TSS ECDSA CLI Secure Code Review

Technical Report

Uniwire

19 May 2025
Version: 1.3

Kudelski Security – Nagravision Sàrl

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

For Public Release

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Uniwire (“the Client”) engaged Kudelski Security (“Kudelski”, “We”) to perform the TSS ECDSA CLI Secure Code Review.

The assessment was conducted remotely by the Kudelski Security Team.

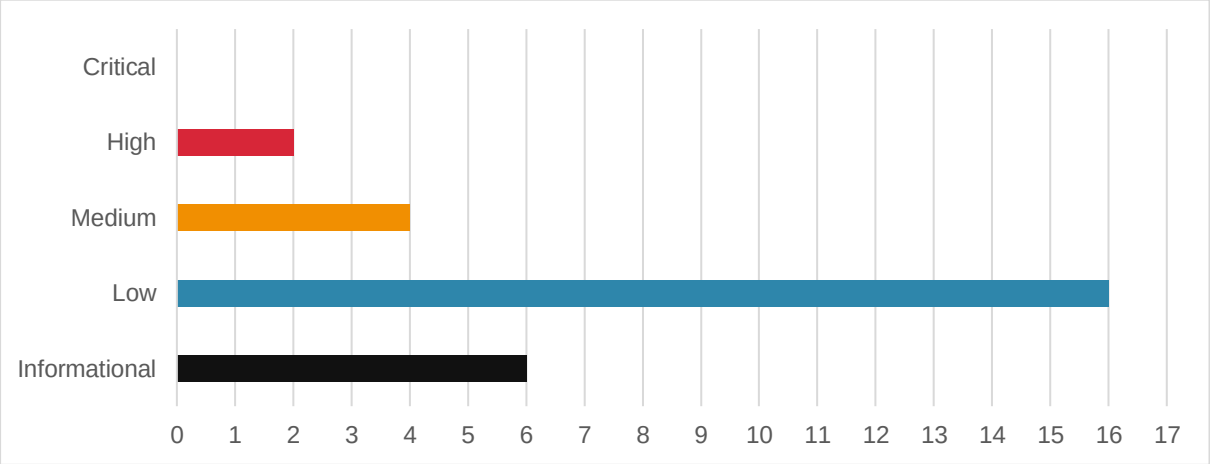
The review took place between 06 February 2025 and 26 February 2025, and focused on the following objectives:

- 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.

- Logical Error of Small Factors Check
- Insufficient Authentication in Signing Room Access Control
- Lack of Input Validation



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 `tss-ecdsa-cli` is a wrapper CLI for a Rust implementation of (t,n)-threshold ECDSA and EdDSA, including the support for HD keys (BIP32).

1.2 Scope

The scope consisted in specific Rust files and folders located at:

- Source code repository : `https://github.com/uniwire/tss-ecdsa-cli/`
    - commit: `ce7a6ca1a31e49198343e78514afbe44be261be3`

The folders and files in scope are:

```
src/
├── common
│   ├── hd_keys.rs
│   ├── manager.rs
│   ├── mod.rs
│   └── signing_room.rs
├── main.rs
├── protocols
│   ├── ecdsa
│   │   ├── curv7_conversion.rs
│   │   ├── keygen.rs
│   │   ├── mod.rs
│   │   └── signer.rs
│   ├── eddsa
│   │   ├── keygen.rs
│   │   ├── mod.rs
│   │   ├── signer.rs
│   │   └── test.rs
      │   └── mod.rs
```

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.

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 Teams
- 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 or acknowledged all vulnerabilities and weaknesses in the following codebase revision:

- feature/audit_applied (commit: ff90e9ff867549656bf9a2b39d0010dd648715b8)

