KUDELSKI SECURITY

Audit of gnark-crypto

Algorand Foundation

20 October 2022

Version: 1.0

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

Kudelski Security ("Kudelski", "we"), the cybersecurity division of the Kudelski Group, was engaged by Algorand Foundation ("the Client") to conduct an external security assessment in the form of a code audit of the cryptographic library gnark-crypto ("the Product") developped by the ConsenSys company. The assessment was conducted remotely by the Kudelski Security Team and coordinated by Antonio De La Piedra, Sylvain Pelissier and Nathan Hamiel, Head of Cybersecurity Research. The audit took place from June 27, 2022 to July 20, 2022 and invloved 30 person-days of work. The audit focused on the following objectives:

- To provide a professional opinion on the maturity, adequacy, and efficiency of the software solution in examination.
- 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

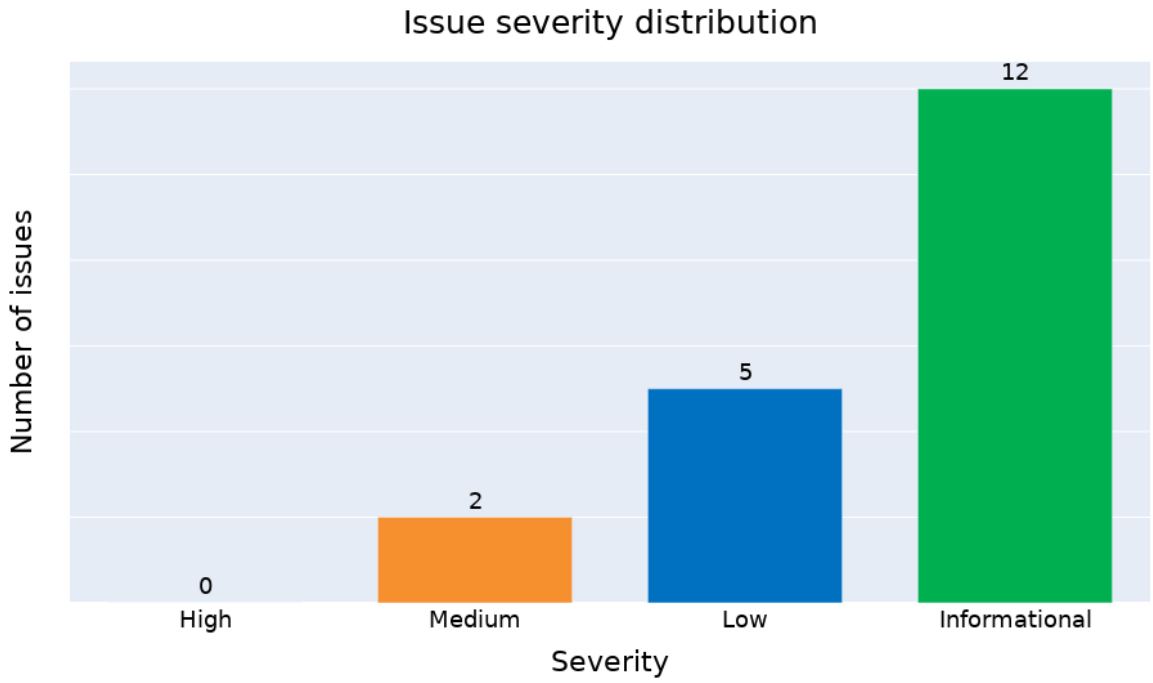
The scope of this work is a code audit of the Product written in Go, with a particular attention to safe implementation of hashing, randomness generation, protocol verification, and potential for misuse and leakage of secrets. The client has noted that constant-time analysis of the Product is out of scope of this audit. The target of the audit was the cryptographic code related to the elliptic curves **BLS12-381** and **BN254** at https://github.com/ConsenSys/gnark-crypto. The **BN254** curve is also named **alt_bn128** in different context [6]. We audited the commit number: `450e0206211eea38bbb5b5ffddf262efe65bd011` of the repository,

Particular attention was given to the security related to its usage regarding the opcode implementation in Algorand virtual machine.

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 the Client and the drafting of this report.

As a result of our work, we have identified **2 Medium**, **5 Low** and **12 Informational** findings.



1.3 Issue Summary List

The following security issues were found:

ID	Severity	Finding	Status
KS-SBCF-F-01	**Medium**	Missing Karabina decompression function case	**Acknowledged**
KS-SBCF-F-02	**Medium**	Missing batch Torus compression check	**Acknowledged**
KS-SBCF-F-03	**Low**	BN254 curve has a low security margin	**Acknowledged**

ID	Severity	Finding	Status
KS-SBCF-F-04	**Low**	Inversion of 0 element is not detected in field arithmetic	**Acknowledged**
KS-SBCF-F-05	**Low**	Batch Jacobian conversion left some element unchanged	**Acknowledged**
KS-SBCF-F-06	**Low**	Out-of-bounds access situation	**Acknowledged**
KS-SBCF-F-07	**Low**	Dependency with security issue	**Acknowledged**

The following are observations related to general design and improvements:

