Audit of tKey

Tor.us Labs

28 October 2020
Version: 1.0

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

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

DOCUMENT PROPERTIES

Version:	1.0
File Name:	Audit_torus_tkey
Publication Date:	28 October 2020
Confidentiality Level:	For public release
Document Owner:	Tommaso Gagliardoni
Document Recipient:	Yong Zhen Yu
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 Tor.us Labs ("the Client") to conduct an external security assessment in the form of a code audit of the tKey distributed key management solution developed by the Client.

The assessment was conducted remotely by Dr. Tommaso Gagliardoni, Cryptography Expert with support of collaborators of the Kudelski Cybersecurity Team. The audit took place from September 24, 2020 to October 16, 2020 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 was a complete code audit of the tKey key management solution client written in Typescript, with a particular attention to safe implementation of cryptographic primitives, randomness generation, and potential for misuse and leakage of secrets.

The target of the audit was commit `4a827c615a4f6f7dbd5bbe7b1319a88761230f9e` in the Git repository provided by the Client.

The documentation and specs of the solution were provided at:

https://hackmd.io/keVuRfrwSxygfyCfzsrQfw

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 **4 Low** and **3 Informational** findings.

Most of these findings are not critical. The most important ones are a consequence of the fact that the specifics of the product do not enforce integrity of the storage layer encryption, but only confidentiality. The most straightforward way to mitigate these is to explicitly enforce the use of strong integrity-preserving encryption schemes.

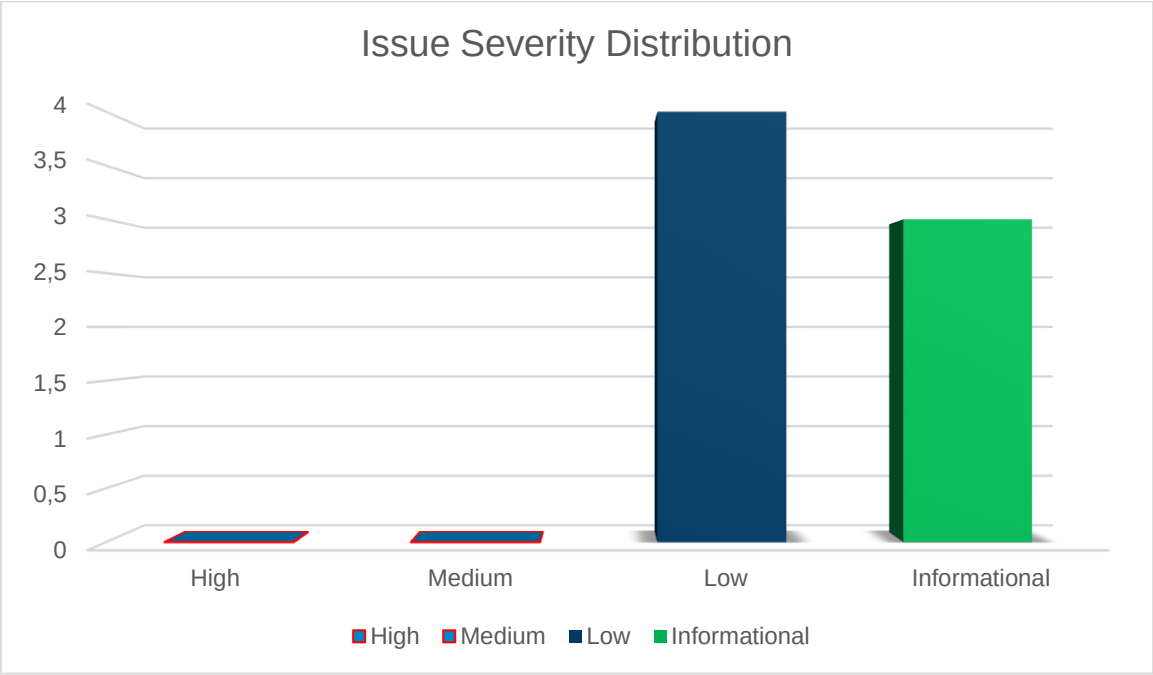

Figure 1 Issue Severity Distribution

1.3 Observations

The code we audited implements tKey, a shared key management and storage solution that uses Shamir Secret Sharing to split a secret in order to facilitate a user's access to that secret with high security assurance. The cryptographic primitives used are relatively simple, which is a very good feature for security, and part of the reason why we could not identify many important vulnerabilities.

In general, we found the implementation to be of high standard and 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.

1.4 Issue Summary List

The following security issues were found:

ID	SEVERITY	FINDING	STATUS
KS-TTKEY-F-01	Low	Missing check that share indices are different	Remediated
KS-TTKEY-F-02	Low	Missing check that share indices are not 0 or 1	Remediated
KS-TTKEY-F-03	Low	Missing check of correct decryption	Remediated
KS-TTKEY-F-04	Low	Possible race condition	Remediated

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

ID	SEVERITY	FINDING	STATUS
KS-TTKEY-O-01	Informational	Key shares are searched across all polynomials stored in metadata	Remediated
KS-TTKEY-O-02	Informational	Ambiguous error message	Remediated
KS-TTKEY-O-03	Informational	Typo in error message	Remediated

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 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.

