ZK Oracle Secure Code Review

Technical Report

ZKON

10 December 2024
Version: 1.1

Kudelski Security – Nagravision Sàrl

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

DOCUMENT PROPERTIES

Version:	1.1
File Name:	Kudelski_Security_Zkon_CR_v1.1.pdf
Publication Date:	10 December 2024
Confidentiality Level:	Confidential
Document Owner:	Luca Dolfi
Document Authors:	Luca Dolfi, Joo Yeon Cho
Document Recipients:	Paul Foix, Juanjo Chust, Thomas Kevers
Document Status:	Proposal

EXECUTIVE SUMMARY

ZKON (“the Client”) engaged Kudelski Security (“Kudelski”, “We”) to perform the ZK Oracle Secure Code Review.

The assessment was conducted remotely by the Kudelski Security Team.

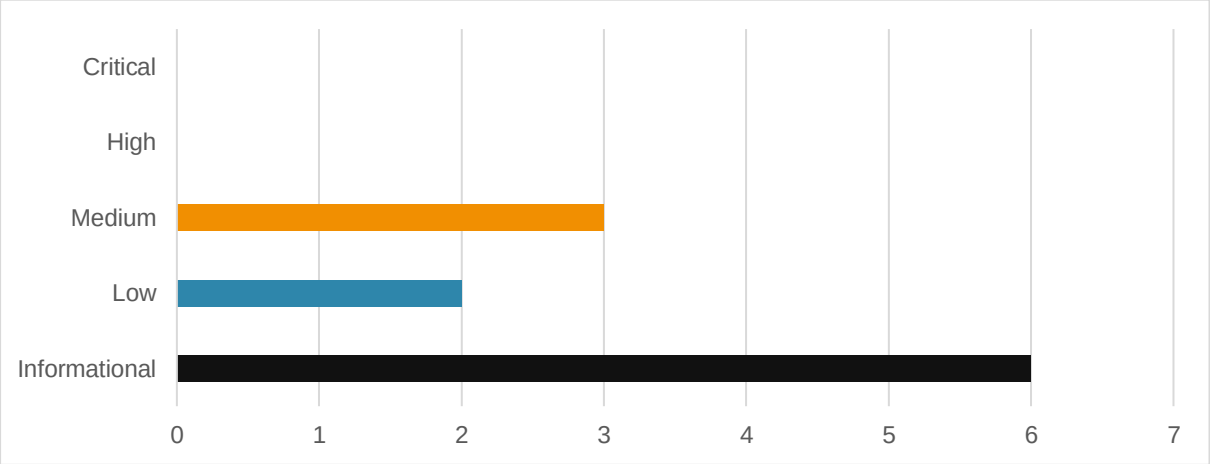
The review took place between 23rd October 2024 and 31th October 2024, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered.
- 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.

Key Findings

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

- TLS private key is imported from a static file.
- Prover certificate is imported without validation.
- VerifySignedHashV2 does not validate messageHash.



Findings ranked by severity

1. 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.

1.1 Context

The ZKON `Mina zkApp` repository contains a set of typescript files to host the zkApps that drive the ZKON Oracle ecosystem on the Mina blockchain. The ZKON `Proof Client` repository contains a set of rust files to run a proof client server over the TLS notary server.

1.2 Scope

The scope consisted in specific rust and typescript files and folders located at:

- zkApp repo: https://github.com/ZKON-Network/zkTLS-Mina-zkApp/
  - Commit hash: `65c875ce000b620e76565f5cf39a5265fe24ab3a`
- Proof client repo: https://github.com/ZKON-Network/zkTLS-Mina-zkApp/
  - Commit hash: `5c8b17e6670a91d7f98db6f3cb63613e60348f04`

The folders in scope are:

```
zkTLS-Mina-zkApp/src
    ├── String.ts
    ├── UInt8.ts
    ├── ZkonRequest.ts
    ├── ZkonRequestCoordinator.ts
    ├── index.ts
    └── zkProgram.ts
```

```
proof-client/src/
    ├── data_models.rs
    ├── main.rs
    ├── proof_client.rs
    ├── proof_service_check.rs
    └── route_handler.rs
```

The client's requirements written in RFP were as follows:

- For the audit, the client expects research, investigation, and review of the ZKON zkApps delivery and integration with Mina followed by issue reporting, along with mitigation and remediation instructions and a final report.
- The client would like a review of ZKON's work on ECDSA verification - ZkProgram now works with provable code, and verifies ECDSA Signature on curve Secp256k1/r1.