ID	Severity	Finding
KS-SBCF-O-01	**Informational**	Pairing function does not check the group membership
KS-SBCF-O-02	**Informational**	Several TODO comments in the code
KS-SBCF-O-03	**Informational**	Error in comments
KS-SBCF-O-04	**Informational**	Error handling is not performed in different places across the code
KS-SBCF-O-05	**Informational**	Multiple functions across the code are not constant-time
KS-SBCF-O-06	**Informational**	Multiple possible nil dereferences across the code
KS-SBCF-O-07	**Informational**	Low code coverage of ecc/bls12-381/multiexp.go and ecc/bls12-381/internal/fptower/e12_pairing.go
KS-SBCF-O-08	**Informational**	Error in test fptower.TestE2ReceiverIsOperand with data race detector enabled
KS-SBCF-O-09	**Informational**	The references to the hash-to-curve draft are not updated to the last version
KS-SBCF-O-10	**Informational**	The function g1SetZ and g1SetZ are commented
KS-SBCF-O-11	**Informational**	The function g1SqrtRatio requires that v is different from 0
KS-SBCF-O-12	**Informational**	Providing a large value for config.NbTasks in the multi-exponentiation algorithm implementation makes the library panic

2 TECHNICAL DETAILS OF SECURITY FINDINGS

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

2.1 KS-SBCF-F-01: Missing Karabina decompression function case

Severity: Medium

Status: Acknowledged

Location:

- ecc/bn254/internal/fptower/e12.go:247
- ecc/bls12-381/internal/fptower/e12.go:247

Description

The Karabina decompression function is used to decompress the cyclotomic square result. This is used in the final exponentiation step of pairing computations. In the gnark library, The tower construction is the following:

$$\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - (9i + 1))$$

$$\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - u)$$

$$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v)$$

Thus, elements of $\mathbb{F}_{12}$ are written this way: $(h_0 + h_1 v + h_2 v^2) + (h_3 + h_4 v^2 + h_5 v^2)w$. For the cyclotomic square in [5] elements are written in this form: $(g_0 + g_1 w) + (g_2 + g_3 w)v + (g_4 + g_5 w)v^2$. Developing and rearranging the previous form gives: $(h_0 + h_3 w) + (h_1 + h_4 w)v + (h_2 + h_5 w)v^2$. The decompression function as defined in the Theorem 3.1 of the previous paper applies the same way. However, a check on the value of g_2 which correspond to h_3 in the implementation is needed before the division to avoid inversion of zero. However this check is not perform in the code:

```
// DecompressKarabina Karabina's cyclotomic square result
func (z *E12) DecompressKarabina(x *E12) *E12 {

    var t [3]E2
    var one E2
    one.SetOne()

    // t0 = g1^2
    t[0].Square(&x.C0.B1)
    // t1 = 3 * g1^2 - 2 * g2
    t[1].Sub(&t[0], &x.C0.B2).
        Double(&t[1]).
        Add(&t[1], &t[0])
        // t0 = E * g5^2 + t1
    t[2].Square(&x.C1.B2)
    t[0].MulByNonResidue(&t[2]).
        Add(&t[0], &t[1])
    // t1 = 1/(4 * g3)
    t[1].Double(&x.C1.B0).
        Double(&t[1]).
        Inverse(&t[1]) // costly
```

In the case of $h_3 = 0$ the implementation would do an inversion by zero and return 0. This ends up in a wrong result of the pairing computation.

Recommendation

All the test cases for decompression should be implemented.

Status Details

Consensys acknowledged the problem and corrected it in PR #219.

2.2 KS-SBCF-F-02: Missing batch Torus compression check

Severity: Medium

Status: Acknowledged

Location:

- ecc/bn254/internal/fptower/e12.go:769
- ecc/bls12-381/internal/fptower/e12.go:798

Description

The torus compression as defined in [8] is defined to be $f(c_0 + c_1\delta) = \frac{1+c_0}{c_1}$. It is not defined for numbers 1 and -1. In the current implementation if these values are given as input the compression will compute a wrong value:

```
// BatchCompressTorus GT/E12 elements to half their size
// using a batch inversion
func BatchCompressTorus(x []E12) ([]E6, error) {

    n := len(x)
    if n == 0 {
        return []E6{}, errors.New("invalid input size")
    }

    var one E6
    one.SetOne()
    res := make([]E6, n)

    for i := 0; i < n; i++ {
        res[i].Set(&x[i].C1)
    }

    t := BatchInvertE6(res) // costs 1 inverse

    for i := 0; i < n; i++ {
        res[i].Add(&x[i].C0, &one).
            Mul(&res[i], &t[i])
    }
```

```
    return res, nil
}
```

Recommendation

Like for the function `CompressTorus` a check on each `C1` coordinates should be implemented.

2.3 KS-SBCF-F-03: BN254 curve has a low security margin

Severity: Low

Status: Acknowledged

Location: ecc/bn254/

Description

According to recent development of the tower field sieve algorithm [2], some curve security are now considered reduced. For BN254 curve the resulting field has size 3072 bits whereas more than 5000 bits is now advised. The security level is now considered to be 103 bits for this curve [1]. Even though the security of the scheme is not practically broken this curve should not be used for future implementation. With a bigger resulting field, BLS12-381 curve seems to have a larger security margin.

Recommendation

The end user should be warned that this curve should be used only for compatibility results and advised to used other curves with larger security margin.

Status Details

ConsenSys estimated the security of BN254 to 103 bits following [4] and using the software shipped with that paper https://gitlab.inria.fr/tnfs-alpha/alpha. However this is the curve used in Ethereum and they implemented it for compatibility reasons.

