Audit of CGGMP21

DFNS

21 September 2023
Version: 1.1

Presented by:
Kudelski Security Research Team
Kudelski Security – Nagravision Sarl

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

DOCUMENT PROPERTIES

Version:	1.1
File Name:	Audit_CGGMP21
Publication Date:	21 September 2023
Confidentiality Level:	Restricted
Document Owner:	Tommaso Gagliardoni
Document Recipient:	Jonathan Katz
Document Status:	Approved

TABLE OF CONTENTS

TABLE OF FIGURES

EXECUTIVE SUMMARY

Kudelski Security (“Kudelski”, “we”), the cybersecurity division of the Kudelski Group, was engaged by DFNS (“the Client”) to conduct an external security assessment in the form of a code audit of the “CGGMP21 Threshold ECDSA Signature Library” solution (“the Product”) developed by the Client.

The assessment was conducted remotely by the Kudelski Cybersecurity Research Team in June 2023 and focused on the following objectives:

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

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

1.1 Engagement Scope

The scope of the audit were the repositories located at:

- https://github.com/dfns-labs/cggmp21

on commit: `d2b8494a480c638d2db81d41e7eb5833b2ebba3b`.

- https://github.com/dfns-labs/generic-ec/

on commit: `c418c93b19fced622382d3df34368911b6b99731`.

- https://github.com/dfns-labs/paillier-zk

on commit: `4f034852bd61d199b3b4462a3079fb7c0fa1adc9`.

1.2 Engagement Analysis

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

As a result of our work, we identified **1 Medium**, and **2 Informational** findings.

We attribute the small number of issues found to the high quality of the implementation and the great care that was put in terms of documentation. The only issue worth of attention is about the possibility of forging fake commitments, which is easily fixable and of low impact for the Product itself, but we think it might be inherited by other implementations with more serious consequences if left unaddressed.

