Secure Code Review of avalabs-mpc

Ava Labs CGGMP21 Implementation

Ava Labs

March 15, 2024
Version: 1.2

Corporate Headquarters
Kudelski Security – Nagravision Sàrl
Route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland

DOCUMENT PROPERTIES

VERSION	1.2
FILE NAME	AvaLabs-CGGMP21_Code_Review_PUBLIC.pdf
PUBLICATION DATE	March 15, 2024
CONFIDENTIALITY LEVEL	For Public Release
DOCUMENT OWNER	Luca Dolfi
DOCUMENT AUTHORS	Luca Dolfi, Adina Nedelcu
DOCUMENT RECIPIENT	Arnold Yau
DOCUMENT STATUS	Final
CLIENT COMPANY NAME	Ava Labs

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

Ava Labs engaged Kudelski Security to perform a secure code assessment of a cryptographic library implementing the threshold signature scheme CGGMP21 [1] in Go. The code implements a threshold signature scheme for ECDSA over the curve secp256k1 with support for offline presigning. The library was commissioned by Ava Labs and implemented by Trail of Bits which worked closely with Kudelski Security Team during the engagement to provide documentation and support during the code audit.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place between the 27th of November 2023 and 24th of January 2024, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered with the cryptographic library.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

1.1. Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Missing domain separation in hashed message for key generation

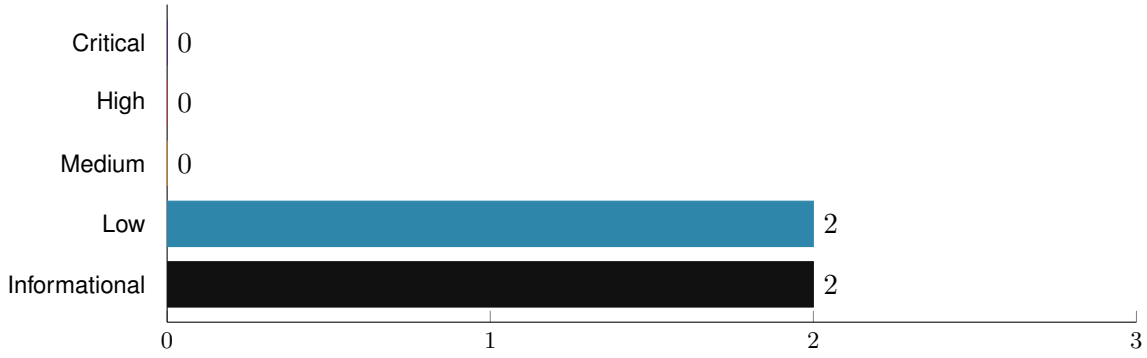

Figure 1.1. Findings ranked by severity.

[1] Canetti, R., et al (2020). *UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts*. Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. Available at: https://eprint.iacr.org/2021/060

2. PROJECT SUMMARY

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

2.1. Scope

The repository `avalabs-mpc` contains the implementation of the CGGMP21 threshold signature scheme. The library is meant to allow peers to join in a distributed protocol to sign messages, while no single user is able to control the signature individually. The code supports implementation over the curve secp256k1 (i.e. Bitcoin curve) for key generation, key refresh and reshare and offline presigning.

The intent of this engagement is to perform a secure code review of the following:

- `avalabs-mpc/crypto`
- `avalabs-mpc/cggmp`
- `avalabs-mpc/network`
- `avalabs-mpc/example`

Present at the commit hash `40f2c79c404e7a8d304f10442e287a02c4d701e8`

In addition, Ava Labs provided Kudelski Security Team with the detailed *CGGMP21 Specification* prepared by Trail of Bits on 14/07/2023 as assisting documentation to the implementation.

Follow-up

After the initial report, Ava Labs addressed the vulnerabilities and weaknesses in the following codebase revision:

- commit hash: `f6a8ae3e2a9d236df8e95d1c69708203a63e7a35`

A re-review was performed on February 23rd, 2024 and the status of the findings (Open, Acknowledged or Resolved) was updated in this report.

2.2. Remarks

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The developers have made a careful and in-depth analysis and documentation of their project.
- In their design decision, Trail of Bits were considerate on specific edge cases and how to handle them with a practical approach going from theory to production.