2.4 KS-SBCF-F-04: Inversion of 0 element is not detected in field arithmetic

Severity: Low

Status: Acknowledged

Location:

- ecc/bn254/internal/fptower/e2_bn254.go:76
- ecc/bn254/internal/fptower/e2.go:261
- ecc/bn254/internal/fptower/e6.go:245, 284
- ecc/bn254/internal/fptower/e12.go:377, 397, 429
- ecc/bls12-381/internal/fptower/e2_bls381.go:78
- ecc/bls12-381/internal/fptower/e2.go:269
- ecc/bls12-381/internal/fptower/e6.go:245, 292
- ecc/bls12-381/internal/fptower/e12.go:377, 405, 429

An inversion by 0 is not detected in the field arithmetic for the bls-12-381 and bn254 curves. For instance, the test below fails without reporting an inversion by zero instead it ends up with a wrong result.

```
properties.Property("[BN254] mul & inverse fails", prop.ForAll(
        func(a, b *E2) bool {
            var c, d E2
            b.SetZero()
            d.Inverse(b)
            c.Set(a)
            c.Mul(&c, b).Mul(&c, &d)
            return c.Equal(a)
        },
        genA,
        genB,
    ))
```

Since the inversion operations in the tower field extensions $\mathbb{F}_2$, $\mathbb{F}_6$ and $\mathbb{F}_{12}$ also rely on successive inversions, and inversion by zero could happen without letting the user knows. For instance, in the inversion operation of an element in $\mathbb{F}_6$, the temporary variable `t6` is inverted:

```
// Inverse an element in E6
func (z *E6) Inverse(x *E6) *E6 {
    // Algorithm 17 from https://eprint.iacr.org/2010/354.pdf
    // step 9 is wrong in the paper it's t1-t4
    var t0, t1, t2, t3, t4, t5, t6, c0, c1, c2, d1, d2 E2
    t0.Square(&x.B0)
    t1.Square(&x.B1)
    t2.Square(&x.B2)
    t3.Mul(&x.B0, &x.B1)
    t4.Mul(&x.B0, &x.B2)
    ...
    t6.Add(&t6, &d1)
    t6.Inverse(&t6)
    z.B0.Mul(&c0, &t6)
    z.B1.Mul(&c1, &t6)
    z.B2.Mul(&c2, &t6)

    return z
}
```

The functions `BatchInvertE2`, `BatchInvertE6`, `BatchInvertE12` and `Exp` (used with a negative exponent) have also the same behavior.

2.4.1 Recommendation

We recommend the client to check if an inversion of 0 is performed and alert the user about this fact. For example, the `ModInverse` of the big numbers package the input value is unchanged and the return value is nil in case the inverse does not exist.

Status Details

ConsenSys acknowledged the behavior and use this is as a convention *i.e* if the input equals 0 then the output equals the input.

2.5 KS-SBCF-F-05: Batch Jacobian conversion left some element unchanged

Severity: Low

Status: Acknowledged

Location:

- src/ecc/bn254/g1.go:823
- src/ecc/bls12-381/g1.go:840

Description

The public function `BatchJacobianToAffineG1` assumes that the `result` parameters contains only zero elements. However, if the parameter was used previously, each element matching a index corresponding to a point at infinity in the `points` would be left unchanged. This would lead to a wrong result.

```
// batch convert to affine.
parallel.Execute(len(points), func(start, end int) {
    for i := start; i < end; i++ {
        if zeroes[i] {
            // do nothing, X and Y are zeroes in affine.
            continue
        }
```

So far in the library the input parameter is always used with an all zero input but this function is exported by the package thus may be used in the wrong way. In addition, this function is not tested.

Recommendation

Either zeroize the elements matching a point at infinity or warn the user of the correct usage.

2.6 KS-SBCF-F-06: Out-of-bounds access situation

Severity: Low

Status: Acknowledged

Location:

- ecc/bls12-381/g1.go:819
- ecc/bn254/g1.go:802

In the `BatchJacobianToAffineG1` public function, the loop is based on the size of the input parameter `points` instead of `result`. If the `result` size is not equal to the size of `points`, an out-of-bounds (OOB) situation is possible.

```
func BatchJacobianToAffineG1(points []G1Jac, result []G1Affine) {
    zeroes := make([]bool, len(points))
    accumulator := fp.One()

    // batch invert all points[].Z coordinates with Montgomery batch
     ↪ inversion trick
    // (stores points[].Z^-1 in result[i].X to avoid allocating a slice
     ↪ of fr.Elements)
    for i := 0; i < len(points); i++ {
        if points[i].Z.IsZero() {
            zeroes[i] = true
            continue
        }
        result[i].X = accumulator
```

Recommendation

For `BatchJacobianToAffineG1`, we recommend the client to check the length of `result` and validate that it has the same length as the `points` array.

2.7 KS-SBCF-F-07: Dependency with security issue

Severity: Low

Status: Acknowledged

Location: See below

Description

The dependency `pkg:golang/golang.org/x/text@v0.3.6` contains the following security issue: [CVE-2021-38561] CWE-125: Out-of-bounds. See for instance https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2021-38561.

Recommendation

We recommend the client to update the affected dependency.

Status Details

The Client communicated to us that this is a transitive dependency, pulled by the latest version of `golang.org/x/crypto` package. This dependency is used only for blake2b calls in the `twistededwards` package. There is no impact on the Algorand scope.

3 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.