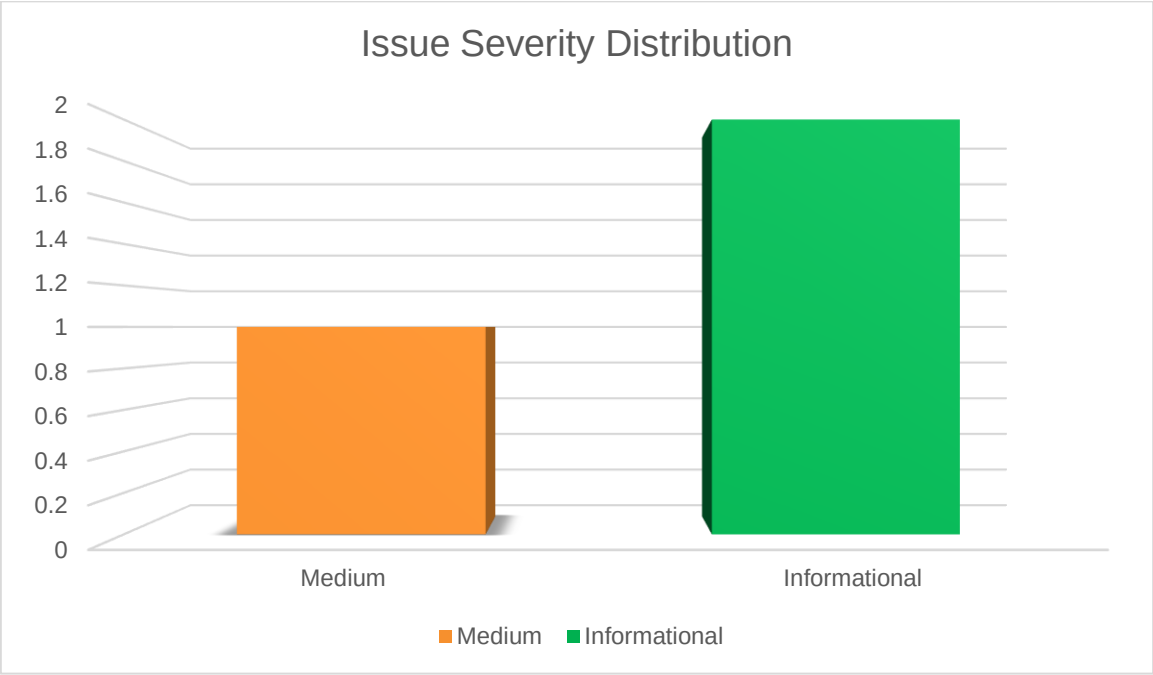

Figure 1 Issue Severity Distribution

1.3 Observations

The Product we audited is an implementation of the MPC Threshold ECDSA Scheme described in [1]. The Client provided a reference document that aims at simplifying [1] by easing notations, correcting typos, and restricting to the actual version of the protocol used. In particular, the Product implements the three-round presign scheme version by providing two different key generation schemes: one that generates an additive n-out-of-n sharing of a key, and the other generates a t-out-of-n Shamir secret sharing of a key. The Client has not implemented a t-out-of-n threshold key refresh mechanism. Threshold presigning is performed instead by first mapping the t-out-of-n key shares to a t-out-of-t sharing of the key (using Lagrange interpolation) and then running the non-threshold t-out-of-t protocol among the t parties.

In general, we found the implementation to be of a high standard and we found very few issues. We believe that all the identified vulnerabilities can be easily addressed. Moreover, we did not find evidence of any hidden backdoor or malicious intent in the code.

[1] https://eprint.iacr.org/2021/060.pdf

1.4 Issue Summary List

The following security issues were found:

ID	SEVERITY	FINDING	STATUS
KS-DF2-F-01	Medium	Malleability Of Tree Hash Commitments	Remediated

The following are non-security observations related to general design and optimization:

ID	SEVERITY	FINDING	STATUS
KS-DF2-O-01	Informational	Warnings With `cargo build`	Informational
KS-DF2-O-01	Informational	Cannot Run `cargo test`	Informational

2. METHODOLOGY

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

Figure 2 Methodology Flow

2.1 Kickoff

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

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

2.2 Ramp-up

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

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

2.3 Review

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

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project

3. Assessment of the cryptographic primitives used
4. Compliance of the code with the provided technical documentation

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

Code Safety

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

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

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

Cryptography

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

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

Technical Specification Matching

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

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

2.4 Reporting

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

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

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

- High
- Medium
- Low
- Informational

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

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

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

2.5 Verify

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

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

2.6 Additional Note

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

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

3. TECHNICAL DETAILS OF SECURITY FINDINGS

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

3.1 Malleability Of Tree Hash Commitments

Finding ID: KS-DF2-F-01

Severity: Medium

Status: Remediated

Location: `/generic-ec-d/generic-ec-zkp/src/hash_commitment.rs`

Description and Impact Summary

We found a potential issue in the way hash commitment of multiple elements is computed. First of all, we observe that the code contains obsolete documentation as comments:

```
//! ## Algorithm
//! Underlying algorithm is based on hash function $\H$. To commit data, we sample a large random nonce,
//! and hash it along with data. When we hash bytestrings, we prepend its length to it, in that way we
//! ensure that there's only one set of inputs that can be decommitted.
//!
//! Roughly, algorithm is:
//!
//! 1. $commit(i_1, \dots, i_n) =$ \
//!    1. $\mathit{nonce} \gets \\{0,1\\}^k$
//!    2. $\text{return}\ \H(\dots \\| \text{u32\\_to\\_be}(\mathit{len}(i_j)) \\| i_j \\| \dots \\| \mathit{nonce}),
\mathit{nonce}$
//!
//! 2. $decommit(commit, nonce, i_1, \dots, i_n) =$
//!    1. $\text{return}\ \H(\dots \\| \text{u32\\_to\\_be}(\mathit{len}(i_j)) \\| i_j \\| \dots \\| nonce) \\?
commit$
```

Instead, what really happens is that commitment of various concatenated elements is computed as a Merkle tree, where elements of different granularity are hashed in a nested way.

```
    pub fn mix_many_bytes(self, list: impl IntoIterator<Item = impl AsRef<[u8]>>) -> Self {
        let hash = list
            .into_iter()
            .fold(D::new(), |d, i| d.chain_update(D::digest(i)))
            .finalize();
        Self(self.0.chain_update(hash))
    }
```

This means that, for example, if `A` is an element and `B[]` is an array of `n` elements, then commitment `com(A,B[],nonce)` is performed as:

`H(H(A)||H(H(B[0])||…||H(B[n-1]))||H(nonce))`

However, notice that this tree hashing method does not include a depth index of the elements. This means that the commitment above is the same as `com(A,C,nonce)` for an element `C` that "just happens" to be a concatenation of `H(B[i])`'s. This might lead to collisions, and more generally defeats the binding property of commitments, which might in turn open the possibility of malicious attacks to the protocol, for example by tricking parties into signing a transaction different from what presented.