2. 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-CHL-F-01	High	Logical Error of Small Factors Check	Resolved
KS-CHL-F-02	High	Insufficient Authentication in Signing Room Access Control	Resolved
KS-CHL-F-03	Medium	Error Message Contains Sensitive Information	Resolved
KS-CHL-F-04	Low	Lack of State Recovery and Session Management	Acknowledged
KS-CHL-F-05	Medium	Dlog Proof Not Validated Properly	Resolved
KS-CHL-F-06	Medium	Chain Code Not Validated Properly	Resolved
KS-CHL-F-07	Medium	Message Hash Assumed Implicitly	Resolved
KS-CHL-F-08	Low	Safe Primes Not Used	Resolved
KS-CHL-F-09	Low	Boundary of System Parameters Not Checked	Resolved
KS-CHL-F-10	Low	Error Handling with unwrap and expect	Resolved
KS-CHL-F-11	Low	Lack of Input Validation	Resolved
KS-CHL-F-12	Low	Total Parties Not Validated	Resolved
KS-CHL-F-13	Low	Party_number not Validated in signing_room.rs	Resolved
KS-CHL-F-14	Low	Party Index Not Counted Accurately	Resolved
KS-CHL-F-15	Low	Parsed Key Data Not Validated	Resolved
KS-CHL-F-16	Low	Unsafe HashMap Access	Resolved
KS-CHL-F-17	Low	HD Child Key Derivation Not Compliant to BIP32	Acknowledged
KS-CHL-F-18	Low	HD Child Private Key Not Used For EdDSA Signing	Resolved
KS-CHL-F-19	Low	EdDSA Key Clamp Not Applied Properly	Resolved

#	SEVERITY	TITLE	STATUS
KS-CHL-F-20	Low	Input Parameter of check_sig Not Validated	Resolved
KS-CHL-F-21	Low	AES256 Key Length Not Checked	Resolved
KS-CHL-F-22	Low	Lack of Test Vectors	Acknowledged

Findings overview.

2.1 KS-CHL-F-01 Logical Error of Small Factors Check

Severity	Impact	Likelihood	Status
High	High	High	Resolved

Description

The variable `failed` is initialized to `false`. Hence, the following logical operation returns always false: `false && is_divisible_by_first_n_primes`.

2.2 KS-CHL-F-02 Insufficient Authentication in Signing Room Access Control

Severity	Impact	Likelihood	Status
High	High	Low	Resolved

Description

The signing room protocol lacks proper authentication for party membership both during signup and signing phases. The only validation relies on `party_number` and auto-generated `party_uuid` pairs, without any cryptographic verification of the party's identity or authorization.

2.3 KS-CHL-F-03 Error Message Contains Sensitive Information

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

The `/get` endpoint in `manager.rs` returns error messages that include the requested key value when a key is not found. This verbose error handling could help attackers enumerate valid keys and gather information about active signing sessions. This can help attacker to gather the information about Active party numbers, Active rounds and, Valid UUIDs.

2.4 KS-CHL-F-04 Lack of State Recovery and Session Management

Severity	Impact	Likelihood	Status
Low	Low	High	Acknowledged

Description

The current implementation of the protocol lacks robust state recovery and session management mechanisms. When parties `timeout` or disconnect during the `signing` process, this is impossible to recover or resume the process.

2.5 KS-CHL-F-05 Dlog Proof Not Validated Properly

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

In the ECDSA sign function, the dlog proof is generated and broadcasted in the phase 5. However, the vector `phase_5a_dlog_vec` is not validated after received.

In a nutshell, the dlog proof is based on a non-interactive Schnorr protocol with Fiat-Shamir transformation as follows:

`Prover: r, u=g*r, y=g*x, c=H(u,g,y), z=r-c*x ->` broadcasts `(y,u,z)`

`Verifier: c=H(u,g,y), v = g*z+y*c = g*(r-c*x)+y*c = g*r-g*c*x+y*c=g*r=g*r`

If `phase_5a_dlog_vec` is zero (point at infinity), i.e., y=u=z=0, then v=0. Since v=u, the verification is successful.

Similarly, the function `verify_dlog_proofs` does not validate the input parameters. If all values in `dlog_proofs_vec` are zero, it could be verified. In addition, if `share_count = y_vec_len = dlog_proofs_vec.len = 0`, it does not execute `DLogProof::verify`, so that any proof could be verified.

2.6 KS-CHL-F-06 Chain Code Not Validated Properly

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

The chain code is used to introduce deterministic random data to the HD key derivation, so that knowing the index and a child key is not sufficient to derive a child key. If `chain_code_in_env` is not empty, the variable `chain_code` is initialized by an environmental variable `chain_code_in_env` regardless of the value derived from the key file.

If this env variable is set to 0 or a known value by chance, the child public key and `f_l_new` can be derived by anyone. If the HD key is supported, `chain_code` is updated once again by `chain_code = g * chain_code`, but it is still constant. Hence, no entropy is added to the HD child key derivation.

2.7 KS-CHL-F-07 Message Hash Assumed Implicitly

Severity	Impact	Likelihood	Status
Medium	Low	High	Resolved

Description

The first step to sign with ECDSA is to hash the message. The function `sign` assumes the input message is already hashed by the signer, as commented. However, such requirement is not given in the main function. Furthermore, if `message` is not a hashed message, then the `sign` function is vulnerable to the forgery attack.