The goal of the evaluation was to perform a security audit on the source code.

- No additional systems or resources were in scope for this assessment.
- The dependencies are out of scope of the review.
- Test codes are out of scope.

- Secp256r1 is not supported yet in the code, so is not in scope.

1.3 Remarks

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

- The code is well structured.
- Quick and open communication via Telegram
- The developers have made a careful and in-depth analysis of their project.
- We had regular and enriching technical exchanges on various topics.

1.4 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 Chapter Vulnerability Scoring System of this document.

1.5 Follow-up

After the initial report (V1.0) was delivered, the Client addressed all vulnerabilities and weaknesses in the following codebase revision:

- zkApp repo
  - ZkProgram Improvement

    (commit `1a6bcef91b72ff78ab4905b1c52bca4fafd30dba`)
  - feat: Add nonce to the request id hash

    (commit `62b8cd02fc7c667f5cc235a6a389ed978c40369c`)
  - feat: Pad bits array to ensure length in string

    (commit `e0a8b982c7a791e2fa97fac8b8dd61f1eca8ffe3`)
- Proof client repo
  - Bump-up dependencies

    (commit `99e8db75292a0a51f7a8d08bcd599d75b6d8ba3d`)
  - Implement audit changes

    (commit `bbd217aa6ec821ef6804139a79ccf573de66ca2a`)

2. STATIC CODE ANALYSIS

2.1 Cargo audit

`Cargo audit` (v0.20.0) identified 5 vulnerabilities and 9 allowed warnings in the proof-client repo. Among those, 5 vulnerabilites are reported in Chapter 3.

2.2 Cargo clippy

`Cargo clippy` (v0.1.77) identified multiple warnings in the proof-client repo. The output of `cargo clippy` is in the Appendix.

2.3 Cargo miri test

No test is defined in the proof-client repo.

2.4 Semgrep

`Semgrep` (v1.85.0) identified one high risk vulnerability in proof-client and two low risk vulnerabilities in zkApp. They are reported in Chapter 3.

2.5 codeQL

`codeQL` (v2.16.6) identified 2 recommendations in the zkApp/src. They are reported in Chapter 4.

2.6 Typescript-eslint

`typescript-eslint` (v2.16.6) identified 6 findings in the zkApp/src. They are reported in Chapter 4.

2.7 Snyk

`snyk-security` (v1.1293.0) identified one medium security risk in proof-client and three warnings for dependencies in zkApp.

3. TECHNICAL DETAILS OF SECURITY FINDINGS

This chapter 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 security findings.

#	SEVERITY	TITLE	STATUS
KS-ZKO-F-01	Medium	Improper Handling of TLS KeyImproper Handling of TLS Key	Resolved
KS-ZKO-F-02	Medium	Client Certificate Not Validated	Resolved
KS-ZKO-F-03	Medium	VerifySignedHashV2 Input Not Validated	Resolved
KS-ZKO-F-04	Low	Field Conversion Not Accurate	Resolved
KS-ZKO-F-05	Low	Outdated Dependencies	Resolved

Findings overview.

3.1 KS–ZKO-F-01 Improper Handling of TLS Key

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

In main.rs for proof-client the TLS keys are loaded from static files in the repository. The private key is stored in cleartext and accessible with the source code. The Client assured Kudelski Security's team that the key present in the repository is a dummy key, not used for anything in production nor testing. However, there are some considerations for keeping the key accessible to the program but safe.

Moreover, hardcoding file paths directly in source code can make them more difficult to manage securely across different environments.

Impact

The private key stored in clear text (and possibly with same read privileges as the source code) could be read by any user or program on the same machine. An adversary can access it directly or exploit the proof-client to leak the contents of `key.rsa`

Evidence

```
    // CONFIGURE TLS CERT
let cert_path = path.to_string() + "/tls/cert.pem";
let key_path = path.to_string() + "/tls/key.rsa";
```

`proof-client/src/main.rs.rs`. Defines static paths to private and public keys.

Affected Resources

- `proof-client/src/main.rs` lines 41-42

Recommendation

Make sure the key.rsa file and any other file containing secrets is ignored by git and never pushed to the repository. If possible, rely on environmental variables or configuration management tools to pass secrets (or file paths) to the application. Otherwise, restrict file access to the server process only. For instance, using `chmod 600` and avoid world-readable permissions. Finaly, regularly rotate keys and certificates, especially if there is any chance of exposure.

