCF-F-09	Low	Error handling negative number in field conversion	Open
KS-SBCF-F-10	Low	Non-constant time comparisons	Remediated
KS-SBCF-F-11	Low	Non-constant time fallback implementation	Open
KS-SBCF-F-12	Low	wNAF scalar multiplication is not constant time	Open
KS-SBCF-F-13	Low	Serialize functions do not strictly follow zcash serialization checks for invalid input	Open
KS-SBCF-F-14	Low	No error check in integer conversion function	Open
KS-SBCF-F-15	Low	Missing mod reduction in doublingStep and additionStep	Open

The following are observations related to general design and improvements:

ID	Severity	Finding	Status
KS-SBCF-O-01	Informational	Faster subgroup check can be implemented	Informational
KS-SBCF-O-02	Informational	Confusion in error messages	Informational
KS-SBCF-O-03	Informational	Dead link to serialization rules for uncompressed points	Informational
KS-SBCF-O-04	Informational	Redundant code in Add functions	Informational
KS-SBCF-O-05	Informational	Error in comment in Affine function	Informational

ID	Severity	Finding	Status
KS-SBCF-O-06	**Informational**	Dead link to Jacobian coordinates addition formula in Add	**Informational**
KS-SBCF-O-07	**Informational**	Speed-up: Implement point addition formulas for special cases	**Informational**
KS-SBCF-O-08	**Informational**	Dead link to Jacobian coordinates addition formulae	**Informational**
KS-SBCF-O-09	**Informational**	Speed-up: Implement doubling formula for special case Z = 1	**Informational**
KS-SBCF-O-10	**Informational**	Set call in Double is not needed	**Informational**
KS-SBCF-O-11	**Informational**	Different algorithms are used to clear the cofactor	**Informational**
KS-SBCF-O-12	**Informational**	Many Field tests does not initialize value randomly	**Informational**
KS-SBCF-O-13	**Informational**	Lack of Field tests with hardcoded values	**Informational**
KS-SBCF-O-14	**Informational**	Speed-up: Faster cofactor clearing	**Informational**
KS-SBCF-O-15	**Informational**	FromBytes in g1.go and g2.go allows inputs bigger than 96 and 192 bytes	**Informational**
KS-SBCF-O-16	**Informational**	Reference to Formula 3 in doublingStep function does not exist in paper	**Informational**
KS-SBCF-O-17	**Informational**	Unused ladd, ldouble function.	**Informational**
KS-SBCF-O-18	**Informational**	Confusing parameter names.	**Informational**
KS-SBCF-O-19	**Informational**	Test vectors from Pairing-Friendly Curves ietf draft (July 2021) produce a different result	**Informational**
KS-SBCF-O-20	**Informational**	Unused parameters in bls12-381	**Informational**
KS-SBCF-O-21	**Informational**	Unused arithmetic functions in the extensions	**Informational**
KS-SBCF-O-22	**Informational**	Redundant return statement in fp.go	**Informational**

2 METHODOLOGY

For this engagement, Kudelski used a methodology that is described at high-level in this section. This is broken up into the following phases.

2.1 Kickoff

The project was kicked off when all of the sales activities had been concluded. We set up a kickoff meeting where project stakeholders were gathered to discuss the project as well as the responsibilities of participants. During this meeting we verified the scope of the engagement and discussed the project activities. It was an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there was an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

2.2 Ramp-up

Ramp-up consisted of the activities necessary to gain proficiency on the particular project. This included the steps needed for gaining familiarity with the codebase and technological innovations utilized, such as:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for the languages used in the code
- Researching common flaws and recent technological advancements.

2.3 Review

The review phase is where a majority of the work on the engagement was performed. In this phase we analyzed the project for flaws and issues that could impact the security posture. This included a review of the code, and a specification matching to match the specification of existing standards to the implemented code.

In this code audit, we performed the following tasks:

1. Review of the code written for the project
2. Assessment of the cryptographic primitives used
3. Compliance of the code with the provided technical documentation.

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging.

This is a general and not comprehensive list, meant only to give an understanding of the issues we have been looking for.

Cryptography

We analyzed the cryptographic primitives and components as well as their implementation. We checked in particular:

- Matching of the proper cryptographic primitives to the desired cryptographic functionality needed

- Security level of cryptographic primitives and their respective parameters (key lengths, etc.)
- Safety of the randomness generation in general as well as in the case of failure
- Safety of key management
- Assessment of proper security definitions and compliance to use cases
- Checking for known vulnerabilities in the primitives used.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description.