Suppose `q` is the multiplicative group order over secp256k1. Then, if `(r,s)` is a valid signature for the `message`, then `(r,s)` is also valid for the message `message + i*q`, where `i=1,2,...,` since `message + i*q` is reduced modulo 2^{256}, then further mapped to a point over secp256k1, which results in the same value as `message`.

2.8 KS-CHL-F-08 Safe Primes Not Used

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

The function `create` is used to generate `party_keys`, but safe primes are not used. In the GG18 paper, it is recommended to use safe primes for strong RSA because the security of ZK proof is based on the assumption that the Prover cannot solve the Strong RSA problem over N.

2.9 KS-CHL-F-09 Boundary of System Parameters Not Checked

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

The GG18 protocol is for (t,n) threshold signature, that is, n ≥ t + 1 and only t + 1 players are needed to sign. However, the bound of critical parameters is not checked. They should satisfy the following conditions:

- `1 <= THRESHOLD < PARTIES <= MAX_ALLOWD_PARTIES`
- `1 <= party_num_int <= PARTIES`

Here, `MAX_ALLOWD_PARTIES` can be set as an environment variable.

2.10 KS-CHL-F-10 Error Handling with `unwrap` and `expect`

Severity	Impact	Likelihood	Status
Low	Medium	Low	Resolved

Description

The code uses `unwrap` and `expect` extensively, which can cause the program to panic and crash if an error occurs. This approach does not provide a graceful way to handle errors and can lead to unexpected program termination.

2.11 KS-CHL-F-11 Lack of Input Validation

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

Since the key generation, the HD key derivation, and the signing function are executed separately, it is important to validate the input parameters of the functions properly, in particular, for the functions declared as "`pub fn`". Although the GG18 protocol will stop and abort if the input data are not valid, there exists a risk that legitimate users may leak their valuable information to an adversary before being aborted. Input validation is the first line of defence to reduce such risk.

2.12 KS-CHL-F-12 Total Parties Not Validated

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

In the function `signup`, `total_parties` is not checked whether it is greater than the input parameter `threshold`. The (t,n) threshold signature is valid only when `total_parties` is greater than `threshold`.

2.13 KS-CHL-F-13 Party_number not Validated in signing_room.rs

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The `SigningRoom` implementation lacks proper validation of party numbers during party addition, potentially leading to protocol inconsistencies and security vulnerabilities. The function `add_party()` accepts any `u16` value without validation against room size or other constraints, and performs unsafe integer conversions.

2.14 KS-CHL-F-14 Party Index Not Counted Accurately

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The variable `total_parties` is used to count the number of participants in the function `signup`. Although the signature is valid if `total_parties == threshold + 1` , it is more accurate and consistent to use `total_parties` instead of `threshold + 1` in the (t,n) threshold signature.

2.15 KS-CHL-F-15 Parsed Key Data Not Validated

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

The key data are loaded from the key file, but they are not validated. Although it is assumed that the key data are stored in a safe place, it would be always recommended to check if the loaded key data are valid since they are critical for the system security.

2.16 KS-CHL-F-16 Unsafe HashMap Access

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The `SigningRoom` implementation uses unsafe `HashMap` access patterns with multiple `unwrap()` calls. The code assumes party existence without proper validation, which can lead to runtime panics and potential denial of service vulnerabilities

2.17 KS-CHL-F-17 HD Child Key Derivation Not Compliant to BIP32

Severity	Impact	Likelihood	Status
Low	High	Low	Acknowledged

Description

The function `hd_key` is used to derive a HD child key for both ECDSA and EdDSA. Although the derived child key could be used for signing, this function is proprietary and not compliant to BIP32, which has limitation to extend the service, and may lead to a unknown attack in the future.

Also, this function does not consider the key clamping for the EdDSA key which is defined in RFC 8032. Since `f_l_new` has no guarantee on its highest bits set or cleared, or the lowest three bits cleared, thus the addition of `f_l_new` to the private key may cause a small cofactor vulnerability.

Note that the party keys are immune to this issue since they are created with the key clamping by the function `create_from_private_key`. This function is called internally from the function `phase1_create`.