3.2 KS-ZKO-F-02 Client Certificate Not Validated

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

In `main.rs`, the client certificate is imported to serve the TLS between the prover and the verifier. However, a basic validation on the certificate is not performed before it is served for TLS. For example, the current dummy certificate `/tls/cert.pem` is already expired as below.

```
$ openssl x509 -in tls/cert.pem -text -noout
       Validity
            Not Before: Aug 13 16:07:04 2016 GMT
            Not After : Feb  3 16:07:04 2022 GMT
```

However, the server gets online with this certificate and a secure negotiation is supported. See the Appendix.

Impact

If a corrupted certificate is imported, the TLS handshake data could be leaked to the adversary.

Evidence

```
   // CONFIGURE TLS CERT
    let cert_path = path.to_string() + "/tls/cert.pem";
    let key_path = path.to_string() + "/tls/key.rsa";

    println!("Server Online.");

    warp::serve(serve)
    .tls()
    .cert_path(cert_path.to_string())
    .key_path(key_path.to_string())
    .run(server_address).await;
```

Location: zkon/proof-client/src/main.rs

Affected Resources

- `zkon/proof-client/`

Recommendation

The client certificate can be validated before usage. The client checks to ensure that the certificate is not expired and that the domain name or IP address on the certificate matches the client's information.

3.3 KS-ZKO-F-03 VerifySignedHashV2 Input Not Validated

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

The `verifySignedHashV2` function does not validate the input parameter `messageHash`. According to the in-line comment in the code, the `verifySignedHashV2` function is a building block of the `verifyV2` function and takes the message hash (a curve scalar) as input, giving you flexibility in choosing the hashing algorithm. Hence, `verifySignedHashV2` itself is not a complete function. This may open up an opportunity that an adversary could use a hash algorithm with truncation (i.e. a short size of hash output) to get a message collision easily and obtain a fake signature.

Suppose it is feasible to create two messages whose hashes are identical. Then, the attack scenario is as follows. The adversary hashes one message and gets it signed by the `signHash` function. Then, he submits the other message for verification with its hashed message and the signature. Since `messageHash` is same as before, the signature is verified without an error.

Impact

If the attack scenario works, then a fake message can be accepted as a valid one.

Evidence

```
const ZkonZkProgram = ZkProgram({
...
          decommitment.assertEquals(commitment.commitment,"Response from proof-
server invalid.");
          return ECDSASign.signature.verifySignedHashV2(
            ECDSASign.messageHash, ECDSASign.publicKey)
        }
```

Location: /zkTLS-Mina-zkApp/src/zkProgram.ts, line 32

Affected Resources

- zkon/zkTLS-Mina-zkApp

Recommendation

Validate `messageHash` to ensure it is not a trivial value or a short size of hash. Consider using the `verifyV2` function for a complete verification.

References

None

3.4 KS-ZKO-F-04 Field Conversion Not Accurate

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

In the `fromField` method, the input `field` is first converted to bits and grouped into bytes. However, the length of bytes is calculated by dividing the length of bits by 8. In this case, only its quotient is taken, and the remainder is discarded. Since `field` is an element of a prime order finite field, the length of `field` in bits could be not a multiple of 8, which may lead to an incorrect type conversion.

Impact

If the left-over bits are discarded, multiple field elements map to a single value, which may lead to a false acceptance.

Evidence

```
    static fromField(field: Field): StringCircuitValue {
        let values: UInt8[] = [];

        // Convert field to bits
        const bits = field.toBits();

        // We are dealing with bytes (8 bits at a time), so group the bits into
bytes
        const byteArray = new Uint8Array(bits.length / 8);

        // Extract each byte from the bits array
        for (let i = 0; i < byteArray.length; i++) {
            let byteValue = 0;
            for (let bitIndex = 0; bitIndex < 8; bitIndex++) {
                if (bits[i * 8 + bitIndex].toBoolean()) {
                    byteValue |= 1 << bitIndex;
                }
            }
            byteArray[i] = byteValue;
        }
```

Location: zkon/zkTLS-Mina-zkApp/src/String.ts

Affected Resources

- `zkon/zkTLS-Mina-zkApp/`

Recommendation

ZKON commented that “we use that `bits.length/8` to convert the bits into bytes, if there are some “left over” bits it’s true that doesn’t get included. We did a few tests converting it from the “toField” method back to the “fromField” method and we never lose any bits of info.” However, it recommend to remove any potential risk which may be explored by adversaries for malicious purpose.