2.4 Reporting

Kudelski delivered to the Client a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project, which is also the general structure of the current final report.

The executive summary contains an overview of the engagement, including the number of findings as well as a statement about our general risk assessment of the project as a whole.

In the report we not only point out security issues identified but also informational findings for improvement categorized into several buckets:

- High
- Medium
- Low
- Informational.

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we performed the audit, we also identified issues that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

2.5 Verify

After the preliminary findings have been delivered, we verified the fixes applied by the Client. After these fixes were verified, we updated the status of the finding in the report.

The output of this phase was the current, final report with any mitigated findings noted.

2.6 Additional Note

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, along with the scope of the agreement.

In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit. Since this is a library, we ranked some of these vulnerabilities potentially higher than usual, as we expect the code to be reused across different applications with different input sanitization and parameters.

Correct memory management is left to GoLang and was therefore not in scope. Zeroization of secret values from memory is also not enforceable at a low level in a language such as GoLang.

While assessment the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination. Information about the severity ratings can be found in **Appendix C** of this document.

3 TECHNICAL DETAILS OF SECURITY FINDINGS

This section contains the technical details of our findings as well as recommendations for mitigation.

3.1 KS-SBCF-F-01: Field Exponentiation functions leak the exponent

Severity: High

Status: Remediated

Location: fp.go:142, fp2.go:193, fp6.go:270, fp12.go:215, fp12.go:226.

Description

The field exponentiation function `exp` leaks the exponent since the timing of the function depends on the exponent. In fp.go there is:

```go
func exp(c, a *fe, e *big.Int) {
    z := new(fe).set(r1)
    for i := e.BitLen(); i >= 0; i-- {
        mul(z, z, z)
        if e.Bit(i) == 1 {
            mul(z, z, a)
        }
    }
    c.set(z)
}
```

The exponent is scanned and checked bit by bit. If it is '1,' then an additional multiplication is performed. Moreover, the loop iterates according to the bit length of the exponent leaking also the length of the exponent in time. The same observation applies for the function `cyclotomicExp` in fp12.go as well.

Recommendation

Use a constant time exponentiation algorithm.

Status details

Coinbase acknwoledged this issue and fixed it.

3.2 KS-SBCF-F-02: Scalar multiplication in G1 and G2 leak the scalar value

Severity: High

Status: Remediated

Location: g1.go:424 and g2.go:433

Description

The function `MulScalar` uses double-and-add algorithm that performs an addition when the exponent bit is 1. Moreover, it also leaks the length of the exponent.

Recommendation

Use Montgomery ladder as described in section 5 of [10].

```
The pairing-based protocols, such as the BLS signatures, use
a scalar multiplication in G_1, G_2 and an exponentiation in
G_3 with the secret key. In order to prevent the leakage of
secret key due to side channel attacks, implementors should
apply countermeasure techniques such as montgomery ladder
[Montgomery] [CF06] when they implement modules of a scalar
multiplication and an exponentiation. Please refer [Montgomery]
and [CF06] for the detailed algorithms of montgomery ladder.
```

Status details

Coinbase acknowledged this issue and fixed it.

3.3 KS-SBCF-F-03: Non-constant time Field inversion

Severity: Medium

Status: Remediated

Location: fp.go:110

Description

The field inversion function `inverse` uses the binary GCD algorithm, which is not constant time. This algorithm was successfully attack in the past with a single-trace attack [5]. If sensitive values are passed to the `inverse` function, they may be recovered by side-channel analysis.

Recommendation

Use a constant-time inversion algorithm like the one provided in [2]. Another option would be to maintain the same algorithm but to generate a blinding factor w (randomly). Then, the inverse is computed with input $a \cdot w$. At the end of the inversion function, a^{-1} is obtained by multiplying the result by w.

Status details

Coinbase corrected this issue by using Euler theorem exponentiation.

