Source Code Review

CoronaMelder, Android and iOS application

commissioned by

Ministerie van Volksgezondheid, Welzijn en Sport

MinVWS-Coronamelder-REPORT-v1.0.pdf
Ben Brücker, BSc. OSCP, GXPN, SEPP
Ralph Moonen, CISSP
Tim Hemel, MSc.
Tom Tervoort, MSc.
Matthijs Koot, Ph.D.
Yurii Bilyk, CEH
Dave Wurtz, OSCP, OSCE
Version 1.0
19 August 2020

Secura BV

Vestdijk 59
5611 CA EINDHOVEN
The Netherlands

Karspeldreef 8
1101 CJ AMSTERDAM
The Netherlands

T +31 (0)40 23 77 990

E info@secura.com

W https://www.secura.com

DOCUMENT MANAGEMENT

Reviewers

Name	Function	Date	Version
Matthijs Koot	Security Specialist	2020-08-19	0.1
Ralph Moonen	Technical Director	2020-08-19	1.0

Changes

Version	Date	Initials	Changes
0.1	2020-08-19	BB	Initial version
1.0	2020-08-19	BB	Internally reviewed final version

CONTENTS

1. MANAGEMENT SUMMARY

This document describes the results of the source code review of the CoronaMelder application that Secura performed at the request of the Ministerie van Volksgezondheid, Welzijn en Sport (hereafter MinVWS). This engagement was performed in July and August 2020 in accordance with relevant sections of testing standards such as OWASP Mobile Testing Guide, Mobile Application Security Verification Standard (M-ASVS) and relevant publications from MinVWS, in addition to Secura's own professional insight and expertise.

Prior to performing the source code review, a list of research questions was derived from the technical, security and privacy requirements as published by MinVWS. These questions are answered in chapter 2 on page 3. The scope of the review was defined as the Android and iOS source code as published on the repository of MinVWS, at the time of reviewing. Backend servers and systems were not a part of this review. Also not a part of the scope were the Google and Apple Exposure Notification (GAEN) functionalities of the mobile phone's operating systems. See below under 'Limitations'.

1.1. *Results of the review*

As a result of the source code review Secura has ascertained that the iOS and Android versions of the application adhere to security and privacy requirements. However, Secura found a number of minor inconsistencies while assessing the application's source code.

First, the iOS application makes use of a software library that implements transport security and other cryptographic functions, in a version (v1.1.1D) that is not the most recent version and is known to contain vulnerabilities. The vulnerable parts of the library are not used by the app however, therefore there is no impact to security or privacy. However, using the most recent version of libraries that do not contain any known vulnerabilities is always recommended, especially when future versions of the app might contain code that does call vulnerable parts of the library.

Second, some cryptographic signatures used to validate the keys that unlock exposure notifications (so-called Temporary Exposure Keys, or TEK s), are only partially checked. As a result, all holders of a certificate recently issued by KPN as part of the PKIOverheid service, could in theory produce valid signatures. This partially violates the integrity requirements defined in the architecture. However, because TEKs are protected in transit by a TLS connection and are also protected by an additional cryptographic GAEN-signature, Secura was not able to identify a scenario where this flaw could be practically exploited by an attacker.

Third, the app does not perform a check to see if it is running on a rooted or jailbroken device. Running any app on a rooted (Android) or jailbroken (iOS) device introduces security and privacy risks and can harm the integrity of all apps and communication. Not all users with a rooted or jailbroken device might be aware of this. A warning to users trying to install the app on a rooted or jailbroken device could make them aware of the additional risks.

Fourth, decoy messages are sent in order to make it more difficult for attackers to gather any meaningful information from any messages. However, when the app is disabled, decoy messages are still sent. This is not fully in compliance with the functional requirements and under specific circumstances could help an attacker identify users of the app even if it is disabled.

1.2. *Recommendations*

Secura BV recommends the following:

- Validate that the "Subject Name"-section of the TEKs signing certificate contains the correct "Common Name"-field (CN).
- Show a warning to users that attempt to install the app on jailbroken or rooted devices.
- Update the OpenSSL library to the newest version that does not include publicly known vulnerabilities.
- Disable sending of decoy messages when the app is disabled.

1.3. Conclusion

Minor improvements to the apps can be made that further strengthen the security and privacy measures already implemented. No material or significant deviations from the security and privacy requirements were identified, and no hidden, unwanted, malicious or suspect functionality was identified in the source code of the apps.

1.4. Limitations of the assessment

The main focus of this assessment was on security vulnerabilities and the impact on the privacy of its users. For this a source code review of the publicly available code was performed. Additionally, towards the end of the assessment, working applications were made available by MinVWS so that Secura could verify information from the source code. The following items were explicitly out of scope for this assessment:

- The underlying Operating System (iOS and Android) and the hardware (including Bluetooth) of the devices.
- The Apple/Google exposure notification API (GAEN).
- The source code of 3rd party software libraries.
- Licenses of 3rd party software libraries.
- The backend server.
- Maintainability of the codebase.

1.5. Final Notes

If MinVWS has any questions with regards to the assessment or presented results, or requires further clarifications, please contact Secura BV.

2. TECHNICAL SUMMARY

This chapter provides a technical overview of all findings in this report. More detailed information can be found in the actual findings later in this report.

2.1. Research questions

The ”Programma van eisen” as published by MinVWS includes a number of assumptions that cover the security and privacy requirements of the application. These assumptions are specifically tested in this report, listed as requirements in the following sections.

2.1.1. Requirements with inconsistencies

In the following requirements, inconsistencies have been found.

Requirement	[F5] The application can be temporarily disabled. When the user enables the application at a later moment it functions as normal. During a deactivated phase, no information will be collected or sent. Additionally, a user will receive a reminder to enable the application again.
Results	Decoy messages will still be sent, even if Exposure Notification is disabled.
Reference	See section 4.6.1 on page 22.

Requirement	[F8] The application removes contact codes that are older then 14 days.
Results	A minor inconsistency is that the application does not specify this timeframe, however, the 14 day deletion is implicit in the use of the underlying Google Apple Exposure Notification (GAEN).
Reference	See section 4.9 on page 24.

Requirement	[Q22] Data processed by the application can not be traced to an individual person.
Results	There are two ways how the backend can potentially identify users, both should be covered as discussed in the DPIA. However, since the backend was not in scope for this assessment, no assurances can be given.
Reference	See section 4.15 on page 36.

Requirement	Correct use of software libraries and library versions
Results	At the time of writing, a number of libraries were out of date. Specifically one OpenSSL library used by the iOS application, which was released on 10 September 2019. Since its release, two security bugs were made public that affect this version: CVE-2019-1551 and CVE-2020-1967. These bugs do not affect the security of the CoronaMelder app, because the library is only used to validate certificates and the bugs do not affect that process. It is nonetheless recommended to upgrade to the most current version, as a general best practice.
Reference	See section 4.21 on page 44.