3.5 KS-ZKO-F-05 Outdated Dependencies

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The `cargo audit` tool identified that the following dependencies include vulnerabilities:

- openssl: RUSTSEC-2024-0357, Vulnerability, Upgrade to >=0.10.66
- rust-crypto: RUSTSEC-2022-0011, Vulnerability, No fixed update is available
- rustc-serialize: RUSTSEC-2022-0004, Vulnerability, No fixed upgrade is available
- time: RUSTSEC-2020-0071, Upgrade to >=0.2.23
- tungstenite: RUSTSEC-2023-0065, Upgrade to >=0.20.1

Additionally, `semgrep` found that the atty dependency has a Low risk vulnerability. Moreover, no patch seems available and the library seem unmantained.

Impact

Outdated dependencies might create security vulnerabilities or undesired errors in the application.

Evidence

```
name = "openssl"
version = "0.10.64"
name = "rust-crypto"
version = "0.2.36"
name = "rustc-serialize"
version = "0.3.25"
name = "time"
version = "0.1.45"
name = "tungstenite"
version = "0.16.0"
```

Location: /zkon/proof-client/Cargo.lock

Affected Resources

- zkon/proof-client

Recommendation

Keep external dependencies up to date to avoid any security breach.

References

- RUSTSEC-2024-0357
- RUSTSEC-2022-0011
- RUSTSEC-2022-0004
- RUSTSEC-2020-0071
- RUSTSEC-2023-0065

4. OBSERVATIONS

This chapter 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.

#	SEVERITY	TITLE	STATUS
KS-ZKO-O-01	Informational	Test Coverage Not Sufficient Coverage Not sufficient	Informational
KS-ZKO-O-02	Informational	Best Secure Code Practices	Informational
KS-ZKO-O-03	Informational	Use of ECDSA with Secp256k1 Not Justified Well	Informational
KS-ZKO-O-04	Informational	Risk of AES CBC Mode in TLS 1.2	Informational
KS-ZKO-O-05	Informational	Risk of MAC-then-Encrypt	Informational
KS-ZKO-O-06	Informational	Input Parameter Not Validated	Informational

Observations overview.

4.1 KS-ZKO-O-01 Test Coverage Not Sufficient

Description

- For proof-client repo, no test vector is defined.
- For zkApp repo, the `npm run coverage` tool (v9.5.1) shows that the overall test coverage of code in scope reaches less than the full coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
String.ts	58.06	50	41.17	57.14	11,18,22-27,35-39,88-108
UInt8.ts	24	33.33	23.07	27.27	6,10-31,39-52,60
ZkonRequest.ts	80	100	75	77.77	54-57
ZkonRequestCoordinator.ts	92	100	88.88	91.11	128-133
zkProgram.ts	33.33	100	0	33.33	31-32
Total	65.62	45.45	47.72	65.69	

- For zkApp repo, the `npm run test` tool (v9.5.1) shows some test failures as below.

```
$ npm run test
...
Test Suites: 2 failed, 1 passed, 3 total
Tests:       3 failed, 7 passed, 10 total
```

Affected Resources

- `zkon/proof-client`
- `zkon/zkTLS-Mina-zkApp`

Recommendation

Add more test vectors and increase the test coverage rate.

4.2 KS-ZKO-O-02 Best Secure Code Practices

Description

For `zkApp` repo, the `codeQL` tool (v2.16.6) identified the following 2 recommendations.

```
"Unused variable, import, function or class","Unused variables, imports, functions or classes may be a symptom of a bug and should be examined carefully.","recommendation","Unused imports Proof, assert.","/src/ZkonRequestCoordinator.ts","1","1","1","149"

"Unused variable, import, function or class","Unused variables, imports, functions or classes may be a symptom of a bug and should be examined carefully.","recommendation","Unused import Mina.","/src/zkProgram.ts","1","1","1","103"
```