2.18 KS-CHL-F-18 HD Child Private Key Not Used For EdDSA Signing

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The function `hd_key` is used to derive the public child key and `f_l_new` for EdDSA. Since `hd_key` does not take any private information, the child public key and `f_l_new` can be derived by anyone who knows the chain code. Later, `f_l_new` is used to update the signature without updating the private key. Although the signature can be verified in this manner, it does not make much sense to use the child key because the child private key is not really used for EdDSA signing. This is not the case for ECDSA where the private key is updated and used for signing process.

2.19 KS-CHL-F-19 EdDSA Key Clamp Not Applied Properly

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The function `update_hd_derived_public_key` is to clamp the EdDSA public key. However, according to RFC 8032, Section 5.1.5, the private key or a secret scala should be clamped to avoid the small subgroup attack. The public key is not relevant to this attack.

2.20 KS-CHL-F-20 Input Parameter of check_sig Not Validated

Severity	Impact	Likelihood	Status
Low	Low	High	Resolved

Description

The function `check_sig` is not applicable for the general ECDSA verification. The message length is implicitly limited to 32 bytes, otherwise an overflow occurs. Also, if `msg` could be an arbitrary message, then an adversary can set the input parameters as follows: `msg = 0 mod q, (r,s) = (a, a)`, where `a` is the x-coordinate of `pk`. Then `u1 = zs^-1 = 0`, and `u2 = rs^-1 = 1`, so that the curve point `(x,y) = u1*g + u2*pk = pk`. Hence, the signature is valid since `r=a=x`. This type of vulnerability can be generalized as presented in the reference below.

2.21 KS-CHL-F-21 AES256 Key Length Not Checked

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

The function `aes_encrypt` encrypts the plaintext using the key which is given as an input parameter. However, the key is not validated upfront. The key should not be `NULL`, and the key length is supposed to be 32 bytes since the AES256 GCM mode is used for encryption.

Due to the same reason, the length of vector `enc_keys` should be 32 bytes.

2.22 KS-CHL-F-22 Lack of Test Vectors

Severity	Impact	Likelihood	Status
Low	High	Low	Acknowledged

Description

According to the `cargo llvm-cov` tool (v0.6.10), the overall test coverage of code in scope reaches less than 6%.

3. 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-CHL-O-01	Informational	Outdated Dependencies	Informational
KS-CHL-O-02	Informational	Best Secure Code Practice	Informational
KS-CHL-O-03	Informational	Security Concerns on GG18 over Ed25519	Informational
KS-CHL-O-04	Informational	Security Overview of Current Implementation	Informational
KS-CHL-O-05	Informational	Bitforge Attack and Prime Generation	Informational
KS-CHL-O-06	Informational	TSSHOCK Attack and Dlog Proof	Informational

Observations overview.

3.1 KS-CHL-O-01 Outdated Dependencies

Description

The `cargo audit` (v0.21.0) tool identified 10 vulnerabilities and 9 allowed warnings on dependencies. Among those, the following dependencies contain vulnerabilities.

3.2 KS-CHL-O-02 Best Secure Code Practice

Description

- The variable `_index` and `_cc_new` are not used at all.
- The variable `keysfile_path` is double referenced in `src/protocols/ecdsa/keygen.rs`, while it is single referenced in `src/protocols/eddsa/keygen.rs`.
- The variable `chain_code` is set by `chain_code_in_env`, and used as an initialized value for the ECDSA keygen. However, it is not used for the EdDSA.
- The function `verify` is redundant because it is already verified internally in the function `output_signature`.
- It is not clear why the function `update_hd_derived_public_key` is needed. According to RFC 8032, the private key (or scalar) should be clamped, not the public key.
- Some functions are unnecessarily declared as “pub fn” although they are used within a module.
- The delay value is hard-coded as a magic number (25ms, 100ms or 250ms), which could be flexible, depending on the use cases. It is recommended to set them from an env variable.
- The function `run_keygen` is too long: 248 lines for ECDSA and 250 lines EdDSA. Also, the function `sign` in ecdsa is even longer: 547 lines. They could be divided into several sub-functions based on the phases to improve the readability and testability.
- The function `generate_shared_chain_code` is used in both ECDSA and EdDSA. However, `exchange_data` is actually located under EdDSA folder, so there is dependency between ECDSA and EdDSA folders.
- The name of repository `tss-ecdsa-cli` is misleading since this cli utility supports not only ECDSA, but also EdDSA, in particular, Ed25519.

3.3 KS-CHL-O-03 Security Concerns on GG18 over Ed25519

Description

GG18 was specifically designed for ECDSA with security proof. However, it is not directly applicable to Ed25519, which could raise some security considerations. Ed25519 uses a different signing equation than ECDSA and relies on different mathematical properties. There is no security proof on the GG18 over EdDSA.