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. Where we expect the code to be reused across different applications with different input sanitization and parameters, we ranked some of the vulnerabilities' severity higher than usual

Correct memory management is left to TypeScript and was therefore not in scope. Zeroization of secret values from memory is also not enforceable at a low level in a language such as TypeScript.

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

3. TECHNICAL DETAILS OF SECURITY FINDINGS

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

3.1 Missing check that share indices are different

Finding ID: KS-CBBLS-F-01

Severity: Low

Status: Remediated

Location: `src/index.ts`

`src/lagrangeInterpolatePolynomial.ts`

Description and Impact Summary

During both the generation of new shares and the fetching of existing shares from stored metadata, it is never checked that share indices for the same polynomial are different. This is particularly troublesome for the `denominator` function, as it can return zero and hence trigger a division by zero during the Shamir secret reconstruction. This is never explicitly handled by error management and can thus lead to unexpected behaviour of the application.

It is important to note that, during normal operation, the chance of generating two random indices that are equal is vanishingly small. However, because the integrity of the (untrusted) storage layer is not enforced, an adversary could alter the value of the indices to cause not only a failure in the recovery of the secret, but also an unhandled exception that could potentially lead to further escalation. Notice: this should be avoided by the encryption layer, but in the specs it is not explicitly mandated for this encryption to provide integrity, just secrecy.

Recommendation

We recommend checking always that all the share indices are different whenever a polynomial is generated, fetched, and reconstructed. As an in-depth defense, explicitly require that the encryption layer on all the stored shares also provides integrity and non-malleability (ideally Authenticated Encryption, such as AES-GCM).

Status Details

This has been fixed as of commit `6304e475d95585c71991720511861b5e02875b3d`. Now, during the generation of a random polynomial structure, it is explicitly checked that no equal indices are generated, and the `interpolationPoly` function raises an error anyway if it encounters a zero denominator. Furthermore, an additional authentication layer has been added to the whole metadata using Keccac-256 and signature by the client's private key.

3.2 Missing check that share indices are not 0 or 1

Finding ID: KS-CBBLS-F-02

Severity: Low

Status: Remediated

Location: `src/index.ts`

`src/lagrangeInterpolatePolynomial.ts`

Description and Impact Summary

During both the generation of new shares and the fetching of existing shares from stored metadata, it is never checked that share indices for the same polynomial are not 0 or 1. This can lead to unexpected behaviour of the application, since the value 0 is reserved for the secret to be reconstructed, while the value 1 is reserved for the share kept by the Service Provider.

It is important to note that, during normal operation, the chance of generating a 0 or 1 index at random is vanishingly small. However, because the integrity of the (untrusted) storage layer is not enforced, an adversary could alter the value of the indices to cause not only a failure in the recovery of the secret, but also an unhandled exception that could potentially lead to further escalation. Notice: this should be avoided by the encryption layer, but in the specs it is not explicitly mandated for this encryption to provide integrity, just secrecy.

Recommendation

We recommend always checking that no share index equals 0 or 1 (except for the one kept by the Service Provider, which must be 1) whenever a polynomial is generated, fetched, and reconstructed. As an in-depth defense, explicitly require that the encryption layer on all the stored shares also provides integrity and non-malleability (ideally Authenticated Encryption).

Status Details

This has been fixed as of commit `97c85f8a25c704517300c7d058e5ca3b25910ba0`.

Now, during the generation of a random polynomial structure, it is explicitly checked that the indices 0 and 1 are not generated by mistake. Furthermore, an additional authentication layer has been added to the whole metadata using Keccac-256 and signature by the client's private key.

3.3 Missing check of correct decryption

Finding ID: KS-CBBLS-F-03

Severity: Low

Status: Remediated

Location: `src/index.ts` @line 713

Description and Impact Summary

The metadata on the storage layer is encrypted with the encKey provided by the service provider. When fetching the metadata, the user's device proceeds to decrypt it with encKey. However, the code does not seem to explicitly handle the case of a decryption failure.

```
    // decrypt and parsing
    const decryptedKeyStore = await this.decrypt(JSON.parse(keyStore));
    const newString = decryptedKeyStore.toString();
    const tkeyStoreData = JSON.parse(newString.toString()) as ISeedPhraseStore
    return tkeyStoreData;
```

This could lead to unexpected behavior of the application. Notice that this condition is relatively easy to trigger, depending on the storage layer: since the storage layer is untrusted, all an adversary has to do is to tamper with the encrypted metadata, and with high probability this would result in an incorrect ciphertext.