Best Practice	Protection of the integrity of the application: No installation on rooted/jailbroken devices
Results	The application can be installed on rooted/jailbroken devices. This is in line with a concious choice made by the developers and discussed in the documentation. However, end users should be informed via a message of the risks of using the application on such a device.
Reference	See section 4.26 on page 53.

Additionally an inconsistency was found that was not an original research questions:

Best Practice	TEK signatures should not be forgeable by an attacker
Results	Any leaf certificate recently issued by KPN as part of the PKIOverheid service would be accepted by the application. If an attacker would manage to obtain a key corresponding to such a certificate, they could forge TEK signatures. However, no exploitable scenario could be devised for this weakness.
Reference	See section 4.23 on page 49.

2.1.2. Mapping of requirements

The following table lists the assessed requirements and the corresponding section of the report:

Requirement	**Reference**
U13: Statistical anonymous data can be collected, but only if this is required to monitor the effectiveness of the application.	4.3 on page 12
F3: The application gives the user an unique, anonymous contact code that changes multiple times per day.	4.4 on page 16
F4: The user can uninstall the application at any time. All data associated with the application should be removed from the device in this instance.	4.5 on page 16
F5: The application can be temporarily disabled. When the user enables the application at a later moment it functions as normal. During a deactivated phase, no information will be collected or sent. Additionally, a user will receive a reminder to enable the application again.	4.6 on page 18
F6: The application collects anonymous contact codes of all users during a configured timeframe in a configured distance. For example 15 minutes within 1,5 meters.	4.7 on page 23
F7: The criteria on which the application registers a contact can be configured in a central location.	4.8 on page 23
F8: The application removes contact codes that are older then 14 days.	4.9 on page 24
F9: The application collects periodically and incrementally contact codes from recent exposures from the server.	4.10 on page 30
F10: The frequency of collecting data from the server can be centrally configured.	4.11 on page 30
F11: The application warns a user automatically if there was a possible exposure, based on relevant contact (ie 15 minutes within 1,5 meters) with a confirmed infected person.	4.12 on page 31
Q13: When the application is unable to communicate because of networking issues, the user will be notified.	4.13 on page 32
Q14: When the application is unable to communicate because of loss of required technology, the user will be notified.	4.14 on page 35

Continues on the next page...

Table 2.1 (continued)

Requirement	Reference
Q22: Data processed by the application can not be traced to an individual person.	4.15 on page 36
Q23: The application does not use location data.	4.16 on page 39
Q26: Contact codes are not based on personal information.	4.17 on page 42
Q28: Use of the application is only possible after explicit consent by the user.	4.18 on page 42
Generation, storage and transmission of local log files	4.19 on page 43
Secure local storage of data and secure deletion of contact codes	4.20 on page 44
Protected communication with the backend servers	4.22 on page 46
Protection against Man-in-the-Middle attacks	4.22 on page 46
Correct use of the Google and Apple Exposure API's	4.24 on page 52
Correct use of software libraries and library versions	4.21 on page 44
Correct use of application permissions	4.25 on page 52
Protection of the integrity of the application: No installation on rooted/jailbroken devices	4.26 on page 53

Table 2.1: Overview of requirements

2.2. *Most Important Findings*

This section highlights the most important findings in this report.

Risk Note 1	**Decoy messages not disabled when Exposure notification is disabled.** CVSSv3 **3.7**
Description	The decoy upload scheduler does not seem to take the disabled status of the application in to account. While this is not a security vulnerability, it might be not intuitive for users of the application. For more information see section 4.6 on page 18.
Risk	The application possibly performs actions that are not in line with the end user's intention. Additionally, in a Man-in-the-Middle (MitM) scenario, an attacker might be able to detect whether this application is installed.
Recommendation	Also disable decoy uploads when the application is disabled.
Applies to	CoronaMelder application

Risk Note 2	**Outdated library used in iOS application: OpenSSL** CVSSv3 **0.0**
Description	OpenSSL version 1.1.1D is used by the iOS application, which was released on 10 September 2019. Since its release, two security bugs were made public that affect this version: CVE-2019-1551 and CVE-2020-1967. These bugs do not affect the security of the CoronaMelder app, because the library is only used to validate certificates and the bugs do not affect that process. It is nonetheless recommended to upgrade to the most current version, as a general best practice. For more information see the description in section 4.21 on page 44.
Risk	When this library would be used for TLS connections, it would be possible to perform a denial of service of a client under specific circumstances via the use of a publicly known vulnerability. In order to prevent issues in future iterations when this library might be used in another way, it is nonetheless recommended to upgrade to the most current version.
Recommendation	Upgrade the software to a version without known vulnerabilities.
Applies to	CoronaMelder application (iOS)

Risk Note 3	**The subject name in the certificate used for TEK signing is not checked.** CVSSv3 **1.9**
Description	When validating the second signature (using the certificate based CMS/PKCS#7 format) on a TEK list received by the server, the root and issuing CA certificates within the chain are checked. The subject name in the leaf certificate is not validated. See section 4.23 on page 49 for more information.
Risk	Any leaf certificate recently issued by KPN as part of the PKIOverheid service would be accepted by the `SignatureValidator` class. If an attacker would manage to obtain a key corresponding to such a certificate, they could forge TEK signatures. This violates the integrity requirements defined in the architecture documentation. Because TEKs are also protected in transit by TLS connection, and have additional protection due to the GAEN signature, it has not become clear during this assessment how this flaw could be practically exploited by an attacker.
Recommendation	Validate that the Subject of the signature certificate contains the correct Common Name (CN).
Applies to	CoronaMelder application

Remark 1	The backend can potentially link TEKs an individual person
Description	The backend — which itself is out of scope of this assessment — can perform IP stripping, as is shown the 'Backend overview' architecture diagram[1] If the IP stripping takes place *after* TLS offloading, the combination of the user's IP address and TEKs is accessible, at least temporarily while a request is being processed. This finding has not been verified on the backend since that was out of scope for this assessment. For more information see section 4.15.1 on page 36
Recommendation	One way to prevent this is to ensure the app does not communicate to the backend directly, but that the communication is brokered via an anonymising network, see the DPIA for more information. TLS encryption will still ensure confidentiality and integrity of the communication between the user and the backend.
Applies to	CoronaMelder application

Remark 2	The backend can potentially link confirmation keys or bucket id's to an individual
Description	An attacker who has access to the backend (prior to IP addresses being stripped) may be able to infer uniquely identifying patterns of times and IP addresses linked to a known bucket id or confirmation key. This finding has not been verified on the backend since that was out of scope for this assessment. For more information see section 4.15.1.1 on page 37.
Recommendation	One way to prevent this is to ensure the app does not communicate to the backend directly, but that the communication is brokered via an anonymising network, such as a custom/private onion routing network operated by entities who are independent from the government and the infrastructure hosting party. TLS encryption will still ensure confidentiality and integrity of the communication between the user and the backend.
Applies to	CoronaMelder application

Remark 4	No notification for installation on jailbroken / rooted devices
Description	The application does not detect if it is installed on a jailbroken/rooted device. All of the application's functionality can be used after installation on a jailbroken/rooted device. This is a concious decision as detailed in the architecture documentation. For more information see section 4.25 on page 52.
Recommendation	Detect if the application is installed on a jailbroken device and notify the user about the possible implications.
Applies to	CoronaMelder application

2.3. *Overview of Findings*

The following table contains links to all findings in this report. Note that these findings are clickable in the PDF to jump to the referenced page.