3.1 KS-SBCF-O-01: Pairing function does not check the group membership

Location:

- ecc/bn254/pairing.go:35
- ecc/bls12-381/pairing.go:34

Description

The function `Pair` performing the pairing operation does implement check on the input P and Q to be in the proper group before executing pairing operations. If the input are not in the correct groups, the operation could result in totally erroneous output.

```
func Pair(P []G1Affine, Q []G2Affine) (GT, error) {
    f, err := MillerLoop(P, Q)
    if err != nil {
        return GT{}, err
    }
    return FinalExponentiation(&f), nil
}
```

For example, a `SetRandom` method call on the G1 and G2 type elements (`fp.Element`, `fptower.E2` respectively) which are exposed publicly doesn't enforce that the resulting point is inside the right subgroup. Instead, a call to clear to cofactor is needed to be sure that the generated random element is in the right subgroup. The client could clarify this behaviour to avoid misusage.

Recommendation

The input element should have been deserialized before usage with the functions `SetByte` which check the group membership.

Moreover, in certain occasions it could be useful to have a random element generation function for both G1, G2 based on the generation of a random seed that is feed into the `HashIntoG1`/`HashIntoG2` functions.

Notes

Consensys assumes that all the input points are always coming from the `SetBytes` function which checks the group membership.

3.2 KS-SBCF-O-02: Several TODO comments in the code

Location: See below

Description

Some comment contains TODO. Some of them concern missing check during computation for example in src/ecc/bn254/hash_to_g2.go:76:

```
gx1.Square(&x1) //    11. gx1 = x1²
//TODO: Beware A != 0
//12. gx1 = gx1 + A
gx1.Mul(&gx1, &x1)                  //    13. gx1 = gx1 * x1
gx1.Add(&gx1, &bTwistCurveCoeff)    //    14. gx1 = gx1 + B
gx1NotSquare = gx1.Legendre() >> 1 //    15.  e1 = is_square(gx1)
// gx1NotSquare = 0 if gx1 is a square, -1 otherwise
```

This may indicates a missing case not handle during computation.

Other TODO comments across the code appear at:

- ecc/bls12-381/hash_to_g1.go:185, 207, 217, 353
- ecc/bls12-381/hash_to_g2.go:231, 253, 269, 417
- ecc/bl-12-381/fp/internal/e12.go:467, 506, 507
- ecc/bn254/hash_to_g1.go:39, 61
- ecc/bn254/internal/fptower/e12.go: 467, 506, 507

Recommendation

These comments should be addressed and corrected.

Notes

3.3 KS-SBCF-O-03: Error in comments

Location:

- ecc/bn254/internal/fptower/e2.go:173
- ecc/bn254/internal/fptower/e12.go:418, 464, 476
- ecc/bls-12-381/g1.go:339
- ecc/bn254/g1.go:339
- ecc/bn254/g2.go:344
- ecc/bn254/internal/fptower/e2_bn254.go:53

For the $\mathbb{F}_2$ and $\mathbb{F}_{12}$ exponentiation functions the comment says:

```
// Exp sets z=x^k (mod q^2) and returns it
```

However the exponentiation is computed in $\mathbb{F}_2$ and $\mathbb{F}_{12}$.

The comment in line 339 of g1.go appears as:

```
// FromAffine sets p = Q, p in Jacboian, Q in affine
```

instead of Jacobian.

Algorihtm 22 of [3] is wrongly referenced in function `squareGenericE2`:

```
// squareGenericE2 sets z to the E2-product of x,x returns z
// note: do not rename, this is referenced in the x86 assembly impl
func squareGenericE2(z, x *E2) {
    // algo 22 https://eprint.iacr.org/2010/354.pdf
```

Recommendation

The comment should be corrected.

3.4 KS-SBCF-O-04: Error handling is not performed in different places across the code

Location:

- ecc/bls12-381/multiexp.go:340, 1410
- ecc/bls12-381/fp/element.go:941, 1094
- ecc/bn254/multiexp.go:340, 1410
- ecc/bn254/fp/element.go:803, 952

There are calls to the `panic` function at several places in the code. For a node running the code it can create DoS problems is the `panic` function is triggered. For example, the function `msmInnerG1Jac` doesn't provide error handlings for the parameter `c`, and panics when an input value is not known:

```
func msmInnerG1Jac(p *G1Jac, c int, points []G1Affine, scalars
 ↪  []fr.Element, splitFirstChunk bool) {

    switch c {

    case 4:
[...]
    default:
        panic("not implemented")
    }
```

Also for instance in the `SetString` method (element.go):

```
func (z *Element) SetString(number string) *Element {
    // get temporary big int from the pool
    vv := bigIntPool.Get().(*big.Int)

    if _, ok := vv.SetString(number, 0); !ok {
        panic("Element.SetString failed -> can't parse number into a
         ↪  big.Int " + number)
```

```
    }

    z.SetBigInt(vv)

    // release object into pool
    bigIntPool.Put(vv)

    return z
}
```

Recommendation

We recommend the client to perform error handling when possible instead of a call to `panic`.