3.4 KS-SBCF-F-04: Null (nil) pointer dereference in G1 and G2 IsZero function

Severity: Medium

Status: Open

Location: g1.go:255, g2.go:265

Description

The `IsZero` implementation of g1.go and g2.go does not check if the input parameter (a pointer) is null. Then, it is possible to make the library panic by importing an invalid point:

```
    g_1_t_v, _ := G1.fromBytesUnchecked(fromHex(48,
        "0xb7f1d3a73197d7942695638c4fa9ac0fc3688c4f9774b905a14e3a3f171...
        "0x08b3f481e3aaa0f1a09e30ed741d8ae4fcf5e095d5d00af600db18cb2c0...
    ))
    //g1_t_v, err := G1.FromBytes(fromHex(48, ))
    if err != nil {
        t.Fatal(err)
```

```
    }
    r := bls.AddPair(g_1_t_v, G2.One()).Result()
    if !r.Equal(expected) {
        t.Fatal("bad pairing")
    }
    if !GT.IsValid(r) {
        t.Fatal("element is not in correct subgroup")
    }
}
```

The result is a nil pointer dereference:

```
=== RUN   TestPairingExpected
--- FAIL: TestPairingExpected (0.00s)
panic: runtime error: invalid memory address or nil pointer dereference [recovered]
    panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x88 pc=0x531d69]

goroutine 6 [running]:
testing.tRunner.func1.2(0x5735c0, 0x699630)
    /usr/lib/go-1.16/src/testing/testing.go:1144 +0x332
testing.tRunner.func1(0xc000001380)
    /usr/lib/go-1.16/src/testing/testing.go:1147 +0x4b6
panic(0x5735c0, 0x699630)
    /usr/lib/go-1.16/src/runtime/panic.go:965 +0x1b9
github.cbhq.net/c3/bls12-381.(*G1).IsZero(...)
    /home/vmr/work/crypto_audits/audits_work/work/git/coinbase-bls2-audit
        /src/bls12-381-main/g1.go:257
github.cbhq.net/c3/bls12-381.(*Engine).isZero(0xc000152000, 0x0,
    0xc00014ed80, 0x40e058)
    /home/vmr/work/crypto_audits/audits_work/work/git/coinbase-bls2-audit
        /src/bls12-381-main/pairing.go:76 +0x29
github.cbhq.net/c3/bls12-381.(*Engine).AddPair(0xc000152000, 0x0, 0xc00014ed80, 0x60)
    /home/vmr/work/crypto_audits/audits_work/work/git/coinbase-bls2-audit
        /src/bls12-381-main/pairing.go:55 +0x45
```

```
github.cbhq.net/c3/bls12-381.TestPairingExpected(0xc000001380)
    /home/vmr/work/crypto_audits/audits_work/work/git/coinbase-bls2-audit
        /src/bls12-381-main/pairing_test.go:39 +0x3b7
testing.tRunner(0xc000001380, 0x5a75f0)
    /usr/lib/go-1.16/src/testing/testing.go:1194 +0xef
created by testing.(*T).Run
    /usr/lib/go-1.16/src/testing/testing.go:1239 +0x2b3
exit status 2
FAIL    github.cbhq.net/c3/bls12-381    0.005s
```

Recommendation

Perform input validation in both importing and `isZero` functions for G1 and G2.

Status details

Coinbase acknowledged this issue and check the `nil` case in the code. The issue is not completely solved but since both methods are private Coinbase is less concerned about them.

3.5 KS-SBCF-F-05: Avoid OOM in certain functions where the input length is not validated

Severity: Medium

Status: Remediated

Location: g1.go:502 and g2.go:551, g1.go:519 and g2.go:569, g2.go:588, pairing.go:13

Description

The length of the input byte array is not validated in the functions depicted below. It is possible to create an out-of-memory (OOM) situation with large input:

```
- EncodeToCurve
- HashToCurve
```

Moreover, the pairing engine in pairing.go accepts and unlimited amount of pairings to be stored.

Recommendation

Limit the length of the input byte array and the number of elements the pairing engine can store.

Status details

Coinbase confirmed that MapToCurve is not problematic and a function called `longerThanLimit` was added to limit the input size for other part of the code.

3.6 KS-SBCF-F-06: G1 and G2 arithmetic fail to catch negative big Int values in scalar input parameters

Severity: Medium

Status: Remediated

Location: g1.go:428, g2.go:439, fp12.go:219, wnaf.go:8

Description

Computations fail when negative values are used in scalar `big.Int` parameters across G1, G2 and pairing arithmetic. For instance:

- pairing_test.go: First case of TestPairingBilinearity fails if `a = big.NewInt(-1)`.
- g1_test.go: First case of TestG1MultiplicativeProperties fails when `s1 = big.NewInt(-1)`.

The origin of this issue is related to the implementation of the scalar multiplication. In g1.go and g2.go the scalar is processed bit by bit and the sign is not taken into account. The same problem affects the GT exponentiation with relation to the $\mathbb{F}_{p^{12}}$ cyclotomic exponentiation and the WNAF algorithm.