- Tests were also provided as part of the project, which is convenient for better understanding how the library works and useful for elaborating scenarios and validating findings.
- Finally, we had regular and very enriching technical exchanges on various topics.

2.3. 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.

While assessing 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 Section 5 of this document.

Deviations from CGGMP21 paper

The security of a cryptographic primitive relies heavily on the chosen security model. Composing cryptographic primitives, each individually proven secure within different security models, may introduce vulnerabilities in the composition, even if the individual primitives remain secure. As acknowledged by Trail of Bits in Section 2.5, the KeyGen protocol from the CGGMP21 paper has been substituted with the protocol introduced by Aumasson et al.[1] from Taurus Group. There exist additional slight deviations from the CGGMP paper, such as not implementing verifiable aborts and implementing key reshare.

As these are not fully captured by the security proof/model, it is advisable to perform a formal cryptographic analysis of the new protocol. However, it is important to note that such an analysis goes beyond the current engagement's scope.

[1] Adaptations from CGGMP21. J.P. Aumasson, A. Hamelink, L. Meier, https://github.com/taurusgroup/multi-party-sig/blob/main/docs/Threshold.pdf

3. FINDINGS

The Findings section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	SEVERITY	TITLE	STATUS
KS-AL-1	Low	SampleInterval does not Work Correctly for Certain Arguments	Resolved
KS-AL-2	Low	Missing SSID in Fiat Shamir Proof	Resolved
KS-AL-3	Informational	Paillier Security Parameter too Small for 128 Bits Security	Informational
KS-AL-4	Informational	Missing Edge Cases Checks	Informational

3.1. KS-AL–1 SampleInterval does not Work Correctly for Certain Arguments

Overall Severity:	LOW
Status:	Resolved

Impact	Likelihood
LOW	LOW

Description

The file `interval.go` contains the function `SampleInterval`. that should return "random value in the range $+/-2^{bits}$". During testing, the Kudelski Security Team has discovered that this function does not work as intented for certain values of the `bits` argument. In particular:

- If $bits < 8$, then the `buf[0 : size-1]` is the empty array and thus the sampled number is zero.
- It $bits$ is not a multiple of 8, then up to 7 bits are discarded and the sampled number will belong to a smaller interval than intended.

Impact

This function is never used with arguments that will cause the incorrect behaviour. However, this could occur in the future; for example, if a developer intends to use the `crypto` as a basis for another project.

Evidence

```
10 // SampleInterval returns a random value in the range +/- 2^bits
11 func SampleInterval(bits uint) *saferith.Int {
12   size := bits/8 + 1
13   buf := make([]byte, size)
14   _, err := rand.Read(buf[:])
15   if err != nil {
16     panic(err)
17   }
18
19   nat := new(saferith.Nat).SetBytes(buf[0 : size-1])
20
21   return new(saferith.Int).SetNat(nat).Neg(saferith.Choice(buf[size
      -1]) & 1)
22 }
```

...interval.go

Proof of Concept

```
package main

import (
  "crypto/rand"
  "fmt"
  "github.com/cronokirby/saferith"
)
func SampleInterval2(bits uint) *saferith.Int {
  size := bits/8 + 1
  fmt.Println("Queried Size: ",size)
  buf := make([]byte, size)
  _, err := rand.Read(buf[:])
  if err != nil {
    panic(err)
  }
  fmt.Println("Sampled Buf: ", buf)

  nat := new(saferith.Nat).SetBytes(buf[0 : size-1])

  fmt.Println("Bytes of nat: ", buf[0 : size-1])
  fmt.Println("Sign : ",saferith.Choice(buf[size-1]) & 1)

  ret := new(saferith.Int).SetNat(nat).Neg(saferith.Choice(buf[size
   -1]) & 1)
  return ret
}



func SampleInterval3(bits uint) *saferith.Int {
  size := bits/8 + 1
  fmt.Println("Queried Size: ", size)
  buf := make([]byte, size)
  _, err := rand.Read(buf[:])
  if err != nil {
    panic(err)
  }
  fmt.Println("Sampled Buf: ", buf)

  nat := new(saferith.Nat).SetBytes(buf[0 : size-1])

  fmt.Println("Bytes of nat: ", buf[0 : size-1])
  fmt.Println("Sign : ",saferith.Choice(buf[size-1]) & 1)

  ret := new(saferith.Int).SetNat(nat).Neg(saferith.Choice(buf[size
   -1]) & 1)
  return ret
}
```

```
47
48
49 func main() {
50   fmt.Println("PoC for bits < 8.")
51   fmt.Println("Executing SampleInterval2(1).")
52   SampleInterval2(1)
53   fmt.Println("PoC for bits = not multiple of 8.")
54   fmt.Println("Executing SampleInterval3(15).")
55   SampleInterval3(15)
56   fmt.Println("End of program.")
57 }
```