Finding	Topic	Reference
Risk Note 1	Decoy messages not disabled when Exposure notification is disabled.	Page 22
Risk Note 2	Outdated library used in iOS application: OpenSSL	Page 45
Risk Note 3	The subject name in the certificate used for TEK signing is not checked.	Page 52
Remark 1	The backend can potentially link TEKs an individual person	Page 37
Remark 2	The backend can potentially link confirmation keys or bucket id's to an individual	Page 39
Remark 3	Decoy traffic is only sent during office hours.	Page 39
Remark 4	No notification for installation on jailbroken / rooted devices	Page 54

Table 2.2: Summary of identified findings

3. DESCRIPTION OF THE ENGAGEMENT

This engagement is based on the proposal "20060524.02-MinVWS-Corona App" which includes the following description (in Dutch):

> *Het ministerie van Volksgezondheid, Welzijn en Sport is een Nederlands ministerie. Dit ministerie draagt in de eerste plaats de zorg voor de volksgezondheid.*
>
> *MinVWS wilt gebruik gaan maken van de Corona applicatie voor zowel iOS en Android. Het gebruik hiervan brengt IT-beveiligingsrisico's met zich mee. MinVWS wil deze risico's inzichtelijk maken en op een aanvaardbaar niveau brengen. Bovendien moet MinVWS voldoen aan wet- en regelgeving en bepaalde normen, zoals de AVG.*

3.1. Scope of the Assessment

The scope of the assessment is shown in table 3.1. No other systems or applications were assessed.

Identification	Description
`https://github.com/minvws/nl-covid19-notification-app-android/releases/tag/v1.0.0`	Version 1.0 of the Android source
`https://github.com/minvws/nl-covid19-notification-app-ios/releases/tag/1.0`	Version 1.0 of the iOS source

Table 3.1: Target systems and applications

Additionally, working applications were made available by MinVWS so that Secura can verify information from the source code.

The following items are explicitly out of scope for this assessment:

- The underlying Operating System (iOS and Android) and the hardware (including Bluetooth) of the devices
- The Apple/Google exposure notification API
- The source code of 3rd party software libraries
- Licenses of 3rd party software libraries
- The backend server
- Maintainability of the codebase

3.2. Information Provided

MinVWS has not provided Secura with additional information to perform this investigation. This is because the available information is public. The design and architecture documentation was provided in the following repositories:

- `https://github.com/minvws/nl-covid19-notification-app-design`
- `https://github.com/minvws/nl-covid19-notification-app-coordination`

3.3. Goal of the Assessment

The goal of this assignment was to independently determine the effectiveness of the security measures implemented to protect the CoronaMelder application, by identifying potential vulnerabilities in this application and to suggest possible improvements to its security. An additional goal was to test whether the CoronaMelder app function correctly without impacting a user's privacy.

3.4. Reporting

The management and technical summaries are intended to provide a high level overview for management and technical staff. The remaining chapters contain more detailed content with reproducible findings to support technical staff in reproduction and mitigation of the identified issues. Detailed scan results and supporting evidence is included in the appendices.

3.5. Approach

The assessment of the application was performed using the crystal-box approach. This approach uses various methods to find vulnerabilities. Automated tests examined the presence of known software packages to identify known vulnerabilities and configuration errors. Manual assessments were also performed to identify potential vulnerabilities that would have not been found through automation. Source code reviews identified specific vulnerabilities, such as logic errors, that would not have been found through the previous methods.

The research questions / assumptions as discussed in the previous chapter were used as a guideline to perform this assessment.

In a crystal-box investigation, extensive information is provided to Secura concerning the software, such as the source code, configuration files, log files, and internal documentation.

3.6. Limitations

A security assessment provides valuable insight into the security of the target system or application. However, it is a snapshot in time and does not provide assurance on the overall security level of the environment and the data. New types of attacks are discovered regularly and small changes in the environment can introduce new vulnerabilities. Processes and procedures, as well as human factors play at least as large a role as technology in information security. This report provides an overview of identified findings and is not an assurance report.

4. APPLICATION ASSESSMENT

The main focus of this source code assessment was to identify security vulnerabilities and to determine their impact to the privacy of users of the CoronaMelder app.

> *"Does the CoronaMelder app function correctly without impacting a user's privacy, in addition to the absence of security vulnerabilities.*

4.1. Application version

During the assessment there were multiple updates to the source code. This report is based on the Android and iOS code versions tagged 1.0, which can be found on the following locations:

- `https://github.com/minvws/nl-covid19-notification-app-android/releases/tag/v1.0.0`
- `https://github.com/minvws/nl-covid19-notification-app-ios/releases/tag/1.0`

4.2. Research questions

The "Programma van eisen" as published by MinVWS includes a number of assumptions that cover the security and privacy requirements of the application. These assumptions are specifically tested in this report, listed as requirements in the following subsection.

U13	Statistical anonymous data can be collected, but only if this is required to monitor the effectiveness of the application.
F3	The application gives the user an unique, anonymous contact code that changes multiple times per day.
F4	The user can uninstall the application at any time. All data associated with the application should be removed from the device in this instance.
F5	The application can be temporarily disabled. When the user enables the application at a later moment it functions as normal. During a deactivated phase, no information will be collected or sent. Additionally, a user will receive a reminder to enable the application again.
F6	The application collects anonymous contact codes of all users during a configured timeframe in a configured distance. For example 15 minutes within 1,5 meters.
F7	The criteria on which the application registers a contact can be configured in a central location.
F8	The application removes contact codes that are older then 14 days.
F9	The application collects periodically and incrementally contact codes from recent exposures from the server.
F10	The frequency of collecting data from the server can be centrally configured.
F11	The application warns a user automatically if there was a possible exposure, based on relevant contact (ie 15 minutes within 1,5 meters) with a confirmed infected person.
Q13	When the application is unable to communicate because of networking issues, the user will be notified.
Q14	When the application is unable to communicate because of loss of required technology, the user will be notified.
Q22	Data processed by the application can not be traced to an individual person.
Q23	The application does not use location data.
Q26	Contact codes are not based on personal information.
Q28	Use of the application is only possible after explicit consent by the user.