Recommendation

Add input validation and filter out negative values in the scalars and/or process accordingly.

Status details

Coinbase corrected the problem by reducing first the scalar to the modulus where is was necessary.

3.7 KS-SBCF-F-07: Prime order subgroup membership not enforced

Severity: Medium

Status: Open

Location: g1.go:168 and g2.go:178, g1.go:183 and g2.go:193, g1.go:201 and g2.go:211 , g1.go and g2.go:337, g1.go, g2.go:385, pairing.go:53, pairing.go:118, pairing.go:84 and gt.go:79, gt.go:84, gt.go:89, gt.go:94 and gt.go:99.

Description

In the BLS12-381 curve, the prime order group is a subgroup of a larger composite-order group. This means that it is possible to mount small subgroup attacks where a point from the composite-order group is used instead of the prime-order group (as described for instance in [11]).

For this reason, we believe that subgroup check for input elements can be enforced (according to the application, scenario and threat model) in the following functions due to possible misuse:

- ToCompressed
- fromBytesUnchecked (and check also that points belong to the curve in this case)
- FromBytes
- ToBytes
- Add
- Doubling
- Neg
- MulScalar
- MultiExp
- AddPair
- GT arithmethic functions
- additionStep
- doublingStep

Recommendation

Enforce subgroup check.

3.8 KS-SBCF-F-08: Coefficient array size in millerLoop can cause OOB

Severity: Medium

Status: Open

Location: pairing.go:167

Description

The size of the of `ellCoeffs` array is hardcoded in the millerLoop before the precomputation of the line functions. However, it is related to the parameter x in bls12381.go. The variable j is used as an index of `ellCoeffs` array during the loop. Any single change in the parameter x (that increments by 1 this parameter), creates an out of bounds (OOB) panic in go when using the pairing engine since j is not checked before accessing the array.

E.g. with

```
var x = bigFromHex("0xd201001100010000")
```

Makes go panic:

```
panic: runtime error: index out of range [68] with length 68 [recovered]
    panic: runtime error: index out of range [68] with length 68

goroutine 7 [running]:
testing.tRunner.func1(0xc0000ba100)
    /usr/lib/go-1.13/src/testing/testing.go:874 +0x3a3
panic(0x58cb20, 0xc000016240)
    /usr/lib/go-1.13/src/runtime/panic.go:679 +0x1b2
github.cbhq.net/c3/bls12-381.(*Engine).preCompute(0xc0000de000,
    0xc0000e4000, 0xc0000d26c0)
    /home/vmr/work/gitlab/coinbase-bls2-audit/src/bls12-381-main/pairing.go:161
        +0x20c
github.cbhq.net/c3/bls12-381.(*Engine).millerLoop(0xc0000de000, 0xc0000d4480)
    /home/vmr/work/gitlab/coinbase-bls2-audit/src/bls12-381-main/pairing.go:175
        +0xc0
```

```
github.cbhq.net/c3/bls12-381.(*Engine).calculate(0xc0000de000, 0xc0000e2090)
    /home/vmr/work/gitlab/coinbase-bls2-audit/src/bls12-381-main/pairing.go:252
        +0x77
github.cbhq.net/c3/bls12-381.(*Engine).Result(...)
    /home/vmr/work/gitlab/coinbase-bls2-audit/src/bls12-381-main/pairing.go:264
github.cbhq.net/c3/bls12-381.TestPairingNonDegeneracy(0xc0000ba100)
    /home/vmr/work/gitlab/coinbase-bls2-audit/src/bls12-381-main
        /pairing_test.go:50 +0x183
testing.tRunner(0xc0000ba100, 0x5b0760)
    /usr/lib/go-1.13/src/testing/testing.go:909 +0xc9
created by testing.(*T).Run
    /usr/lib/go-1.13/src/testing/testing.go:960 +0x350
exit status 2
FAIL    github.cbhq.net/c3/bls12-381    0.006s
```

Recommendation

Derive the size of ellCoeffs at runtime based on x, or create a constant based on x that can be used in the declaration of `ellCoeffs`. Or check the value of j during the loop to avoid OOB.

3.9 KS-SBCF-F-09: Error handling negative number in field conversion

Severity: Low

Status: Open

Location: field_element.go:45

Description

The field conversion function `setBig` does not check if the parameter number is negative.

```
45 func (fe *fe) setBig(a *big.Int) *fe {
46     return fe.setBytes(a.Bytes())
47 }
```

If so the value will return a field element corresponding to the absolute value of the parameter.