For `zkApp` repo, the `typescript-eslint` tool (v8.57.1) identified the following 6 findings.

```
/zkTLS-Mina-zkApp/src/String.ts

  57:13  error  'values' is never reassigned. Use 'const' instead  prefer-const

zkon/zkTLS-Mina-zkApp/src/ZkonRequest.ts

  10:12  error  'ZkonProof_' is never reassigned. Use 'const' instead  prefer-const

/zkTLS-Mina-zkApp/src/ZkonRequestCoordinator.ts

     1:91  error  'assert' is defined but never used  @typescript-eslint/no-unused-vars

     1:99  error  'Proof' is defined but never used  @typescript-eslint/no-unused-vars

    32:12  error  'ZkonProof_' is never reassigned. Use 'const' instead  prefer-const

/zkTLS-Mina-zkApp/src/zkProgram.ts

    1:10  error  'Mina' is defined but never used  @typescript-eslint/no-unused-vars
```

In the `zkApp` repo, it is recommended to add an error message when `assertEquals` is used.

```
      currentOwner.assertEquals(this.sender.getAndRequireSignatureV2());

      caller.assertEquals(this.oracle.getAndRequireEquals());

      decommitment.assertEquals(commitment.commitment,"Response from proof-server invalid.");
```

In the `zkApp` repo, the following code snippet is missing in the coding section of `README.md`, line `87` and `110`.

```
      const coordinatorAddress = this.coordinator.getAndRequireEquals();
```

Affected Resources

- `zkon/zkTLS-Mina-zkApp/`

Recommendation

Consider updating the code and removing the lint errors.

4.3 KS-ZKO-O-03 Use of ECDSA with Secp256k1 Not Justified Well

Description

The use of ECDSA with secp256k1 is commonly associated with Bitcoin and Ethereum. If this is not the case, Ed25519, i.e., EdDSA with curve25519, offers improved performance and security features in general (see Reference).

In the whitepaper [1], the use of the secp256k1 curve ensures compatibility with widely used cryptographic standards, which is not very convincing why ECDSA with secp256k1 is chosen unless there is a Bitcoin or Ethereum interoperability requirement.

```
class Secp256k1 extends createForeignCurveV2(Crypto.CurveParams.Secp256k1) {}
class Scalar extends Secp256k1.Scalar {}
class Ecdsa extends createEcdsaV2(Secp256k1) {}

class ECDSAHelper extends Struct({
  messageHash: Scalar,
  signature: Ecdsa,
  publicKey: Secp256k1
}){}
```

Location: zkon/zkTLS-Mina-zkApp/src/zkProgram.ts

Affected Resources

- `zkon/zkTLS-Mina-zkApp/`

Recommendation

As commented in the Mina documentation, “ECDSA is used in many blockchains, including Ethereum, to sign transactions. ECDSA works with different elliptic curves. Bitcoin and Ethereum both use the secp256k1 curve.” If zkApp would support only Bitcoin or Etheruem, it needs to be clearly stated on the code or written in the white paper.

Reference

- SafeCurves: https://safecurves.cr.yp.to/

4.4 KS-ZKO-O-04 Risk of AES CBC Mode in TLS 1.2

Description

In the whitepaper [1], ZKON supports commonly used cipher suites in TLS 1.2, such as AES CBC HMAC and AES GCM. However, AES CBC mode has been removed from TLS 1.3 due to both implementation challenges and inherent design weaknesses (making it impossible to implement with both high performance and high security). The implementation of AES CBC mode is vulnerable to multiple attacks, including Lucky13 and BEAST.

In TLS 1.2, the HMAC authentication tag is created over the plaintext before encryption (MAC-then-encrypt), making it vulnerable to CBC padding oracle attacks. The CBC mode requires an unpredictable IV, and attacks like BEAST have demonstrated how predictable IVs can be exploited. As noted in TLS 1.2 RFC, a timing attack on the CBC padding is demonstrated based on the time required to compute the MAC.

Affected Resources

- `zkon/zkTLS-Mina-zkApp/`

Recommendation

Support only the AES GCM mode if possible. If AES CBC is required, then implement the CBC mode carefully as guided here to avoid any security breach. The default implementation of the AES CBC mode is insecure.

References

- Using AES-CBC in TLS1.2
- Cryptopals: Exploiting CBC Padding Oracles
- Password Interception in a SSL/TLS Channel

4.5 KS-ZKO-O-05 Risk of MAC-then-Encrypt

Description

In the whitepaper [1], ZKON introduced selective opening techniques that allow to reveal or redact specific parts of the plaintext. This is achieved by leveraging the MAC-then-encrypt structure of TLS and constructing proofs that focus on specific parts of the data.