Additionally an assessment will be performed to determine whether an attacker can manipulate the application. The research questions for this are defined as follows:

- Generation, storage and transmission of local log files
- Secure local storage of data and secure deletion of contact codes
- Protected communication with the backend servers
- Protection against Man-in-the-Middle attacks
- Correct use of the Google and Apple Exposure API's
- Correct use of software libraries and library versions

- Correct use of application permissions

As mentioned in the scope, the following items are explicitly out of scope for this assessment:

- The underlying Operating System (iOS and Android) and the hardware (including Bluetooth) of the devices
- The Apple/Google exposure notification API
- The source code of 3rd party software libraries
- Licenses of 3rd party software libraries
- The backend server
- Maintainability of the codebase

4.3. U13 - Data collection

Requirement	U13: Statistical anonymous data can be collected, but only if this is required to monitor the effectiveness of the application.
Conclusion	Secura finds no inconsistencies to this requirement.

4.3.1. iOS

The app offers the following methods to log information, as defined in the `Logging` class:

- `logDebug`
- `logInfo`
- `logWarning`
- `logError`

Of these, the application only uses `logDebug` and `logError`. Additionally, the app uses the `CocoaLumberjack` library and configures it with a `DDOSLogger`, that uses the `os_log` call:

```
DDLog.add(DDOSLogger.sharedInstance) // Uses os_log
```

According to Apple's documentation[1], log messages at DEBUG level are not saved in the local storage.

It is possible to omit some data from logs by using the `%{private}` modifier, but the code does not use this. In fact, the `DDOSLogger` logs every message with the `%{public}` modifier under the hood:

```
vendor/CocoaLumberjack/Sources/CocoaLumberjack/DDOSLogger.m
102:                    os_log_error(logger, "%{public}s", msg);
106:                    os_log_info(logger, "%{public}s", msg);
111:                    os_log_debug(logger, "%{public}s", msg);
```

By default, the `os_log` method will redact dynamic strings and complex dynamic objects, but this modifier will not redact them. The app does not use this facility to redact sensitive data from log files. However, A quick inspection of calls to `logError` did not show sensitive information being logged.

Calls to logDebug however show information such as exposure reports and HTTP responses being logged. All can contain sensitive information. Since debug messages are not persisted, the only way for an attacker to see these would be to get access to someone's phone, run the application with debug logging enabled, and watch the logs via an USB connection.

The app also stores data. The `StorageController` defines two types of storage: `secure` and `insecure`. `secure` uses the keychain, while `insecure` uses the (sandboxed) filesystem, or memory. The keychain offers more possibilities to protect

[1]Source: `https://developer.apple.com/documentation/os/logging/generating_log_messages_from_your_code?language=occ`

data, such as requiring a user to unlock the data explicitly before the app can access it. The app configuration files did not mention anything keychain related, which means that it will not share the keychain with other apps.

`ExposureDataController` defines the following storages:

```
struct ExposureDataStorageKey {
static let labConfirmationKey = CodableStorageKey<LabConfirmationKey>
(name: "labConfirmationKey", storeType: .secure)
static let uploadedRollingStartNumbers = CodableStorageKey<[UInt32]>
(name: "uploadedRollingStartNumbers", storeType: .secure)
static let appManifest = CodableStorageKey<ApplicationManifest>
(name: "appManifest", storeType: .insecure(volatile: true))
static let appConfiguration = CodableStorageKey<ApplicationConfiguration>
(name: "appConfiguration", storeType: .insecure(volatile: true))
static let exposureKeySetsHolders = CodableStorageKey<[ExposureKeySetHolder]>
(name: "exposureKeySetsHolders", storeType: .insecure(volatile: false))
static let lastExposureReport = CodableStorageKey<ExposureReport>
(name: "exposureReport", storeType: .secure)
static let lastExposureProcessingDate = CodableStorageKey<Date>
(name: "lastExposureProcessingDate", storeType: .insecure(volatile: false))
static let lastLocalNotificationExposureDate = CodableStorageKey<Date>
(name: "lastLocalNotificationExposureDate", storeType: .insecure(volatile: false))
static let lastENStatusCheck = CodableStorageKey<Date>
(name: "lastENStatusCheck", storeType: .insecure(volatile: false))
static let exposureConfiguration = CodableStorageKey<ExposureRiskConfiguration>
(name: "exposureConfiguration", storeType: .insecure(volatile: false))
static let pendingLabUploadRequests = CodableStorageKey<[PendingLabConfirmationUp
    loadRequest]>
(name: "pendingLabUploadRequests", storeType: .secure)
static let firstRunIdentifier = CodableStorageKey<Bool>
(name: "firstRunIdentifier", storeType: .insecure(volatile: false))
static let exposureApiCallDates = CodableStorageKey<[Date]>
(name: "exposureApiCalls", storeType: .insecure(volatile: false))
}
```

These are the only storages that the StorageController uses. None of the data in storage marked `insecure` contains information that one can directly relate to an exposure.

The production version of the app uses the following entitlements:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>com.apple.developer.default-data-protection</key>
        <string>NSFileProtectionCompleteUntilFirstUserAuthentication</string>
        <key>aps-environment</key>
        <string>production</string>
        <key>com.apple.developer.exposure-notification</key>
        <true/>
</dict>
</plist>
```

The data protection setting means that only this app can access the file. According to `https://wojciechkulik.pl/ios/data-protection-on-ios-security-overview`:

> *The class key is protected with a key derived from the user passcode and the device UID. This class behaves in the same way as Complete Protection, except that the decrypted class key isn't removed from memory when the device is locked.*

4.3.2. Android

The application uses uses WorkManager diagnostics, according to the Android Manifest:

```
        [...]
        <receiver android:name="androidx.work.impl.diagnostics.DiagnosticsReceiver"
 android:permission="android.permission.DUMP" android:enabled="true" android:
exported="true" android:directBootAware="false">
            <intent-filter>
                <action android:name="androidx.work.diagnostics.REQUEST_DIAGNOSTICS
"/>
            </intent-filter>
        </receiver>
[...]
</manifest>
```

The following command was run in order to obtain diagnostics data[2]:

```
adb shell am broadcast -a "androidx.work.diagnostics.REQUEST_DIAGNOSTICS" -p "nl.
rijksoverheid.en"
Broadcasting: Intent { act=androidx.work.diagnostics.REQUEST_DIAGNOSTICS pkg=nl.
rijksoverheid.en }
Broadcast completed: result=0
```

The following data was obtaining by running `adb logcat`:

```
Process nl.rijksoverheid.en created for broadcast nl.rijksoverheid.en/androidx.work
.impl.diagnostics.DiagnosticsReceiver

PID: 26167   UID:    GIDs:

                 System  W  ClassLoader referenced unknown path: /data/app/nl.
rijksoverheid.en-1/lib/x86
          EnApplication  D  onCreate
     WM-DiagnosticsRcvr  D  Requesting diagnostics
WM-PackageManagerHelper  D  androidx.work.impl.background.systemjob.
SystemJobService enabled
          WM-Schedulers  D  Created SystemJobScheduler and enabled SystemJobService
   WM-ForceStopRunnable  D  Performing cleanup operations.
WM-PackageManagerHelper  D  androidx.work.impl.background.systemalarm.
RescheduleReceiver enabled
  WM-SystemJobScheduler  D  Scheduling work ID f9c02602-db1e-4abc-9520-270537b92cb1
 Job ID 0
     WM-GreedyScheduler  D  Starting work for f9c02602-db1e-4abc-9520-270537b92cb1
           WM-Processor  D  Processor: processing f9c02602-db1e-4abc-9520-270537
b92cb1
```

[2]https://developer.android.com/jetpack/androidx/releases/work