Recommendation

Check the sign of the parameter with the `Sign` method and report an error or reduce the parameter to the modulus if negative. The function seems to not be used a lot, thus it may also be a possibility to remove it altogether and thus to remove the dependency to the `math/big` package.

3.10 KS-SBCF-F-10: Non-constant time comparisons

Severity: Low

Status: Remediated

Location: field_element.go:142 and :153

Description

In the functions `cmp` and `equal` The comparison loop exits after a chunk comparison differs.

```
153 func (fe *fe) equal(fe2 *fe) bool {
154     return fe2[0] == fe[0] && fe2[1] == fe[1] && fe2[2] == fe[2] && ...
155 }
```

This is not time constant since the function will return as soon as a condition is not met.

Recommendation

Since a chunk is 64-bit long, it is unlikely this would be exploitable, but may still be problematic depending the usage of the library.

Status details

Constant time comparisons and equality check have been implemented.

3.11 KS-SBCF-F-11: Non-constant time fallback implementation

Severity: Low

Status: Open

Location: arithmetic_fallback: 54, 77, 120, 143, 193, 213. 335 and 439

Description

many of these function are not time constant, some of them even have a warning in the comment.

```
192     if b != 0 {
193         var c uint64
194         z[0], c = bits.Add64(z[0], 13402431016077863595, 0)
195         z[1], c = bits.Add64(z[1], 2210141511517208575, c)
196         z[2], c = bits.Add64(z[2], 7435674573564081700, c)
```

Depending the usage of the library it could introduced leakage of sensitive values.

Recommendation

If the fallback library is not used, it should be removed.

3.12 KS-SBCF-F-12: wNAF scalar multiplication is not constant time

Severity: Low

Status: Open

Location: wnaf.go:1 and g2.go:449

Description

As defined in wnaf.go and g2.go in function `wnafMul`. It has been attacked recently using lattice reduction [9].

Recommendation

The severity of this issue is Low since it is used only to multiply by the cofactor in the hash-into-the-curve operation. This is a compromise between speed and security. The Montgomery ladder is an alternative in this case.

3.13 KS-SBCF-F-13: Serialize functions do not strictly follow zcash serialization checks for invalid input

Severity: Low

Status: Open

Location: g1.go:105, g1.go:121, g2.go:111 and g2.go:127

Description

Some validation checks can be enforced in ToUncompressed and FromCompressed functions in both the G1 and G2 implementations. Moreover, these functions does not strictly follow the zcash point serialization procedure described in Section C.1 and C.2 of [10].

Recommendation

Concerning the implementation of the `FromCompressed` function. In g1.go, check that the length of the input is exactly 48 bytes. Otherwise, abort (Section C.2. of [10]).

In g2.go, check that the length of the input is exactly 96 bytes. Otherwise, abort (Section C.2 of [10]).

Concerning `ToUncompressed`, it can be convenient to check that the the point is on the curve and in the correct subgroup to thwart possible misuse of this function.

In general and related to every deserialization functions in the code, the IETF draft recommends to abort if the following values are found in the first byte of the input: 0x20, 0x60 and 0xE0.

3.14 KS-SBCF-F-14: No error check in integer conversion function

Severity: Low

Status: Open

Location: util.go:7

Description

The `bigFromHex` function does not check nor returns error coming from the `SetString` function. Thus, if a malformed string is converted, the results will be `nil` but not detected as an error.

Recommendation

Check the output error.

Status: Open

3.15 KS-SBCF-F-15: Missing mod reduction in doublingStep and additionStep

Severity: Low

Status: Open

Location: pairing.go:84 and pairing.go:118

Description

- Algorithm 11 in Section B.2 of [1] performs a modulo reduction p when obtaining the coordinate Y. This modular reduction is missing in the doublingStep implementation.
- Concerning the additionStep, Values Y_3 and t_2 contain a mod reduction p in [1]. However, these reductions are not preformed in the additionStep implementation.

Recommendation

Perform the modular reduction to avoid interoperability and / or arithmetic problems.

4 OTHER OBSERVATIONS

This section contains additional observations that are not directly related to the security of the code, and as such have no severity rating or remediation status summary. These observations are either minor remarks regarding good practice or design choices or related to implementation and performance. These items do not need to be remediated for what concerns security, but where applicable we include recommendations.