3.5 KS-SBCF-O-05: Multiple functions across the code are not constant-time

Location:

- ecc/bls12-381/g2.go:386 and 425
- ecc/bls12-381/fp/element.go:874
- ecc/bls12-381/internal/fptower/e2.go:174
- ecc/bls12-381/internal/fptower/e12.go: 468, 420, and 508
- ecc/bn254/g2.go:402 and 950
- ecc/bn254/fp/element.go:738
- ecc/bn254/internal/fptower/e2.go:174
- ecc/bn254/internal/fptower/e12.go: 468, 420, and 508

Several functions across the code are not constant-time. The windowed scalar multiplication G1 method (`mulWindowed`) computes a 2-bit windowed scalar multiplication, based on the sometimes, sensitive exponent `s`:

```
// mulWindowed computes a 2-bits windowed scalar multiplication
func (p *G1Jac) mulWindowed(a *G1Jac, s *big.Int) *G1Jac {

    var res G1Jac
    var ops [3]G1Jac

    res.Set(&g1Infinity)
    ops[0].Set(a)
    ops[1].Double(&ops[0])
    ops[2].Set(&ops[0]).AddAssign(&ops[1])

    b := s.Bytes()
    for i := range b {
        w := b[i]
        mask := byte(0xc0)
        for j := 0; j < 4; j++ {
            res.DoubleAssign().DoubleAssign()
            c := (w & mask) >> (6 - 2*j)
            if c != 0 {
                res.AddAssign(&ops[c-1])
            }
            mask = mask >> 2
        }
    }
    p.Set(&res)

    return p

}
```

The exponentiation method of `Element` type relies on square and multiply:

```
    for i := e.BitLen() - 2; i >= 0; i-- {
        z.Square(z)
```

```
        if e.Bit(i) == 1 {
            z.Mul(z, &x)
        }
    }
```

The exponentiation and `Sqrt` methods of `E2` type:

```
func (z *E2) Exp(x E2, k *big.Int) *E2 {
    if k.IsUint64() && k.Uint64() == 0 {
        return z.SetOne()
    }

    e := k
    if k.Sign() == -1 {
        // negative k, we invert
        x.Inverse(&x)

        // we negate k in a temp big.Int since
        // Int.Bit(_) of k and -k is different
        e = bigIntPool.Get().(*big.Int)
        defer bigIntPool.Put(e)
        e.Neg(k)
    }

    z.SetOne()
    b := e.Bytes()
    for i := 0; i < len(b); i++ {
        w := b[i]
        for j := 0; j < 8; j++ {
            z.Square(z)
            if (w & (0b10000000 >> j)) != 0 {
                z.Mul(z, &x)
            }
        }
```

```
    }

    return z
}
```

The exponentiation and the GLV methods for $\mathbb{F}_{12}$:

```
// uses 2-bits windowed method
func (z *E12) Exp(x E12, k *big.Int) *E12 {
    if k.IsUint64() && k.Uint64() == 0 {
        return z.SetOne()
    }

    e := k
    if k.Sign() == -1 {
        // negative k, we invert
        x.Inverse(&x)

        // we negate k in a temp big.Int since
        // Int.Bit(_) of k and -k is different
        e = bigIntPool.Get().(*big.Int)
        defer bigIntPool.Put(e)
        e.Neg(k)
    }

    var res E12
    var ops [3]E12

    res.SetOne()
    ops[0].Set(&x)
    ops[1].Square(&ops[0])
    ops[2].Set(&ops[0]).Mul(&ops[2], &ops[1])

    b := e.Bytes()
```

```
    for i := range b {
        w := b[i]
        mask := byte(0xc0)
        for j := 0; j < 4; j++ {
            res.Square(&res).Square(&res)
            c := (w & mask) >> (6 - 2*j)
            if c != 0 {
                res.Mul(&res, &ops[c-1])
            }
            mask = mask >> 2
        }
    }
    z.Set(&res)

    return z
}
```

The cyclotomic exponentiation for $\mathbb{F}_{12}$:

```
// uses 2-NAF decomposition
// x must be in the cyclotomic subgroup
// TODO: use a windowed method
func (z *E12) CyclotomicExp(x E12, k *big.Int) *E12 {
    if k.IsUint64() && k.Uint64() == 0 {
        return z.SetOne()
    }

    e := k
    if k.Sign() == -1 {
        // negative k, we invert (=conjugate)
        x.Conjugate(&x)

        // we negate k in a temp big.Int since
```

```
        // Int.Bit(_) of k and -k is different
        e = bigIntPool.Get().(*big.Int)
        defer bigIntPool.Put(e)
        e.Neg(k)
    }

    var res, xInv E12
    xInv.InverseUnitary(&x)
    res.SetOne()
    eNAF := make([]int8, e.BitLen()+3)
    n := ecc.NafDecomposition(e, eNAF[:])
    for i := n - 1; i >= 0; i-- {
        res.CyclotomicSquare(&res)
        if eNAF[i] == 1 {
            res.Mul(&res, &x)
        } else if eNAF[i] == -1 {
            res.Mul(&res, &xInv)
        }
    }
    z.Set(&res)
    return z
}
```

Recommendation

We recommend the client to rely on constant-time implementations when possible. For example, branch-free point addition and doubling can be performed using the complete addition law formulas proposed by Renes et al. [7]. More precisely via Algorithm 7 for point addition and via Algorithm 9 for point doubling with an increase in the number of multiplications (1 multiplication for point addition and 6 multiplications for doubling).

3.6 KS-SBCF-O-06: Multiple possible nil dereferences across the code

Location:

- Several, see below.