```
       WM-WorkerWrapper  D  Starting work for androidx.work.impl.workers.
DiagnosticsWorker
     WM-DiagnosticsWrkr  I  Running work:
                         I   Id       Class Name      Job Id      State      Unique
Name      Tags
                         I  f9c02602-db1e-4abc-9520-270537b92cb1      androidx.work.
impl.workers.DiagnosticsWorker      0      RUNNING          androidx.work.impl.
workers.DiagnosticsWorker
       WM-WorkerWrapper  D  androidx.work.impl.workers.DiagnosticsWorker returned a
 Success {mOutputData=Data {}} result.
                         I  Worker result SUCCESS for Work [ id=f9c02602-db1e-4abc
-9520-270537b92cb1, tags={ androidx.work.impl.workers.DiagnosticsWorker } ]
WM-PackageManagerHelper  D  androidx.work.impl.background.systemalarm.
RescheduleReceiver disabled
           WM-Processor  D  Processor f9c02602-db1e-4abc-9520-270537b92cb1 executed
; reschedule = false
     WM-GreedyScheduler  D  Cancelling work ID f9c02602-db1e-4abc-9520-270537b92cb1
           WM-Processor  D  Processor stopping background work f9c02602-db1e-4abc
-9520-270537b92cb1
                         D  WorkerWrapper could not be found for f9c02602-db1e-4abc
-9520-270537b92cb1
    WM-StopWorkRunnable  D  StopWorkRunnable for f9c02602-db1e-4abc-9520-270537
b92cb1; Processor.stopWork = false
     WM-DiagnosticsRcvr  D  Requesting diagnostics
WM-PackageManagerHelper  D  androidx.work.impl.background.systemalarm.
RescheduleReceiver enabled
  WM-SystemJobScheduler  D  Scheduling work ID 0d09beba-3499-45dc-a68d-1e1fbf0c16fb
 Job ID 1
     WM-GreedyScheduler  D  Starting work for 0d09beba-3499-45dc-a68d-1e1fbf0c16fb
           WM-Processor  D  Processor: processing 0d09beba-3499-45dc-a68d-1
e1fbf0c16fb
       WM-WorkerWrapper  D  Starting work for androidx.work.impl.workers.
DiagnosticsWorker
     WM-DiagnosticsWrkr  I  Recently completed work:
                         I   Id       Class Name      Job Id      State      Unique
Name      Tags
                         I  f9c02602-db1e-4abc-9520-270537b92cb1      androidx.work.
impl.workers.DiagnosticsWorker      null      SUCCEEDED          androidx.work.impl.
workers.DiagnosticsWorker
                         I  Running work:
                         I   Id       Class Name      Job Id      State      Unique
Name      Tags
                         I  0d09beba-3499-45dc-a68d-1e1fbf0c16fb      androidx.work.
impl.workers.DiagnosticsWorker      1      RUNNING          androidx.work.impl.
workers.DiagnosticsWorker
       WM-WorkerWrapper  D  androidx.work.impl.workers.DiagnosticsWorker returned a
 Success {mOutputData=Data {}} result.
                         I  Worker result SUCCESS for Work [ id=0d09beba-3499-45dc-
a68d-1e1fbf0c16fb, tags={ androidx.work.impl.workers.DiagnosticsWorker } ]
WM-PackageManagerHelper  D  androidx.work.impl.background.systemalarm.
RescheduleReceiver disabled
           WM-Processor  D  Processor 0d09beba-3499-45dc-a68d-1e1fbf0c16fb executed
; reschedule = false
```

```
     WM-GreedyScheduler  D  Cancelling work ID 0d09beba-3499-45dc-a68d-1e1fbf0c16fb
           WM-Processor  D  Processor stopping background work 0d09beba-3499-45dc-
a68d-1e1fbf0c16fb
                         D  WorkerWrapper could not be found for 0d09beba-3499-45dc
-a68d-1e1fbf0c16fb
    WM-StopWorkRunnable  D  StopWorkRunnable for 0d09beba-3499-45dc-a68d-1
e1fbf0c16fb; Processor.stopWork = false
```

No personal or sensitive information was observed in the collected diagnostic data. Additionally, no other sensitive diagnostics data was recovered from the local storage of the application.

4.4. F3 - Anonymous contact codes

Requirement	F3: The application gives the user an unique, anonymous contact code that changes multiple times per day.
Conclusion	Secura finds no inconsistencies to this requirement.

This functionality is completely covered by the correct use of the Google/Apple Exposure Notification (GAEN). Also, no other code was discovered that attempted to create these contact codes. Therefore it is out of scope for this assessment.

4.5. F4 - Secure uninstallation

Requirement	F4: The user can uninstall the application at any time. All data associated with the application should be removed from the device in this instance.
Conclusion	Secura finds no inconsistencies to this requirement.

4.5.1. iOS

The filesystem locations where the app stores its data depends on whether the data is classified as volatile or not. The code for this is:

```
    private func storeUrl(isVolatile: Bool) -> URL? {
        let base = isVolatile ? localPathProvider.path(for: .cache) :
localPathProvider.path(for: .documents)

        return base?.appendingPathComponent("store")
    }
```

The class LocalPathProvider defines these locations, to be the application's cache directory (volatile) and documents directory (non-volatile). Both locations are correctly deleted when uninstalling the application.

4.5.2. Android

Android also stores all data in the app's local sandbox directory which is structured as follows:

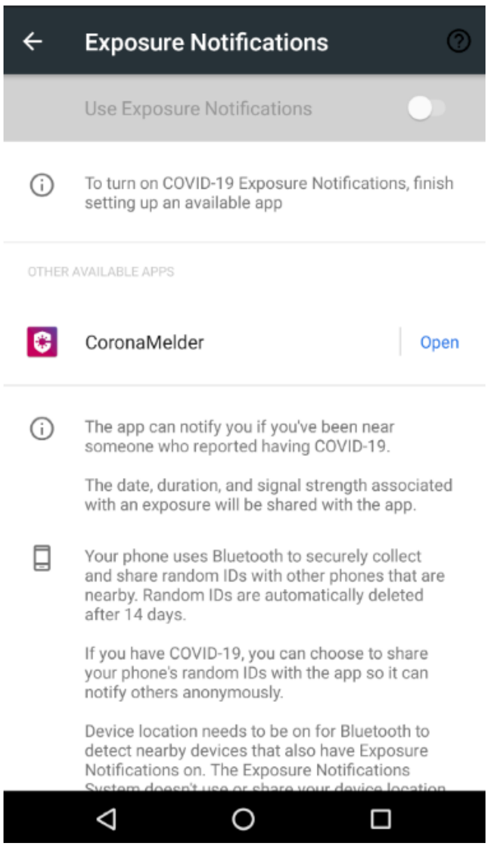
```
tree /data/data/nl.rijksoverheid.en
nl.rijksoverheid.en
|-- cache
|   `-- http
|       |-- 459c7bf8530fbb9e7af2fa5759a09d85.0
|       |-- 459c7bf8530fbb9e7af2fa5759a09d85.1
|       |-- cca3b8d69f895fb355b4b6d0332abcb8.0
|       |-- cca3b8d69f895fb355b4b6d0332abcb8.1
|       `-- journal
|-- code_cache
|-- databases
|-- no_backup
|   `-- androidx.work.workdb
`-- shared_prefs
    |-- nl.rijksoverheid.en.cache.xml
    |-- nl.rijksoverheid.en.notifications.xml
    |-- nl.rijksoverheid.en.onboarding.xml
    `-- nl.rijksoverheid.en.testphase.xml
```

Sensitive data in the shared preferences was stored in an encrypted fashion:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="__androidx_security_crypto_encrypted_prefs_value_keyset__">1286
    [...]
    676c65617069732e636f6d2f676f6f676c652e63727970746f2e74696e6b2e41657347636d4b6
    579100118dee5d85d2001</string>
    <string name="__androidx_security_crypto_encrypted_prefs_key_keyset__">12a901
    [...]
    17069732e636f6d2f676f6f676c652e63727970746f2e74696e6b2e4165735369764b65791001
    18d9e5c4c7072001</string>
</map>
```