Recommendation

We recommend to explicitly consider (and to handle appropriately) the case where a decryption is unsuccessful. As an in-depth defense, explicitly require that the encryption layer on all the stored shares also provides integrity and non-malleability (ideally Authenticated Encryption).

Status Details

This has been fixed as of commit `97c85f8a25c704517300c7d058e5ca3b25910ba0`.

Now it is explicitly checked that decryption succeeds and if not, an error is returned. Furthermore, an additional authentication layer has been added to the whole metadata using Keccac-256 and signature by the client's private key.

3.4 Possible race condition

Finding ID: KS-CBBLS-F-04

Severity: Low

Status: Remediated

Location: `src/index.ts` @line 221

Description and Impact Summary

During secret reconstruction, the metadata field is scanned searching for shares for the correct polynomial. As soon as one of these shares is found, this is added to the pool of valid shares, the corresponding index is removed from the list of required indices, and a counter is decreased. The problem is that, in the code, these 3 steps actually happen in reverse order.

```
            if (latestShareRes.latestShare.polynomialID === pubPolyID) {
              sharesLeft -= 1;
              delete shareIndexesRequired[shareIndexesForPoly[k]];
              this.inputShare(latestShareRes.latestShare);
            }
```

In a correct execution of the application, this should never be a problem. However, if the application is stopped in the middle of execution (either maliciously or because of some error) the consequence might be an inconsistent state left in memory, which might for example introduce misleading traces during logging or debugging.

Recommendation

Perform the above operations in reverse order in respect to what is now done in the code.

Status Details

This has been fixed as of commit `6304e475d95585c71991720511861b5e02875b3d`.

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 Key shares are searched across all polynomials stored in metadata

Observation ID: KS-CBBLS-O-01

Location: `src/index.ts` @line 205

Description and Impact Summary

When reconstructing the secret from the shares stored in the metadata for a given polynomial, the metadata field is scanned through all the stored polynomials, so also the shares related to other polynomialIDs are searched.

```
    // we don't just check the latest poly but
    //  we check if the shares on previous polynomials in our stores have the
share indexes we require
```

This is not a security issue, as the shares are anyway encoded so that there is no ambiguity to which polynomial they belong to. However, we notice two possible drawbacks:

1) Performance hit in the case that the metadata field contains many shares for many different polynomials not related to the secret being reconstructed.
2) Having a share encoded for polynomial ID X being found in the metadata field for polynomial ID Y might indicate a possible problem.

Recommendation

It might be better to catch this behaviour and raise an explicit error if this happens.

Notes

This has been fixed as of commit `97c85f8a25c704517300c7d058e5ca3b25910ba0`.

Now an error is raised if a share is found inside an unexpected poly structure.

4.2 Ambiguous error message

Observation ID: KS-CBBLS-O-02

Location: `src/lagrangeInterpolatePolynomial.ts` @line 123

Description and Impact Summary

When deterministic shares are input by the user to generate a new secret polynomial, it is checked that there is still space left in the polynomial representation to include a secret of high enough entropy. If these deterministic shares are too many (for a given polynomial degree) an error is raised.

```
  if (determinsticShares.length > degree) {
    throw new TypeError("determinsticShares in generateRandomPolynomial need t
o be less than degree to ensure an element of randomness");
  }
```

However, the error message is slightly misleading: it should be “less *or equal* than degree”. In fact, for example a polynomial of degree two has three coefficients: one is the secret, and the other two can be both nondeterministic shares.

Recommendation

Simply fix the error message as suggested above.

Notes

This has been fixed as of commit `97c85f8a25c704517300c7d058e5ca3b25910ba0`.

4.3 Typo in error message

Observation ID: KS-CBBLS-O-03

Location: `src/index.ts` @line 693

Description and Impact Summary

There is a typo `hrase' in the error message.

```
throw new Error("Tkey store does not exist. Unable to delete seed hrase");
```

Recommendation

Replace with `phrase' or change the error message.

Notes

This has been fixed as of commit `97c85f8a25c704517300c7d058e5ca3b25910ba0`.

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	16 October 2020	Tommaso Gagliardoni	First draft
0.2	Draft	28 October 2020	Tommaso Gagliardoni	Proposal
1.0	Draft	28 October 2020	Tommaso Gagliardoni	Final version

REVIEWER	POSITION	DATE	VERSION	COMMENTS
Nathan Hamiel	Head of Security Research	19 October 2020	0.1	
Nathan Hamiel	Head of Security Research	28 October 2020	0.2	
Nathan Hamiel	Head of Security Research	28 October 2020	1.0	

APPROVER	POSITION	DATE	VERSION	COMMENTS
Nathan Hamiel	Head of Security Research	19 October 2020	0.1	
Nathan Hamiel	Head of Security Research	28 October 2020	0.2	
Nathan Hamiel	Head of Security Research	28 October 2020	1.0	

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.