Block ciphers require the length of plaintext messages to be a multiple of the cipher's block size (typically 16 bytes for AES). To achieve this, padding schemes like PKCS#7 are used to extend the message length. During decryption, this padding must be validated. However, depending on the mode of operation and the padding scheme, this could lead to padding oracle attacks. In these attacks, an attacker manipulates the ciphertext and observes whether the padding validation succeeds or fails, either through error messages or timing differences. This could eventually lead to full plaintext recovery.

In TLS versions up to 1.2, the use of MAC-then-encrypt meant that padding validation occurred after MAC verification, making the protocol vulnerable to sophisticated timing

attacks like BEAST and Lucky 13. TLS 1.3 addresses this by mandating AEAD cipher suites, which combine authentication and encryption in a way that prevents these types of attacks.

Affected Resources

- `zkon/zkTLS-Mina-zkApp/`

Recommendation

Use the encrypt-then-MAC structure, which is to encrypt the data first, then compute the MAC of the ciphertext. This leads to a situation where the receiving endpoint checks the MAC first, before performing decryption, and drops the connection if the MAC is incorrect. Since the attacker can't forge the MAC without knowing the session key, this completely negates the padding oracle attack.

References

- The Transport Layer Security (TLS) Protocol Version 1.2

4.6 KS-ZKO-O-06 Input Parameter Not Validated

Description

In the method prepayRequest, the input parameter `requestAmount` is not validated, which allows a user to send a request with zero amount. This could be used for a malicious purpose, i.e., an adversary could repeat requests over and over without paying any fee, which may lead to an ddos-type attack.

```
  async prepayRequest(requestAmount: UInt64, beneficiary: PublicKey) {


    const ZkToken = new FungibleToken(this.zkonToken.getAndRequireEquals());


    const feePrice = this.feePrice.getAndRequireEquals();
    const totalAmount = feePrice.mul(requestAmount);


    await ZkToken.transfer(this.sender.getAndRequireSignatureV2(),
this.owner.getAndRequireEquals(), totalAmount);


    //Get the Blockchain length
    const createdAt = this.self.network.blockchainLength


    const event = new RequestPaidEvent({
      zkApp: beneficiary,
      requestsPaid: requestAmount.toFields()[0],
      createdAt: createdAt.getAndRequireEquals()
```

```
    });
```

```
        Location: zkTLS-Mina-zkApp/src/ZkonRequestCoordinator.ts, line 104
```

Affected Resources

- `zkon/zkTLS-Mina-zkApp/`

Recommendation

ZKON commented that “It allows to be zero, which will do nothing, the zkapp of the token will actually transfer 0 tokens. If you prepay a request with zero, when the oracle sees your request and try to fulfill it, he will not do it, since your balance is zero.” However, a vague request could be prevented before it is delivered to the oracle. Hence, it depends on the policy.

5. METHODOLOGY

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

5.1 Kickoff

The Kudelski Security Team 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.

5.2 Ramp-up

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

5.3 Review

The review phase is where most of the work on the engagement was performed. In this phase we have analyzed the project for flaws and issues that could impact the security posture. 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 was used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

Code Review

Kudelski Security Team reviewed the code within the project utilizing an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content, to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- authentication (*e.g.* A07:2021, CWE-306)
- authorization and access control (*e.g.* A01:2021, CWE-862)
- auditing and logging (*e.g.* A09:2021)
- injection and tampering (*e.g.* A03:2021, CWE-20)
- configuration issues (*e.g.* A05:2021, CWE-798)
- logic flaws (*e.g.* A04:2021, CWE-190)
- cryptography (*e.g.* A02:2021)

These categories incorporate common weaknesses and vulnerabilities such as the OWASP Top 10 and MITRE Top 25.

5.4 Smart Contracts

We reviewed the smart contracts, checking for additional specific issues that can arise such as:

- risk of centralization
- reentrancy
- (non)-adherence to existing standards
- unsafe arithmetic operations

5.5 Reporting

Kudelski Security delivered to the Client a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project.

In the report we not only point out security issues identified but also observations for improvement. The findings are categorized into several buckets, according to their overall severity: **Critical**, **High**, **Medium**, **Low**.