4.1 KS-SBCF-O-01: Faster subgroup check can be implemented

Status: Informational

Location: g1.go:280 and g2.go:290

Description

In g1.go and in g2.go (function `InCorrectSubgroup`) checks if a point is in the correct subgroup by multiplying the point by q and checking if the result is the identity.

Recommendation

It is possible to use the endorphism to perform this check faster as described in Sections 3.1 and 3.2 of [3].

4.2 KS-SBCF-O-02: Confusion in error messages

Status: Informational

Location: g1.go:57

Description

In g1.go, in the implementation of `FromUncompressed`, the error messages are incorrect. They are always related to the length of the input. However, even if the point is not in the curve or the point is on in the correct subgroup, the error message is related to the input string size.

Recommendation

Present to the user the correct error messages.

4.3 KS-SBCF-O-03: Dead link to serialization rules for uncompressed points

Status: Informational

Location: g1.go:59 and g2.go:69

Description

Related to g1.go and g2.go, in the `FromUncompressed` function, the link to librustzcash does not work anymore.

Recommendation

Add the correct link to the specification.

4.4 KS-SBCF-O-04: Redundant code in Add functions

Status: Informational

Location: g1.go:309 and g2.go:336

Description

There is redundant code to verify that the input points are the same at the beginning of the `Add` function.

Recommendation

Replace lines 336 to 343 by the Equal function (g1.go) that ascertain if two points are the same. The same applies to g2.go.

4.5 KS-SBCF-O-05: Error in comment in Affine function

Status: Informational

Location: g1.go:310

Description

This function transforms a point in Jacobian coordinates to its affine representation. However, the comment at the beginning of the function is incorrect.

Recommendation

Correct comment.

4.6 KS-SBCF-O-06: Dead link to Jacobian coordinates addition formula in Add

Status: Informational

Location: g1.go:328 and g2.go:337

Description

The link is broken.

Recommendation

Replace it with a valid one like

4.7 KS-SBCF-O-07: Speed-up: Implement point addition formulas for special cases

Status: Informational

Location: g1.go:326 and g2.go:337

Description

Different addition formulas can be implemented to speed up addition in certain cases:

- 11M for addition with Z2=1: 7M+4S.
- 7M for addition with Z1=Z2: 5M+2S.
- 6M for addition with Z1=1 and Z2=1: 4M+2S.

vs. always performing 11M + 5S the speed up is considerable.

See http://www.hyperelliptic.org/EFD/g1p/auto-shortw-jacobian-0.html

Recommendation

Implement them to speed up the library, taking into account that special cases will create extra branches in the addition function.

4.8 KS-SBCF-O-08: Dead link to Jacobian coordinates addition formulae

Status: Informational

Location: g1.go:376 and g2.go:386

Description

The link is broken.

Recommendation

Replace it with a valid one.

4.9 KS-SBCF-O-09: Speed-up: Implement doubling formula for special case Z = 1

Status: Informational

Location: g1.go:375 and g2.go:386

Description

There is a doubling formula for special case Z1 = 1 that can be implemented to speed-up the library. See http://www.hyperelliptic.org/EFD/g1p/auto-shortw-jacobian-0.html#doubling-dbl-2009-l and http://www.hyperelliptic.org/EFD/g1p/data/shortw/jacobian-0/doubling/mdbl-2007-bl

Recommendation

Implement it to speed up the library taking into account that this will create extra branches in the doubling function implementation.

4.10 KS-SBCF-O-10: Set call in Double is not needed

Status: Informational

Location: g1.go:400 and g2.go:386.

Description

Calling to set method is not needed.

Recommendation

Replace

```
sub(t[1], t[0], t[2])
```

by

```
sub(ˆr[1], t[0], t[2])
```

to avoid one call to Set.

4.11 KS-SBCF-O-11: Different algorithms are used to clear the cofactor

Status: Informational

Location: g1.go:438 and g2.go:448

Description

In G1 the cofactor is cleared using the double-and-add method (MulScalar) whereas in G2 the cofactor is is cleared using wNAF.

Recommendation

If performance is a goal in this case, use the fastest scalar multiplication algorithm provided by the library in both cases.

4.12 KS-SBCF-O-12: Many Field tests does not initialize value randomly

Status: Informational

Location: fp_test.go

Description

At many places in file fp_test.go, a the method `rand` of types `fe2` is sometimes called with blank identifier. For example at line 554:

```
554         a, b := new(fe2), new(fe2)
555         _, _ = a.rand(rand.Reader)
556         b.set(a)
```

The value will not be randomly generated and stay unchanged. In most of the case the default values will stay at zero and the tests will not cover properly the properties tested.