Across the code there is no validation of input arguments against `nil`, which can make an implementation relying on gnark panic e.g.

```
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=0x20 pc=0x55a500]
```

In ecc/bls-12-381/:

- g1.go : `Set`, `ScalarMultiplication`, `Add`, `Sub`, `Equal`, `FromJacobian`, `Neg`, `AddAssign`, `AddMixed`, `Double`, `FromAffine`, `mulWindowed`, `phi`, `mulGLV`, `ClearCofactor`, `FromJacExtended`, `unsafeFromJacExtended`, `add`, `double`, `subMixed`, `addMixed`, `doubleNegMixed`, `doubleMixed`, `BatchScalarMultiplicationG1`.
- g2.go: `Set`, `ScalarMultiplication`, `Add`, `Sub`, `Equal`, `Neg`, `FromJacobian`, `SubAssign`, `AddAssign`, `AddMixed`, `Double`, `FromAffine`, `mulWindowed`, `psi`, `phi`, `mulGLV`, `ClearCofactor`, `fromJacExtended`, `unsafeFromJacExtended`, `add`, `double`, `subMixed`, `addMixed`, `doubleNegMixed`, `doubleMixed`, `BatchScalarMultiplicationG2`.
- hash_to_g1.go: `g1IsogenyXNumerator`, `g1IsogenyXDenominator`, `g1IsogenyYNumerator`, `g1IsogenyYDenominator`, `g1Isogeny`, `g1SqrtRatio`, `g1MulByZ`, `mapToCurve`, `g1EvalPolynomial`, `g1Sgn0`, `g1NotZero`.
- hash_to_g2.go: `g2IsogenyXNumerator`, `g2IsogenyXDenominator`, `g2IsogenyYNumerator`, `g2IsogenyYDenominator`, `g2Isogeny`, `g2SqrtRatio`, `g2NotOne`, `g2MulByZ`, `mapToCurve2`, `g2EvalPolynomial`, `g2Sgn0`, `g2NotZero`.

- multiexp.go: `msmInnerG1Jac`, `msmReduceChunkG1Affine`, `msmInnerG2Jac`, `msmReduceChunkG2Affine`.
- pairing.go: `FinalExponentiation`, `DoubleStep`, `AddMixedStep`.
- ecc/bls12-381/element.go: `Mul`, `Square`, `Add`, `Double`, `Sub`, `Neg`, `Select`, `mulGeneric`, `fromMontGeneric`, `reduceGeneric`, `butterFlyGeneric`, `Exp`, `ToBigInt`, `ToBitIntRegular`, `Bytes`, `setBytes`, `setBigInt`, `approximate`, `linearComb`, `montReduceSigned`, `negL`, `mulWNonModular`, `linearCombNonModular`.
- ecc/bls12-381/internal/fp/tower/e2_adx_amd64.s: `Mul`, `MulByNonResidue`, `Square`.
- ecc/bls12-381/internal/fp/tower/e2_bls381_fallback.go: `MulByNonResidue`, `Mul`, `Square`.
- ecc/bls12-381/internal/fp/tower/e2_bls381.go: `mulGenericE2`, `squareGenericE2`, `MulByNonResidueInv`, `Inverse`, `norm`, `MulBybTwistCurveCoeff`.
- ecc/bls12-381/internal/fp/tower/e2_fallback.go: `addE2`, `subE2`, `doubleE2`, `negE2`.
- ecc/bls12-381/internal/fp/tower/e2.go: `Equal`, `Set`, `Add`, `Sub`, `Double`, `Neg`, `MulByElement`, `Conjugate`, `Exp`, `Sqrt`, `Select`, `Div`.
- ecc/bls12-381/internal/fp/tower/e2.go: `Equal`, `Set`, `Add`, `Neg`, `Sub`, `Double`, `MulByNonResidue`, `MulByE2`, `MulBy01`, `MulBy1`, `Mul`, `Square`, `Inverse`.
- ecc/bls12-381/internal/fp/tower/e12_pairing.go: `ExpHalf`, `Expt`, `MulBy014`, `Mul014By014`.
- ecc/bls12-381/internal/fp/tower/e12.go: `Equal`, `Set`, `Add`, `Sub`, `Double`, `Square`, `CyclotomicSquareCompressed`, `DecompressKarabina`, `CyclotomicSquare`, `Inverse`, `Exp`, `CyclotomicExp`, `ExpGLV`, `InverseUnitary`, `Conjugate`,
- ecc/bls12-381/internal/fptower/frobenius.go: `Frobenius`, `FrobeniusSquare`, `MulByNonResidue1Power1`, `MulByNonResidue1Power2`, `MulByNonResidue1Power3`, `MulByNonResidue1Power4`, `MulByNonResidue1Power5`, `MulByNonResidue2Power1`, `MulByNonResidue2Power2`, `MulByNonResidue2Power3`, `MulByNonResidue2Power4`, `MulByNonResidue2Power5`.

The same applies to the functions related to the BN254 curve.

Recommendation

We recommend the client to validate the input parameters against `nil` in public functions to avoid crashes in an application that relies on gnark.

Status Details

The client has communicated to us that they assume that the inputs are properly allocated.

3.7 KS-SBCF-O-07: Low code coverage of ecc/bls12-381/multiexp.go and ecc/bls12-381/internal/fptower/e12_pairing.go

Location:

- ecc/bls12-381/multiexp.go
- ecc/bn254/multiexp.go
- ecc/bls12-381/internal/fptower/e12_pairing.go
- ecc/bn254/internal/fptower/e12_pairing.go

Description

According to:

```
go test -race -covermode=atomic -coverprofile=cover.out
go tool cover -html=cover.out
```

The multiexp_test.go test only covers 38 % of the multi-exponentiation implementation (37.1 % in the BN254 case). Further, e12_pairing.go has a code coverage of only 3.6 % (3.2 % in the BN254 case).