No data in other location was discovered. This is also supported by the fact that there is no `WRITE_EXTERNAL_STORAGE` permission defined in the Android Manifest.

The application was also registered as available for the Exposure Notifications:



Exposure Notifications

Use Exposure Notifications

To turn on COVID-19 Exposure Notifications, finish setting up an available app

OTHER AVAILABLE APPS

CoronaMelder Open

The app can notify you if you've been near someone who reported having COVID-19.

The date, duration, and signal strength associated with an exposure will be shared with the app.

Your phone uses Bluetooth to securely collect and share random IDs with other phones that are nearby. Random IDs are automatically deleted after 14 days.

If you have COVID-19, you can choose to share your phone's random IDs with the app so it can notify others anonymously.

Device location needs to be on for Bluetooth to detect nearby devices that also have Exposure Notifications on. The Exposure Notifications System doesn't use or share your device location.

Figure 4.1: Exposure Notifications settings

The application was uninstalled via the standard Android uninstallation process. All data from the `/data/data/nl.rijksoverheid.en` directory was removed. Also the application was removed from the Exposure Notifications settings.

4.6. F5 - Temporary disabling the application

Requirement	The application can be temporarily disabled. When the user enables the application at a later moment it functions as normal. During a deactivated phase, no information will be collected or sent. Additionally, a user will receive a reminder to enable the application again.
Conclusion	Decoy messages will still be sent, even if Exposure Notification is disabled.

The user can disable the app via the settings menu, according to the user interface design linked from `https://github.com/minvws/nl-covid19-notification-app-design`. Also the texts for the corresponding screen suggest to open the settings:

```
"enableSettings.exposureNotifications.title" = "Enable contact notifications";
"enableSettings.exposureNotifications.action" = "Open Settings";
"enableSettings.exposureNotifications.step1" = "Open <b>Settings</b> on your phone
using the button below";
```

```
"enableSettings.exposureNotifications.step2" = "Turn on contact notifications";
"enableSettings.exposureNotifications.step2.action.title" = "Notifications about
exposure to COVID-19";
```

The button opens the settings via the URL `UIApplication.openSettingsURLString`. The app introduces a setting named `exposureNotifications`.The app maintains an `ExposureState` object in the `ExposureStateStream` class. The field `currentExposureState` gives the exposure state at that moment. The `ExposureState` class has two substates:

```
struct ExposureState: Equatable {
    let notifiedState: ExposureNotificationState
    let activeState: ExposureActiveState
}
```

The first has just two values: 'notified + the date of notification', or 'not notified'

```
enum ExposureNotificationState: Equatable {
    case notified(Date)
    case notNotified
}
```

The active state has four main values:

```
enum ExposureActiveState: Equatable {
    /// Exposure Notification is active
    case active

    /// Exposure Notification is inactive, inactiveState contains the reason why
    case inactive(ExposureStateInactiveState)

    /// No authorisation has been given yet
    case notAuthorized

    /// Authorisation has been explicitly denied
    case authorizationDenied
}
```

If the state is `inactive`, there can be several substates:

```
enum ExposureStateInactiveState: Equatable {
    case disabled
    case bluetoothOff
    case pushNotifications
    case noRecentNotificationUpdates
}
```

If the user explicitly disables the app, the inactive state `disabled` would be true. `bluetoothOff` means Bluetooth has been disabled, `pushNotifications` means push notifications are disabled, and `noRecentNotificationUpdates` means that the app did not update its information recently.

The following other source files refer to `currentExposureState`:

```
Sources/ENCore/App/Features/Main/Status/StatusViewController.swift
Sources/ENCore/App/Features/Main/MainViewController.swift
Sources/ENCore/App/Features/Onboarding/Models/OnboardingConsentManager.swift
Sources/ENCore/App/Features/DeveloperMenu/DeveloperMenuViewController.swift
Sources/ENCore/App/Features/ExposureController/ExposureController.swift
```

4.6.0.1. MainViewController.swift

When the main view is loaded, the method `viewDidLoad` is called. This checks if the current state is `disabled` and if so, it will request the user to give permission via `ExposureController.requestExposureNotificationPermission`. Therefore, the user will get a notification to enable the app if it is disabled.

`handleUpdateAppSettings` will show the screen corresponding to the button `updateAppSettings`. It looks at the exposure state. In case the app is inactive because Bluetooth is disabled, the app will ask to enable Bluetooth. If the user disabled it, it will ask the user to enable the app. If push notifications are enabled, it will ask to enable push notifications.

If there were no recent notification updates, the app will try to update via the local method `updateWhenRequired` (see section 4.6.0.5 on the following page). Other reasons why the app could be inactive could be the result of programming errors, but this code still detects such a situation and logs an error. If the app is not authorised yet, it will ask the user for permission via `requestExposureNotificationPermission`. If authorisation is denied, it will go to the screen `enableExposureNotifications`. If the app is active, this method will do nothing.

4.6.0.2. StatusViewController.swift

When the status view finished loading, `viewDidLoad` is called. This looks at the current exposure state and calls the local method `update`. Depending on the current exposure state, this method selects the corresponding status view screen and will show it.

4.6.0.3. OnboardingConsentManager.swift

This class, used during the onboarding phase, uses the current exposure state to detect if Bluetooth is enabled, but also to ask to enable exposure notifications. If the user has authorised the app, it will continue. If the user has not authorised the app, it will ask the user for permission via `ExposureController.requestExposureNotificationPermission`, and wait until the user has authorised the app. Note that it does not check for the state `disabled`, as this state is only relevant *after* onboarding.