Observations are considered to be **Informational**. Observations can also consist of code review, issues identified during the code review that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

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

5.6 Verify

After the preliminary findings have been delivered, we verify 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 is the final report with any mitigated findings noted.

6. 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 below. The overall severity score is then computed as described in the next section.

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

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** 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.

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 effect 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.

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.

7. REFERENCE

[1] ZKON WHITEPAPER, ZKON, 08 October 2024

[2] TLSNotary, https://tlsnotary.org/

[3] zkApps Overview, Mina Documentation, https://docs.minaprotocol.com/zkapps/writing-a-zkapp

8. CONCLUSION

The objective of this code review was to evaluate the overall security of the code base and identify any vulnerabilities that would put the product at risk.

The Kudelski Security Team identified 5 security issues: 3 medium risks, and 2 lower risks. 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:

- TLS private key is imported in a secure way;
- Prover certificate is validated before executing a handshake protocol;
- Field is converted accurately without any potential risk.
- Potential weaknesses in TLS 1.2 are avoided.

The ZKON team addressed all these vulnerabilities and observations in the follow-up revision of the codebase.

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

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

9. APPENDIX

9.1 Cargo Clippy Logs

```
warning: `format!` in `println!` args
   --> src/proof_client.rs:106:5
    |
106 |       println!("{}", format!("Got response from API: {}/{}",
&req_data.baseURL,req_data.path));
    |
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^
    |
    = help: combine the `format!(..)` arguments with the outer
`println!(..)` call
    = help: or consider changing `format!` to `format_args!`
    = help: for further information visit https://rust-
lang.github.io/rust-clippy/master/index.html#format_in_format_args
    = note: `#[warn(clippy::format_in_format_args)]` on by default


warning: `format!` in `println!` args
   --> src/proof_client.rs:119:5
    |
119 |      println!("{}", format!("Proof written to file {}-proof-
{}.json.",&req_data.baseURL,milliseconds...
    |
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: combine the `format!(..)` arguments with the outer
`println!(..)` call
    = help: or consider changing `format!` to `format_args!`
    = help: for further information visit https://rust-
lang.github.io/rust-clippy/master/index.html#format_in_format_args


warning: `format!` in `println!` args
  --> src/proof_service_check.rs:99:5
```

```
   |
99 |     println!("{}", format!("Proof Server working as expected. Proof written to file {}-proof-{}.json. Performance: {:?}.",SERVER_DOMAIN,milliseconds,elapsed...
   | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
   |
   = help: combine the `format!(..)` arguments with the outer `println!(..)` call
   = help: or consider changing `format!` to `format_args!`
   = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#format_in_format_args

warning: unnecessary use of `to_string`
  --> src/main.rs:48:16
   |
48 |     .cert_path(cert_path.to_string())
   |                ^^^^^^^^^^^^^^^^^^^^^ help: use: `&cert_path`
   |
   = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#unnecessary_to_owned
   = note: `#[warn(clippy::unnecessary_to_owned)]` on by default

warning: unnecessary use of `to_string`
  --> src/main.rs:49:15
   |
49 |     .key_path(key_path.to_string())
   |               ^^^^^^^^^^^^^^^^^^^^ help: use: `&key_path`
   |
   = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#unnecessary_to_owned
```

```
warning: `zkon-proof-client` (bin "zkon-proof-client") generated 5 warnings (run `cargo clippy --fix --bin "zkon-proof-client"` to apply 2 suggestions)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.19s
```