We observe that, in the context of the CGGMP21 implementation in exam, exploiting this vulnerability is probably not trivial, because the attacker generally doesn't control the structure of the tree, they can only change content and number of leaves in the tree. However, we believe this issue has a high possibility of being inherited in derived code that is used in more flexible scenarios where the risk would become much more tangible, this is why we ranked this severity as Medium.

Recommendation

We recommend either including the depth index in the hash computation, or using standardized constructions, for example RFC 9162 [2], which solve this issue by treating hash of leaves and nodes differently.

[2] https://datatracker.ietf.org/doc/html/rfc9162#name-merkle-trees\

Status Details

This has been fixed in PR #4, by making subtree hashing distinct from leaf hashing.

4. OTHER OBSERVATIONS

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

4.1 Warnings With `cargo build`

Observation ID: KS-DF2-O-01

Location: `various`

Description and Impact Summary

We observe warnings when running `cargo build`:

```
cargo build -p cggmp21 --no-default-features --features paillier-zk/rust

warning: unused import: `subset`
  --> cggmp21/src/key_share.rs:15:42
   |
15 | use crate::utils::{lagrange_coefficient, subset};
   |                                          ^^^^^^
   |
   = note: `#[warn(unused_imports)]` on by default

warning: variants `NoKeyShares`, `DifferentKeyShares`, `TooFewKeyShares`, `Subset`, and
`Interpolation` are never constructed
   --> cggmp21/src/key_share.rs:585:5
    |
583 | enum ReconstructErrorReason {
    |      ---------------------- variants in this enum
584 |     #[error("no key shares provided")]
585 |     NoKeyShares,
    |     ^^^^^^^^^^^
...
589 |     DifferentKeyShares,
    |     ^^^^^^^^^^^^^^^^^^
590 |     #[error("expected at least `t={t}` key shares, but {len} key shares were provided")]
591 |     TooFewKeyShares { len: usize, t: u16 },
    |     ^^^^^^^^^^^^^^^
592 |     #[error("subset function returned error (seems like a bug)")]
593 |     Subset,
    |     ^^^^^^
594 |     #[error("interpolation failed (seems like a bug)")]
595 |     Interpolation,
```

```
    |     ^^^^^^^^^^^^
    |
    = note: `ReconstructErrorReason` has a derived impl for the trait `Debug`, but this is
intentionally ignored during dead code analysis
    = note: `#[warn(dead_code)]` on by default

warning: `cggmp21` (lib) generated 2 warnings (run `cargo fix --lib -p cggmp21` to apply 1 suggestion)
    Finished dev [unoptimized + debuginfo] target(s) in 25.40s
```

Recommendation

Fix the warnings.

Status

The Client acknowledged the issue and is working on a patch. As a temporary solution, warnings disappear using `cargo build -p cggmp21 --no-default-features --features paillier-zk/rust --features spof`.

4.2 Cannot Run `cargo test`

Observation ID: KS-DF2-O-02

Location: `various`

Description and Impact Summary

Running `cargo test` makes the compilation crash, generating gigabytes of garbage artifacts that must be subsequently cleaned with `cargo clean`. For our testing environment, the issue appears to be caused by the `gmp-mpfr-sys` dependency.

Recommendation

Fix the dependency issue.

Status

The Client acknowledged the issue and is investigating the cause.

APPENDIX A: ABOUT KUDELSKI SECURITY

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

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

Kudelski Security

Route de Genève, 22-24

1033 Cheseaux-sur-Lausanne

Switzerland

Kudelski Security

5090 North 40th Street

Suite 450

Phoenix, Arizona 85018

APPENDIX B: DOCUMENT HISTORY

VERSION	STATUS	DATE	AUTHOR	COMMENTS
0.1	Draft	23 June 2023	Tommaso Gagliardoni	Initial draft
1.0	Final	13 July 2023	Tommaso Gagliardoni	Final version
1.1	Final	21 September 2023	Tommaso Gagliardoni	Fixed commit number

REVIEWER	POSITION	DATE	VERSION	COMMENTS
Nathan Hamiel	Senior Director - Research	26 June 2023	0.1	Reviewed
Nathan Hamiel	Senior Director - Research	13 July 2023	1.0	Reviewed
Nathan Hamiel	Senior Director - Research	21 September 2023	1.1	Reviewed

APPROVER	POSITION	DATE	VERSION	COMMENTS
Nathan Hamiel	Senior Director - Research	26 June 2023	0.1	Approved
Nathan Hamiel	Senior Director - Research	13 July 2023	1.0	Approved
Nathan Hamiel	Senior Director - Research	21 September 2023	1.1	Approved

APPENDIX C: SEVERITY RATING DEFINITIONS

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

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

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