Some key differences between GG18 over Ed25519 and ECDSA include:

- GG18 relies on Paillier encryption during distributed key generation. The homomorphic properties used in GG18 are specifically designed for ECDSA's multiplicative structure. The MtA (multiplicative-to-additive) conversion that GG18 uses for ECDSA is not required to Ed25519's signing equation.
- The zero-knowledge proofs in GG18 are constructed for ECDSA's mathematical relationships, which is not really needed for EdDSA.
- Side-Channel Vulnerabilities Ed25519 was designed to be resistant to certain side-channel attacks through constant-time operations. The GG18 protocol modifications required for Ed25519 might reintroduce timing dependencies that Ed25519 was specifically designed to avoid.
- Multiple projects for GG18 over EdDSA have been developed but not maintained anymore.

Therefore, it would be safer to use protocols specifically designed for EdDSA, for example, Frost in the future.

3.4 KS-CHL-O-04 Security Overview of Current Implementation

Description

The current code was developed based on the Kzen multiparty ecdsa / eddsa library which were originally created to enable users to experiment with protocols. The keygen and signing protocols are currently implemented based on examples and test codes which are obviously not suitable for production.

The HD key derivation function is proprietary, not based on the standard BIP32 specification. The code was forked from here, which does not provide any security proof. The code is somehow similar to the function derive_tweak in the BIP32 repository where a tweak value can be derived to generate the child key.

3.5 KS-CHL-O-05 Bitforge Attack and Prime Generation

Description

GG18 uses the Paillier cryptosystem for additive homomorphic properties. When setting up the Paillier cryptosystem, each participant generates two large prime numbers (p, q) as their private key, and then `N = p * q` as their public key.

The specification of the GG18 threshold ECDSA signature protocol may be vulnerable when a malicious signer is able to use a modulus N containing small factors (say, less than 2^20) i.e. `N = small_prime_1 * small_prime_2 * ... * q`, and none of the other participants check, which allowing an attacker to recover the shared secret key. The master key can then be reconstructed from these shares.

This attack is protected by checking whether N is divisible by the first n primes. However,

- it is not explained why primes up to 2^25 is chosen;
- a proprietary prime generation function is used.

3.6 KS-CHL-O-06 TSSHOCK Attack and Dlog Proof

Description

The TSSHOCK vulnerabilities stem from implementation mistakes in the range proof subprotocol dlog proof, allowing malicious actors to forge proofs and potentially recover private keys. According to the TSSHOCK article (see Reference below), Zengo-X multi-party ECDSA is claimed to be vulnerable to this attack, hence, by nature, this is of concern on the code in scope.

In the ECDSA sign function, the dlog proof is generated in phase 5. It is actually a non-interactive Schnorr protocol with Fiat-Shamir transformation, where the c-split attack is not directly applicable. In theory, a malicious party could choose a rho in such a way that the proof could be forged without knowing the secret key. The vulnerability could be exploited to recover the private key of the party. However, it seems not feasible in the protocol level since the rho is randomly chosen in the library.

Note that this attack is not applicable to the EdDSA because there is no MtA conversion there, hence, neither the range proof nor the dlog proof is required.

4. 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.

4.1 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.

4.2 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.

4.3 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.

Cryptography

We analyze the cryptographic primitives and components as well as their implementation. We check 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

4.4 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.

4.5 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.

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 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 effect 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 effect 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.

6. REFERENCES

- Fast Multiparty Threshold ECDSA with Fast Trustless Setup
- https://github.com/ZenGo-X/multi-party-ecdsa
- https://github.com/ZenGo-X/multi-party-eddsa
- Crate Multi-party ECDSA
- https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
- BIP32-Ed25519 Hierarchical Deterministic Keys over a Non-linear Keyspace
- https://www.fireblocks.com/blog/gg18-and-gg20-paillier-key-vulnerability-technical-report
- https://verichains.io/tsshock/

7. 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 22 security issues: 2 high risks, 4 medium risks, and 16 low risks. On average, the effort needed to mitigate these risks is estimated as `medium`.

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

- Fix the logical errors.
- Authenticate parties to access the signing room.
- Validate the input parameters of public functions.
- Derive the HD keys compliant to BIP32.

The Client addressed or acknowledged 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 Uniwire for their trust, help and support over the course of this engagement and is looking forward to cooperating in the future.