4.6.0.4. DeveloperMenuViewController.swift

The developer menu is only available is the app is built with the build option `USE_DEVELOPER_MENU`. It allows a developer to change settings easily for testing purposes.

The developer menu shows the exposure state and notified state among other things. It looks at the current exposure state to display it, and lets the user change the state via `changeExposureState` and `changeNotified`. It offers to set the state to one of the following values:

```
.active,
.authorizationDenied,
.inactive(.bluetoothOff),
.inactive(.disabled),
.inactive(.noRecentNotificationUpdates)
```

The method `updateExposureState` will change the current exposure state and refresh the developer menu options.

4.6.0.5. ExposureController.swift

This class' method `updateWhenRequired` looks at the current exposure state. If the app is active or inactive because of no recent updates, it will call `fetchAndProcessExposureKeySets`, otherwise it will give an error.

This means that the app will not load any exposure keysets while it is disabled or does not have authorisation.

4.6.0.6. Other functionality

Uploading TEKs to the back-end happens interactively if the user chooses to do so. While the app is disabled, this functionality would be hidden from any of the screens, and therefore the app will not accidentally upload any keys.

The *decoy uploads* however happen in the background. The method `scheduleTasks` asks the `ExposureController` if the app is deactivated via `isAppDectivated` (sic) and will not schedule any background tasks if it is. In fact, it will then remove all tasks.

`scheduleTasks` is called once from the `RootRouter` class in the method `didEnterBackground`, called whenever the application is sent to run in the background.

If the app re-enters the foreground or becomes active, it will call `refreshStatus` on the `ExposureController`. This will call `updateStatusStream` and `updatePushNotificationState`.

`isAppDectivated` calls the method of the same name on `ExposureDataController`, which checks the application configuration via `requestApplicationConfiguration` and looks at the `decativated` (sic) attribute.

`requestApplicationConfiguration` in turn calls `requestApplicationManifest` and gets the configuration section from it via the operation `requestAppConfigurationOperation`. The manifest itself is retrieved via the operation `requestManifestOperation`.

Both operations are implemented in the classes `RequestAppManifestDataOperation` and `RequestAppConfigurationDataOperation` respectively. In this case, the application configuration is the stored configuration containing the server dictated settings.

Neither the attribute `decativated`, nor the attribute `coronaMelderDeactivated`, from which it is copied, are documented in the backend API specification (see `https://petstore.swagger.io/?url=https://raw.githubusercontent.com/minvws/nl-covid19-notification-app-coordination/master/architecture/api/apispec.yaml`).

Decoy updates appear to continue working even if the application is locally disabled. This breaks requirement F5. One could argue that this is to prevent an attacker to analyze traffic and see whether someone has disabled the app, but then the app should also continue all other operations, or at least fake them, which it does not.

Other background tasks that also appear to be active even if the user has disabled the app, are scheduling updates and status checks, via the methods `scheduleUpdate` and `scheduleENStatusCheck` respectively.

`scheduleENStatusCheck` schedules a task to run after one hour. The method `handle` on the `BackgroundController` handles the exposure notification status check via `handleENStatusCheck` and after that, schedules it again.

It checks the exposure notification status via the `ExposureManager`'s `getExposureNotificationStatus` method. There is some synchronisation between the two via the `ExposureController` class.

The `ExposureController` has methods `activate` and `deactivate`, which call the `ExposureManager`'s method of the same name. Only the `activate` method updates the status stream, by calling the `ExposureManager`'s method `getExposureNotificationStatus` (which calls the GAEN library) and converts that result into a value to put on the stream. The `ExposureManager` class is a wrapper around the GAEN library, imported as `ExposureNotification`. It also has methods `activate` and `deactivate`, that call the GAEN library to enable or disable the exposure notification.

Back in `handleEnStatusCheck`, if the state is already active, it skips the check. It looks at the `ExposureController`'s method `lastENStatusCheckDate`, which gets the `lastENStatusCheck` object from the `storageController`. If it cannot find this, it will call `notifyUserENFrameworkDisabled` If the last checkdate is less than 24 hours ago, the task is completed successfully, otherwise, it calls `notifyUserEnFrameworkDisabled`. This methods sends a push notification to the user, and calls `setLastEndStatusCheckDate` on the `ExposureController` with the current date. This sets the `lastENStatusCheck` object in the `storageController` to the current date.

This background tasks effectively checks every hour if the application has not been active for more than 24 hours.

`scheduleUpdate` follows a similar pattern, but the interval is configurable. Both `scheduleUpdate` and `handleUpdate` ask the `ExposureManager` for its `authorizationStatus` property. If the `ExposureManager` (and indirectly the GAEN library) says the state is not authorised, no update task is scheduled.

`handleUpdate` looks at a stream of functions, which it will call. This stream is initialised with two events:

```
exposureController.updateWhenRequired,
exposureController.processPendingUploadRequests
```

`updateWhenRequired` is discussed in section 4.6.0.5 on the previous page.

`processPendingUploadRequests` looks at the application configuration via `requestApplicationConfiguration`, finds its padding parameters and uses them to via `processPendingLabConfirmationUploadRequestsOperation`.

The `ProcessPendingLabConfirmationUploadRequestsDataOperation` operation processes a list of pending upload requests via the method `uploadPendingRequest`, which calls the `NetworkController`'s `postKeys` method.

4.6.1. *Submission of Decoys while application is disabled*

On both iOS and Android, the decoy upload scheduler does not seem to take the disabled status of the application in to account. While this is not a security vulnerability, it might be not intuitive for users of the application.

RISK NOTE 1	20060524-N1
Topic	Decoy messages not disabled when Exposure notification is disabled.
Applies to	CoronaMelder application
Description	The decoy upload scheduler does not seem to take the disabled status of the application in to account. While this is not a security vulnerability, it might be not intuitive for users of the application. For more information see section 4.6 on page 18.
Risk	The application possibly performs actions that are not in line with the end user's intention. Additionally, in a Man-in-the-Middle (MitM) scenario, an attacker might be able to detect whether this application is installed.
Recommendation	Also disable decoy uploads when the application is disabled.
Reproduction	See the description in section 4.6 on page 18
Metadata	Likelihood: Low, Impact: Low, CVSS-Score:3.7 CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

4.7. F6 - Collection of anonymous contact codes

Requirement	The application collects anonymous contact codes of all users during a configured timeframe in a configured distance. For example 15 minutes within 1,5 meters.
Conclusion	Secura finds no inconsistencies to this requirement.

The GAEN library takes care of this. The app itself will not collect contact codes, it can only see which exposure TEKs have been in contact with the phone, by asking the GAEN library for those keys, according to certain exposure criteria. Secura discovered no vulnerabilities in the source code where these GAEN APIs are called. See section 4.9 on the following page.

4.8. F7 - Central configuration

Requirement	The criteria on which the application registers a contact can be configured in a central location.
Conclusion	Secura finds no inconsistencies to this requirement.