Code to test the function TestInterval and print out relevant values.

```
sh-3.2$ go run .
PoC for bits < 8.
Executing SampleInterval2(1).
Queried Size:  1
Sampled Buf:  [5]
Bytes of nat:  []
Sign :  1
PoC for bits = not multiple of 8.
Executing SampleInterval3(15).
Queried Size:  2
Sampled Buf:  [16 208]
Bytes of nat:  [16]
Sign :  0
End of program.
```

Console output. With argument 1...7, `nat` will be an empty buffer. With a non-multiple of 8 argument, the last byte of `buf` will not be included into `nat`.

Affected Resources

- `crypto/random/interval.go` line 10-22

Recommendation

Correct the behaviour of the function for the edge cases outlined. Alternatively, perform input validation and only accept non-zero multiples of 8 as arguments.

References

[1] CWE-20: Improper Input Validation

3.2. KS-AL–2 Missing SSID in Fiat Shamir Proof

Overall Severity:	LOW
Status:	Resolved

Impact	Likelihood
MEDIUM	LOW

Description

According to the technical specification [1] provided, `aux` should be "a byte string containing binding data, such as party and session ID". Furthermore, according to Section 8.6, Step 9a in `AFF_PROVE`, this should be included when computing the challenge (as part of the Fiat-Shamir transformation.)

Impact

The session identifier and the other parameters included in `aux` should be included in the challenge computation to prevent replays across different sessions.

Evidence

```
100   fiatshamir := fsAffg{
101     Prm:  prm,
102     N0:   k.PublicKey().N().Nat(),
103     N1:   f.PublicKey().N().Nat(),
104     C:    k.Nat(),
105     D:    d.Nat(),
106     Y:    f.Nat(),
107     BigX: bigX,
108     S:    S,
109     T:    T,
110     A:    A,
111     Bx:   bx,
112     By:   by,
113     E:    E,
114     F:    F,
115   }
116
117   // compute challenge
118   e, err := challenge(fiatshamir)
```

`...aff.go` The function `challenge` is called with only one parameter.

```
283   func challenge(fs fsAffg, aux ...any) (*saferith.Int, error) {
284     return util.NewChallenge(common.L, "PI_AFF-G", aux, fs.Prm, fs.
      N0, fs.N1, fs.C, fs.D, fs.Y, fs.BigX, fs.S, fs.T, fs.A, fs.Bx,
      fs.By, fs.E, fs.F)
```

```
285     }
```

...aff.go. Definifion of the challenge function.

Affected Resources

- crypto/zkproof/paillier/aff.go lines 100-118

Recommendation

Call the challenge function with the aux parameter included.

References

[1] Trail of Bits (14/07/2023). *CGGMP21 Specification*.

3.3. KS-AL–3 Paillier Security Parameter too Small for 128 Bits Security

Overall Severity: INFORMATIONAL

Description

Although the CGGMP21 [1] paper says that the size of the Paillier modulus should be at least 2048 bits when working on a 256-bit elliptic curve, following NIST recommendations [2] to match the desired security level of 128 bits the Paillier modulus should have a size of 3072 bits.

Remark

After discussion with Ava Labs and Trail of Bits, we include their answer on the finding:

> *Bit-equivalent security for RSA moduli is fundamentally heuristic. While we believe that 2048-bit moduli are sufficient for all practical purposes, we have added documentation describing how users can increase this parameter if desired.*

Impact

The current security level of Paillier corresponds to 112 bits, which is lower than the security level of secp256k1 (128 bits). The use of 2048 bits is a common choice if a security level of 112 bits is considered sufficient or if switching to bigger size is not possible for performance reasons.