Recommendation

We recommend the client to increase the amount of testing of both modules as good practice.

3.7.1 Status Details

The Client communicated to us that `e12_pairing.go` is tested at the root package `in pairing_test.go`.

3.8 KS-SBCF-O-08: Error in test fptower.TestE2ReceiverIsOperand with data race detector enabled

Location:

- ecc/bls12-381/internal/fptower/
- ecc/bn254/internal/fptower/

Description

Via go test -race in `bls12-381/internal/fptower` and `bn254/internal/fptower`:

```
WARNING: DATA RACE
Write at 0x00000083f8c5 by goroutine 32:
fptower.TestE2ReceiverIsOperand()
ecc/bls12-381/internal/fptower/e2_test.go:196 +0x1ba6
  testing.tRunner()
/usr/lib/go-1.17/src/testing/testing.go:1259 +0x22f
  testing.(*T).Run·dwrap·21()
/usr/lib/go-1.17/src/testing/testing.go:1306 +0x47

Previous read at 0x00000083f8c5 by goroutine 34:
fptower.TestE2Ops()
ecc/bls12-381/internal/fptower/e2_test.go:418 +0x2046
  testing.tRunner()
/usr/lib/go-1.17/src/testing/testing.go:1259 +0x22f
  testing.(*T).Run·dwrap·21()
/usr/lib/go-1.17/src/testing/testing.go:1306 +0x47

Goroutine 32 (running) created at:
  testing.(*T).Run()
/usr/lib/go-1.17/src/testing/testing.go:1306 +0x726
  testing.runTests.func1()
/usr/lib/go-1.17/src/testing/testing.go:1598 +0x99
  testing.tRunner()
/usr/lib/go-1.17/src/testing/testing.go:1259 +0x22f
```

```
  testing.runTests()
/usr/lib/go-1.17/src/testing/testing.go:1596 +0x7ca
  testing.(*M).Run()
/usr/lib/go-1.17/src/testing/testing.go:1504 +0x9d1
  main.main()
_testmain.go:115 +0x22b

Goroutine 34 (finished) created at:
  testing.(*T).Run()
/usr/lib/go-1.17/src/testing/testing.go:1306 +0x726
  testing.runTests.func1()
/usr/lib/go-1.17/src/testing/testing.go:1598 +0x99
  testing.tRunner()
/usr/lib/go-1.17/src/testing/testing.go:1259 +0x22f
  testing.runTests()
/usr/lib/go-1.17/src/testing/testing.go:1596 +0x7ca
  testing.(*M).Run()
/usr/lib/go-1.17/src/testing/testing.go:1504 +0x9d1
  main.main()
_testmain.go:115 +0x22b
==================
--- FAIL: TestE2ReceiverIsOperand (0.35s)
    e2_test.go:193: disabling ADX
    testing.go:1152: race detected during execution of test
FAIL
exit status 1
FAIL    github.com/consensys/gnark-crypto/ecc/bls12-381/internal/fptower
 ↪  1.468s
```

3.9 KS-SBCF-O-09: The references to the hash-to-curve draft are not updated to the last version

Location:

- ecc/bls12-381/hash_to_g1.go: 136, 187, 323
- ecc/bls12-381/hash_to_g2.go: 136, 233, 378
- ecc/bn254/hash_to_g1.go: 26, 146
- ecc/bn254/hash_to_g2.go: 26, 156

Description

The last version, 16, is located at https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/.

3.10 KS-SBCF-O-10: The function g1SetZ and g1SetZ are commented

Location:

- ecc/bls12-381/hash_to_g1.go
- ecc/bls12-381/hash_to_g2.go

Description

The function `g1SetZ` and `g1SetZ` are commented:

```
/*
// g1SetZ sets z to [11].
func g1SetZ(z *fp.Element) {
    z.Set( &fp.Element {9830232086645309404, 1112389714365644829,
 ↪  8603885298299447491, 11361495444721768256, 5788602283869803809,
 ↪  543934104870762216} )
}*/

/*
// g2SetZ sets z to [-2, -1].
func g2SetZ(z *fptower.E2) {
    z.Set( &fptower.E2 {
A0: fp.Element{9794203289623549276, 7309342082925068282,
 ↪  1139538881605221074, 15659550692327388916, 16008355200866287827,
 ↪  582484205531694093},
```

```
A1: fp.Element{4897101644811774638, 3654671041462534141,
 ↪  569769440802610537, 17053147383018470266, 17227549637287919721,
 ↪  291242102765847046},
} )
}*/
```

3.11 KS-SBCF-O-11: The function g1SqrtRatio requires that v is different from 0

Status: Acknowledged

Location:

- ecc/bls12-381/hash_to_g1.go:135

Description

According to the hash-to-curve ietf draft, F.2.1.2, the optimized `sqrt_ratio` for q congruent with 3 mod 4 requires that v is not zero. This check is not performed the `g1SqrtRatio` function:

```
// g1SqrtRatio computes the square root of u/v and returns 0 iff u/v was
 ↪  indeed a quadratic residue
// if not, we get sqrt(Z * u / v). Recall that Z is non-residue// The
 ↪  main idea is that since the computation of the square root involves
 ↪  taking large powers of u/v, the inversion of v can be avoided
func g1SqrtRatio(z *fp.Element, u *fp.Element, v *fp.Element) uint64 {
    // Taken from
     ↪  https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/13/
     ↪  F.2.1.2. q = 3 mod 4
    var tv1 fp.Element
    tv1.Square(v)
    var tv2 fp.Element
    tv2.Mul(u, v)
    tv1.Mul(&tv1, &tv2)
```

```
    var y1 fp.Element
    {
```

Status Details

The client has communicated to us that they assume that the caller (the SSWU map) fulfills its side of the contract, that is, providing a non-zero value `v`.