9.2 TLS Connection Logs

```
$ cargo run --release main.rs
    Finished `release` profile [optimized] target(s) in 0.70s
     Running `target/release/zkon-proof-client main.rs`
Server Online.
```

```
$ openssl s_client -connect localhost:5000 -tls1_2
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = testserver.com
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN = testserver.com
verify error:num=21:unable to verify the first certificate
verify return:1
---
Certificate chain
 0 s:CN = testserver.com
   i:CN = ponytown RSA level 2 intermediate
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEADCCAmigAwIBAgICAcgwDQYJKoZIhvcNAQELBQAwLDEqMCgGA1UEAwwhcG9u
eXRvd24gUlNBIGxldmVsIDIgaW50ZXJtZWRpYXRlMB4XDTE2MDgxMzE2MDcwNFoX
DTIyMDIwMzE2MDcwNFowGTEXMBUGA1UEAwwOdGVzdHNlcnZlci5jb20wggEiMA0G
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCpVhh1/FNP2qvWenbZSghari/UThwe
dynfnHG7gc3JmygkEdErWBO/CHzHgsx7biVE5b8sZYNEDKFojyoPHGWK2bQM/FTy
```

```
niJCgNCLdn6hUqqxLAml3cxGW77hAWu94THDGB1qFe+eFiAUnDmob8gNZtAzT6Ky
b/JGJdrEU0wj+Rd7wUb4kpLInNH/Jc+oz2ii2AjNbGOZXnRz7h7Kv3sO9vABByYe
LcCj3qnhejHMqVhbAT1MD6zQ2+YKBjE52MsQKU/xhUpu9KkUyLh0cxkh3zrFiKh4
Vuvtc+n7aeOv2jJmOl1dr0XLlSHBlmoKqH6dCTSbddQLmlK7dms8vE01AgMBAAGj
gb4wgbswDAYDVR0TAQH/BAIwADALBgNVHQ8EBAMCBsAwHQYDVR0OBBYEFMeUzGYV
bXwJNQVbY1+A8YXYZY8pMEIGA1UdIwQ7MDmAFJvEsUi7+D8vp8xcWvnEdVBGkpoW
oR6kHDAaMRgwFgYDVQQDDA9wb255dG93biBSU0EgQ0GCAXswOwYDVR0RBDQwMoIO
dGVzdHNlcnZlci5jb22CFXNlY29uZC50ZXN0c2VydmVyLmNvbYIJbG9jYWxob3N0
MA0GCSqGSIb3DQEBCwUAA4IBgQBsk5ivAaRAcNgjc7LEiWXFkMg703AqDDNx7kB1
RDgLalLvrjOfOp2jsDfST7N1tKLBSQ9bMw9X4Jve+j7XXRUthcwuoYTeeo+Cy0/T
1Q78ctoX74E2nB958zwmtRykGrgE/6JAJDwGcgpY9kBPycGxTlCN926uGxHsDwVs
98cL6ZXptMLTR6T2XP36dAJZuOICSqmCSbFR8knc/gjUO36rXTxhwci8iDbmEVaf
BHpgBXGU5+SQ+QM++v6bHGf4LNQC5NZ4e4xvGax8ioYu/BRsB/T3Lx+RlItz4zdU
XuxCNcm3nhQV2ZHquRdbSdoyIxV5kJXel4wCmOhWIq7A2OBKdu5fQzIAzzLi65EN
RPAKsKB4h7hGgvciZQ7dsMrlGw0DLdJ6UrFyiR5Io7dXYT/+JP91lP5xsl6Lhg9O
FgALt7GSYRm2cZdgi9pO9rRr83Br1VjQT1vHz6yoZMXSqc4A2zcN2a2ZVq//rHvc
FZygs8miAhWPzqnpmgTj1cPiU1M=
-----END CERTIFICATE-----
subject=CN = testserver.com

issuer=CN = ponytown RSA level 2 intermediate

---
No client certificate CA names sent
Peer signing digest: SHA512
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 1498 bytes and written 281 bytes
Verification error: unable to verify the first certificate
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
```

```
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID:
82A62E4ACC320E55017D5BF0DA2A3233AA8151C74AF91352AE2A7805A7DE2F67
    Session-ID-ctx:
    Master-Key:
6DF8E6438F0AC4BDB1C8905FF153D896681B7957570CFFA9A2E53F222177BF4C853
D76DDB31E2914E4AD431318B86805
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1730360474
    Timeout   : 7200 (sec)
    Verify return code: 21 (unable to verify the first certificate)
    Extended master secret: yes
---
```

DOCUMENT RECIPIENTS

NAME	POSITION	CONTACT INFORMATION
Paul Foix	CEO	paul@zkon.xyz
Juanjo Chust	CTO	juanjo@zkon.xyz
Thomas Kevers		thomas.kevers@minafoundation.com

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Jean-Sebastien Nahon	Application and Blockchain Security Practice Manager	jean-sebastien.nahon@kudelskisecurity.com
Ana Acero	Project Manager/ Operations Coordinator	ana.acero@kudelskisecurity.com

DOCUMENT HISTORY

VERSION	DATE	STATUS/ COMMENTS
V1.0	4 November 2024	Initial draft
V1.1	10 December 2024	Final proposal