Evidence

```
7  // SecParam is the bit-length kappa of an EC curve element. Because
       this
8  // library implements only secp256k1, this is obviously 256.
9  SecParam = 256
10 SecBytes = SecParam / 8
11
12 // According to the CGGMP paper, l, l', eps are set to 1, 5, 2
       factor of the
13 // EC element bit-length SecParam.
14 L       = 1 * SecParam
15 LPrime  = 5 * SecParam
16 Epsilon = 2 * SecParam
17 )
18
19 var PaillierBits = 8 * SecParam
```

...params.go.

Affected Resources

- `cggmp/common/params.go` line 19

Recommendation

To achieve the NIST recommended level of 128 bits of security, the value of the `PaillierBits` global variable should be increased to at least 3072.

References

[1] Canetti, R., et al (2020). *UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts*. Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. Available at: https://eprint.iacr.org/2021/060, page 5

[2] (NIST, 2020) *NIST Special Publication 800-57 Part 1 Revision 5, Recommendation for Key Management: Part 1 – General*, page 54-55 Available at: https://nvlpubs.nist.gov

3.4. KS-AL–4 Missing Edge Cases Checks

Overall Severity: INFORMATIONAL

Description

Either the paper [1] or the technical specification mention some edge cases that should be checked for, such as:

- sampled scalar should not be zero;
- primes p and q should be distinct from each other;
- elements used in Paillier should be invertible.

These are extremely unlikely to happen given the size of the security parameters implemented, however it is good practice to include these sanity checks to avoid undefined behaviours.

Remark

After discussion with Ava Labs and Trail of Bits, it was made clear that these checks were intentionally omitted due to the negligible probability of the adverse events occurring:

> *The probability of any of these events occurring is less than the probability that an attacker successfully guesses the user's private key at random. We determined that inclusion of these checks would create unnecessary side-channel attack surface in order to prevent a cryptographically-negligible occurrence.*

Impact

Mathematical operations with edge cases might not work as intended, leading to undefined behaviour.

Evidence

Invertibility Check

Several places in [2], such as 8.2, Step 1b in `PRM_GEN`, describe a check for invertibility, checking that $gcd(r, N) = 1$. While these are laid out in the specification, they are not performed in the codebase.

Primes should be distinct

```
74  func GenerateKey(size int) SecretKey {
75  if size&1 != 0 {
76    panic("internal␣error:␣requested␣Paillier␣key␣size␣is␣not␣even"
     )
77  }
78
79  concurrency := runtime.NumCPU()
80  // Buffered so that workers don't block on output
81  outChan := make(chan *saferith.Nat, concurrency)
82  cancelChan := make(chan struct{})
```

```
83    for i := 0; i < concurrency; i++ {
84      go func() {
85        // Initialize a new thread-local fast RNG for each worker
86        seed := random.Bytes32()
87        reader, err := blake2b.NewXOF(blake2b.OutputLengthUnknown,
      seed[:])
88        if err != nil {
89          panic("internal␣error:␣failed␣initializing␣fast␣RNG")
90        }
91        outChan <- random.SafePrime(size/2, reader, cancelChan)
92      }()
93    }
94    p := <-outChan
95    q := <-outChan
96    if p == nil || q == nil {
97      panic("internal␣error:␣failed␣generating␣safe␣primes")
98    }
99    close(cancelChan) // Cancel remaining workers
100   return SecretKeyFromFactors(p, q, size)
101 }
```

`...secret.go` There is no check to determine whether the sampled primes are distinct frome each other.

Affected Resources

- `crypto/paillier/secret.go` line 74-101
- `crypto/paillier/public.go` line 45-62

Recommendation

Though these sanity checks are highly improbable and cryptographically negligible, omitting them from the code technically constitutes a (negligible) deviation from the specifications outlined in the paper. Following discussions with Ava Labs and the developers, the Kudelski Security Team also recognized that certain potential remediation measures, particularly those involving GCD checks, could inadvertently introduce a new and potentially more susceptible vulnerability to side-channel attacks.

References

[1] Canetti, R., et al (2020). *UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts*. Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. Available at: https://eprint.iacr.org/2021/060

[2] Trail of Bits (14/07/2023). *CGGMP21 Specification*.

4. 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.

4.1. 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

4.2. 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

4.3. 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.