3.12 KS-SBCF-O-12: Providing a large value for config.NbTasks in the multi-exponentiation algorithm implementation makes the library panic

Status: Acknowledged

Location:

- ecc/bls12-381/multiexp.go:176, 87
- ecc/bn254/multiexp.go:176, 87

Description

The configuration for the public function `MultiExp`, including the `nbTasks` parameter is not validated. For a very large value, the library crashes with a panic and an out of memory error. To reproduce this issue, `MultiExp` can be called as:

```
splitted2.MultiExp(samplePointsLarge[:], sampleScalars[:],
↪   ecc.MultiExpConfig{NbTasks: 9999999999999})
```

producing the following output:

```
fatal error: runtime: out of memory

runtime stack:
runtime.throw({0x600751, 0x48c273c00000})
```

```
/usr/lib/go-1.17/src/runtime/panic.go:1198 +0x71
runtime.sysMap(0xc009400000, 0x428ee0, 0xc001289e90)
/usr/lib/go-1.17/src/runtime/mem_linux.go:169 +0x96
runtime.(*mheap).grow(0x7b2f40, 0x246139ca9)
/usr/lib/go-1.17/src/runtime/mheap.go:1393 +0x225
runtime.(*mheap).allocSpan(0x7b2f40, 0x246139ca9, 0x0, 0x1)
/usr/lib/go-1.17/src/runtime/mheap.go:1179 +0x165
runtime.(*mheap).alloc.func1()
/usr/lib/go-1.17/src/runtime/mheap.go:913 +0x69
runtime.systemstack()
/usr/lib/go-1.17/src/runtime/asm_amd64.s:383 +0x49

goroutine 18 [running]:
runtime.systemstack_switch()
/usr/lib/go-1.17/src/runtime/asm_amd64.s:350 fp=0xc00008ec98
 ↪  sp=0xc00008ec90 pc=0x461ee0
runtime.(*mheap).alloc(0x7f130a7d45d0, 0x8, 0x1, 0x0)
/usr/lib/go-1.17/src/runtime/mheap.go:907 +0x73 fp=0xc00008ece8
 ↪  sp=0xc00008ec98 pc=0x425213
runtime.(*mcache).allocLarge(0x53, 0x48c273950058, 0xb4, 0x1)
/usr/lib/go-1.17/src/runtime/mcache.go:227 +0x89 fp=0xc00008ed48
 ↪  sp=0xc00008ece8 pc=0x415e89
runtime.mallocgc(0x48c273950058, 0x0, 0x1)
/usr/lib/go-1.17/src/runtime/malloc.go:1082 +0x5c5 fp=0xc00008edc8
 ↪  sp=0xc00008ed48 pc=0x40cbc5
runtime.makechan(0x5, 0x9184e729fff)
/usr/lib/go-1.17/src/runtime/chan.go:101 +0x93 fp=0xc00008ee08
 ↪  sp=0xc00008edc8 pc=0x404fd3
github.com/consensys/gnark-crypto/ecc/bls12-381.partitionScalars(
 ↪  {0xc0001e0000, 0x3b5, 0x3b5}, 0x4, 0x0,
 ↪  0x9184e729fff)
```

Recommendation

We recomend the client to either perform input validation of the parameter `nbTasks` or limit the number of tasks to avoid a crash in a particular implementation using the library.

4 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

5 APPENDIX B: 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.

5.1 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

5.2 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

5.3 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 an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Assessment of the cryptographic primitives used
4. 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

5.4 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.

5.5 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.

5.6 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 TypeScript 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 TypeScript.

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.

6 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, Youssef El Housni, and Aurore Guillevic. 2022. A survey of elliptic curves for proof systems. Retrieved from https://eprint.iacr.org/2022/586

[2]
Razvan Barbulescu and Sylvain Duquesne. 2017. Updating key size estimations for pairings. DOI:https://doi.org/10.1007/s00145-018-9280-5

[3]
Jean-Luc Beuchat, Jorge Enrique González Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. 2010. High-speed software implementation of the optimal ate pairing over barreto-naehrig curves. Retrieved from https://eprint.iacr.org/2010/354

[4]
Aurore Guillevic and Shashank Singh. 2019. On the alpha value of polynomials in the tower number field sieve algorithm. Retrieved from https://eprint.iacr.org/2019/885

[5]
Koray Karabina. 2010. Squaring in cyclotomic subgroups. Retrieved from https://eprint.iacr.org/2010/542

[6]
Ethereum Improvement Proposals. 2017. EIP-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128. (2017). Retrieved from https://eips.ethereum.org/EIPS/eip-196

[7]
Joost Renes, Craig Costello, and Lejla Batina. 2015. Complete addition formulas for prime order elliptic curves. Retrieved from https://eprint.iacr.org/2015/1060

[8]
Karl Rubin and Alice Silverberg. 2008. Compression in finite fields and torus-based cryptography. *SIAM J. Comput.* 37, (January 2008), 1401–1428. DOI:https://doi.org/10.1137/060676155