Recommendation

Use the correct identifier to randomly initialize field values. The behaviour of the methods `rand` differs between type `fe` and `fe2` this may be clearer if it is unified. A check on the return error as well from the `rand` method could help. A good practice is to break tests to smaller tests which test only one property to avoid side effect coming from previous property tests.

4.13 KS-SBCF-O-13: Lack of Field tests with hardcoded values

Status: Informational

Location: fp_test.go

Description

A lot of arithmetic properties are tested but not a lot of test include hardcoded values. This would allow to catch some regressions in the future and allow easier verification of the implementation.

Recommendation

Add test with hardcoded values computed by another library or software. Specially for function using heavy computation like `exp` or `inverse`.

4.14 KS-SBCF-O-14: Speed-up: Faster cofactor clearing

Status: Informational

Location: g1.go:438 and g2.go:448

Description

Cofactor clearing via scalar multiplication as performed in g1.go and g2.go (which is much slower than the former) can be performed using different tricks as described in Section 7 of [6].

Recommendation

Implement one of the following methods:

- Scott et al. endomorphism [8]
- Fuentes-Castaneda et al. trick [7]

and the instantiations in the paper of [4] for BLS curves.

4.15 KS-SBCF-O-15: FromBytes in g1.go and g2.go allows inputs bigger than 96 and 192 bytes

Status: Informational

Location: g1.go:201 and g2.go:211

Status: Informational

Description

The function allows inputs with an unlimited amount of bytes, even if only inputs with sizes 96 and 192 bytes are allowed and utilized.

Recommendation

Enforce length validation on input parameters.

4.16 KS-SBCF-O-16: Reference to Formula 3 in doublingStep function does not exist in paper

Status: Informational

Location: pairing.go:84

Description

We did not find the formula the comment in the doublingStep of pairing.go refers to.

Recommendation

Fix this comment and clarify.

4.17 KS-SBCF-O-17: Unused ladd, ldouble function.

Status: Informational

Location: arithmetic_fallback.go:88 and arithmetic_x86.s:129

Description

Functions `ladd`, `ldouble` are not used and not exported thus it may be removed from the package.

Recommendation

Remove the functions.

4.18 KS-SBCF-O-18: Confusing parameter names.

Status: Informational

Location: field_element.go: 60, 142 and 153

Description

`fe2` type is element representation of $\mathbb{F}_{p^2}$ which is quadratic extension of base field $\mathbb{F}_p$ but for some function is is also the input parameter like for `set` function:

```
func (fe *fe) set(fe2 *fe) *fe {
    fe[0] = fe2[0]
    fe[1] = fe2[1]
    fe[2] = fe2[2]
    fe[3] = fe2[3]
    fe[4] = fe2[4]
    fe[5] = fe2[5]
    return fe
}
```

This notation creates confusion and if later the type and the parameter are used in the same function it may create errors.

Recommendation

Rename the parameters.

4.19 KS-SBCF-O-19: Test vectors from Pairing-Friendly Curves ietf draft (July 2021) produce a different result

Status: Informational

Location: pairing_test.go:15

Description

The same inputs defined in the Pairing-Friendly curves ietf draft are used for testing the pairing operation when using the optimal Ate pairing. However, there is a comparison in place with a hardcoded value from GT that does not correspond to the output of the draft.

Recommendation

For reasons of interoperability, try to reproduce the same result in the tests.

4.20 KS-SBCF-O-20: Unused parameters in bls12-381

Status: Informational

Location: bls12_381.go:17, :71 and :74

Description

The following parameters: `inp`, `cofactorG1` and `cofactorG2` and never used in the library.

Recommendation

Use them if needed or remove them.

4.21 KS-SBCF-O-21: Unused arithmetic functions in the extensions

Status: Informational

Location: fp2.go:63, fp2.go:113, fp6.go:123

Description

The following functions are never used in the code:

```
209 fp2.go:63:15: func (*fp2).fromMont is unused (U1000)
210 fp2.go:113:15: func (*fp2).conjugate is unused (U1000)
211 fp6.go:123:15: func (*fp6).conjugate is unused (U1000)
```

Recommendation

Use them if needed or remove them.