4.3.1. Code Review

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

1. general code safety and susceptibility to known issues;
2. poor coding practices and unsafe behavior;
3. leakage of secrets or other sensitive data through memory mismanagement;
4. susceptibility to misuse and system errors;
5. 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.

4.3.2. Cryptography

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

1. matching of the proper cryptographic primitives to the desired cryptographic functionality needed;
2. security level of cryptographic primitives and their respective parameters (key lengths, etc.);
3. safety of the randomness generation in general as well as in the case of failure;
4. safety of key management;
5. assessment of proper security definitions and compliance to use cases;
6. checking for known vulnerabilities in the primitives used.

4.3.3. Technical Specification Matching

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

1. proper implementation of the documented protocol phases.
2. proper error handling.
3. adherence to the protocol logical description.

4.4. Reporting

Kudelski delivered to Ava Labs 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:

1. **Critical**;
2. **High**;
3. **Medium**;
4. **Low**;
5. **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. VULNERABILITY SCORING SYSTEM

Kudelski Security utilizes a custom approach when computing the vulnerability score, based primarily on the **Impact** of the vulnerability and **Likelihood** of an attack.

Each metric is assigned a ranking of either low, medium or high, based on the criteria defined in in Section 5.2 and Section 5.3. The overall severity score is then computed as described in the next section.

5.1. Severity

Severity is the overall score of the finding, weakness or vulnerability as computed from Impact and Likelihood. Other factors, such as availability of tools and exploits, number of instances of the vulnerability and ease of exploitation might also be taken into account when computing the final severity score.

IMPACT \ LIKELIHOOD	LOW	MEDIUM	HIGH
HIGH	MEDIUM	HIGH	HIGH
MEDIUM	LOW	MEDIUM	HIGH
LOW	LOW	LOW	MEDIUM

Table 5.1. How to compute overall Severity from Impact and Likelihood. The final severity factor might vary depending on a project's specific context and risk factors.

- **Critical** The identified issue may be immediately exploitable, causing a strong and major negative impact system-wide. They should be urgently remediated or mitigated.
- **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. Informational findings are not assigned a severity score and are classified as "Informational" instead.

5.2. Impact

The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

- **High** The vulnerability has a severe effect on the company and systems or has an affect within one of the primary areas of concern noted by the client.
- **Medium** It is reasonable to assume that the vulnerability would have a measurable affect on the company and systems that may cause minor financial or reputational damage.
- **Low** There is little to no affect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

5.3. Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, difficulty of exploitation and institutional knowledge.

- **High** It is extremely likely that this vulnerability will be discovered and abused.
- **Medium** It is likely that this vulnerability will be discovered and abused by a skilled attacker.
- **Low** It is unlikely that this vulnerability will be discovered or abused when discovered.

6. CONCLUSION

The objective of this secure code audit was to evaluate whether there were any vulnerabilities that would put Ava Labs or users of the `avalabs-mpc` library at risk.

The Kudelski Security Team identified 2 security issues: 0 critical risks, 0 major risks, 0 medium risks and 2 lower risk. On average, the effort needed to mitigate these risks is estimated as low.

In order to mitigate the risks posed by this engagement's findings, the Kudelski Security Team recommends applying the following best practices:

1. Apply domain separation for recipients and dealers IDs in the keygeneration.
2. Pay particular attention to the reuse of presignatures.

Kudelski Security remains at your disposal should you have any questions or need further assistance.

Kudelski Security would like to thank Ava Labs for their trust, help and support over the course of this engagement and is looking forward to cooperating in the future.

RECIPIENT CONTACTS

NAME	POSITION	CONTACT INFORMATION
Arnold Yau	Security Lead	arnold@avalabs.com

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Jean-Sébastien Nahon	Application and Blockchain Security Practice Manager	jean-sebastien.nahon@kudelskisecurity.com
Alex Kopferschmitt	Sales Manager - Blockchain Security	alex.kopferschmitt@kudelskisecurity.com
Luca Dolfi	Security Engineer	luca.dolfi@kudelskisecurity.com

DOCUMENT HISTORY

VERSION	DATE	AUTHOR	COMMENT
1.0	24.01.2024	Luca Dolfi	First draft
1.1	23.02.2024	Luca Dolfi	Final report
1.2	15.03.2024	Luca Dolfi	Document prepared for public release