4.22 KS-SBCF-O-22: Redundant return statement in fp.go

Status: Informational

Location: fp.go:169

Description

The return statement at the end of the function is redundant.

```
    // Phase 2
    for i := k; i < 384*2; i++ {
```

```
        double(u, u)
    }
    inv.set(u)
    return
```

Recommendation

It can be removed.

5 APPENDIX A: ABOUT KUDELSKI SECURITY

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit https://www.kudelskisecurity.com.

Kudelski Security
Route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland

Kudelski Security
5090 North 40th Street
Suite 450
Phoenix, Arizona 85018

6 APPENDIX B: DOCUMENT HISTORY

Version	Status	Date	Author	Comments
0.1	Draft	13 August 2021	Kudelski Security Research Team	
0.2	Final	10 September 2021	Kudelski Security Research Team	

Reviewer	Position	Date	Version	Comments
Nathan Hamiel	Head of Security Research	13 August 2021	0.1	

Approver	Position	Date	Version	Comments
Nathan Hamiel	Head of Security Research	13 August 2021	0.1	

7 APPENDIX C: SEVERITY RATING DEFINITIONS

Kudelski Security uses a custom approach when determining criticality of identified issues. This is meant to be simple and fast, providing customers with a quick at a glance view of the risk an issue poses to the system. As with anything risk related, these findings are situational. We consider multiple factors when assigning a severity level to an identified vulnerability. A few of these include:

- Impact of exploitation
- Ease of exploitation
- Likelihood of attack
- Exposure of attack surface
- Number of instances of identified vulnerability
- Availability of tools and exploits

Severity	Definition
High	The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
Medium	The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
Low	The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
Informational	Informational findings are best practice steps that can be used to harden the application and improve processes.

REFERENCES

[1]

Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio López. 2011. Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In *Advances in cryptology – EUROCRYPT 2011*, Springer Berlin Heidelberg, Berlin, Heidelberg, 48–68.

[2]

Daniel J. Bernstein and Bo-Yin Yang. 2019. Fast constant-time gcd computation and modular inversion. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019, 3 (May 2019), 340–398. Retrieved from https://tches.iacr.org/index.php/TCHES/article/view/8298

[3]

Sean Bowe. 2019. Faster subgroup checks for BLS12-381. *IACR Cryptol. ePrint Arch.* 2019, (2019), 814. Retrieved from https://eprint.iacr.org/2019/814

[4]

Alessandro Budroni and Federico Pintore. 2020. Efficient hash maps to $\mathbb{G}_2$ on BLS curves. *Applicable Algebra in Engineering, Communication and Computing* (July 2020). DOI:https://doi.org/10.1007/s00200-020-00453-9

[5]

Alejandro Cabrera Aldaya and Billy Brumley. 2020. When one vulnerable primitive turns viral: Novel single-trace attacks on ECDSA and RSA. In *Transactions on Cryptographic Hardware and Embedded Systems*.

[6]

Armando Faz-Hernández, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. 2021. *Hashing to Elliptic Curves*. Internet Engineering Task Force; Internet Engineering Task Force. Retrieved from https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-11

[7]

Laura Fuentes-Castañeda, Edward Knapp, and Francisco Rodríguez-Henríquez. 2012. Faster Hashing to $\mathbb{G}_2$. DOI:https://doi.org/10.1007/978-3-642-28496-0_25

[8]

Steven D. Galbraith, Xibin Lin, and Michael Scott. 2009. Endomorphisms for Faster Elliptic Curve Cryptography on a Large Class of Curves. In *Advances in cryptology - EUROCRYPT 2009*, Springer Berlin Heidelberg, Berlin, Heidelberg, 518–535.

[9]

Gabrielle De Micheli, Rémi Piau, and Cécile Pierrot. 2019. A Tale of Three Signatures: Practical Attack of ECDSA with wNAF. *Progress in Cryptology - AFRICACRYPT 2020* 12174, (2019), 361–381.

[10]

Yumi Sakemi, Tetsutaro Kobayashi, Tsunekazu Saito, and Riad S. Wahby. 2021. *Pairing-Friendly Curves*. Internet Engineering Task Force; Internet Engineering Task Force. Retrieved from https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-pairing-friendly-curves-10

[11]

Omer Shlomovits. 2021. Baby Sharks: Small-Subgroup Attacks to Disrupt Large Distributed Systems. In *Black Hat USA 2021*. Retrieved from https://www.blackhat.com/us-21/briefings/schedule/index.html#baby-sharks-small-subgroup-attacks-to-disrupt-large-distributed-systems-22608