The central API offers a call to `/riskcalculationparameters/{id}` as defined in the documentation[3].

4.8.1. iOS

The application manifest in the `ApplicationManifest` struct show the following fields, including risk calculation parameters:

```
struct ApplicationManifest: Codable {
    let exposureKeySetsIdentifiers: [String]
    let riskCalculationParametersIdentifier: String
    let appConfigurationIdentifier: String
    let creationDate: Date
    let iOSMinimumKillVersion: String?
}
```

The `detectExposures` method in the `ENManager` class uses an `ENExposureConfiguration` object. This contains the following fields:

```
protocol ExposureConfiguration {
    var minimumRiskScope: UInt8 { get }
    var attenuationLevelValues: [UInt8] { get }
    var daysSinceLastExposureLevelValues: [UInt8] { get }
    var durationLevelValues: [UInt8] { get }
    var transmissionRiskLevelValues: [UInt8] { get }
    var attenuationDurationThresholds: [Int] { get }
}
```

`detectExposures` is part of the `ExposureNotification` API[4].

The class `RequestExposureConfigurationDataOperation` contains code to update and store this configuration.

[3]Source: https://raw.githubusercontent.com/minvws/nl-covid19-notification-app-coordination/master/architecture/api/apispec.yaml
[4]Source: https://developer.apple.com/documentation/exposurenotification/enmanager/3586331-detectexposures

4.8.2. Android

Within the Android source code, a similar function is defined that sets these values:

```
private suspend fun getConfigurationFromManifest(manifest: Manifest):
ExposureConfiguration {
    val riskCalculationParameters =
        api.getRiskCalculationParameters(manifest.riskCalculationParametersId)
    return ExposureConfiguration.ExposureConfigurationBuilder()
        .setDurationAtAttenuationThresholds(*riskCalculationParameters.
durationAtAttenuationThresholds.toIntArray())
        .setMinimumRiskScore(riskCalculationParameters.minimumRiskScore)
        .setTransmissionRiskScores(*riskCalculationParameters.
transmissionRiskScores.toIntArray())
        .setDurationScores(*riskCalculationParameters.durationScores.toIntArray())
        .setAttenuationScores(*riskCalculationParameters.attenuationScores.
toIntArray())
        .setDaysSinceLastExposureScores(*riskCalculationParameters.
daysSinceLastExposureScores.toIntArray())
        .build()
}
```

This data is in turn used in the `processExposureKeySets` function in order to calculate a possible exposure based on the given values.

4.9. F8 - Removal of old data

Requirement	F8: The application removes contact codes that are older then 14 days.
Conclusion	A minor inconsistency is that the application does not specify this timeframe, however, the 14 day deletion is implicit in the use of the underlying Google Apple Exposure Notification (GAEN).

4.9.1. iOS

The `ENManager` documentation (`https://developer.apple.com/documentation/exposurenotification/enmanager`) lists the following methods/fields related to exposures:

- `detectExposures`
- `getExposureWindows`
- `ENGetExposureWindowsHandler`
- `getUserTraveled`
- `ENGetUserTraveledHandler`
- `getExposureInfo` (deprecated)

There are calls to `detectExposure` and to the deprecated method `getExposureInfo`, but the code does not refer to the other methods or fields. The GAEN library will only give information about exposed contact codes to the app. For that reason, the method needs a list of downloaded diagnostic keys (i.e., the list of exposure TEKs and exposure criteria. To be compatible with iOS 13.5 or lower, the GAEN library on iOS has a call limit of 15 times per day. But this value can be overwritten:

```
ENCore/App/Features/DeveloperMenu/DeveloperMenuViewController.swift
414:            return ProcessExposureKeySetsDataOperationOverrides.
respectMaximumDailyKeySets ? "15" : "unlimited"
```

4.9.1.1. *Calling detectExposures*

The `ProcessExposureKeySetsDataOperation`'s method `detectExposures` calls the GAEN library's method of the same name.

If the flag `fakeProcessKeySets` is set, will just mark all keys as processed. In the real processing, it will first look for the local files of a keyset. For this, the signature and the binary files must exist. These file URLs are found via the methods `signatureFileUrl` and `binaryFileUrl` respectively. These are the downloaded keys against which the GAEN library will check exposures.

The call will return exposure results, which the app will transform into `ExposureKeySetDetectionResult` objects, and later into `ExposureDetectionResult` objects. These objects contain the following fields:

```
private struct ExposureKeySetDetectionResult {
    let keySetHolder: ExposureKeySetHolder
    let processDate: Date?
    let isValid: Bool
}

private struct ExposureDetectionResult {
    let keySetDetectionResults: [ExposureKeySetDetectionResult]
    let exposureSummary: ExposureDetectionSummary?
    let exposureReport: ExposureReport?
}
```

An `ExposureKeySetHolder` contains:

```
struct ExposureKeySetHolder: Codable {
    let identifier: String
    let signatureFilename: String
    let binaryFilename: String
    let processDate: Date?
    let creationDate: Date

    var processed: Bool { processDate != nil }
}
```

None of these data structures contain contact codes. The results of the exposure detection will only show with which exposure TEK there has been contact.

4.9.1.2. *Calling getExposureInfo*

According to the iOS GAEN API, `getExposureInfo` returns an `ENExposureDetectionSummary`[5]. This object contains details about the exposures, but does not appear to list any of the contact codes. The field `metadata` is, according to the API documentation, not used.

4.9.1.3. *Persistence of Exposure Results*

In the class `ProcessExposureKeySetsDataOperation`, described in section 4.9.1.1, the execute method calls the methods `persist` and `persistResult`, which stores the exposure results. The former stores the exposure report, while the latter updates the currently stored keys with the results.

[5]Source: `https://developer.apple.com/documentation/exposurenotification/enexposuredetectionsummary`

4.9.2. Removal of exposure TEKs

In the interpretation of contact codes, being the exposure TEKs that the app downloads, the app does indeed store these. There is a method `removeBlobs` that removes the binary files and signatures for keys that are processed or for which the exposure check failed. After the app stored the exposure report, it calls this method. There is no explicit time period in this method.

In `RequestExposureKeySetsDataOperation`'s method `createKeySetHolders`, a file with exposed keys is removed if it has the same name as a newly downloaded file.

4.9.3. Android

The application uses Google Exposure API for the purpose of contacts tracing[6]. The `resetExposures` function removes previously stored contact codes:

```
nl-covid19-notification-app-android/app/src/main/java/nl/rijksoverheid/en/
ExposureNotificationsRepository.kt
[...]

fun resetExposures() {
        preferences.edit {
            // Use putString instead of remove, otherwise encrypted shared
preferences don't call
            // an associated shared preferences listener.
            putString(KEY_LAST_TOKEN_ID, null)
            putString(KEY_LAST_TOKEN_EXPOSURE_DATE, null)
        }
    }
```

The `resetExposures` function is called by the function `removeExposure`:

```
nl-covid19-notification-app-android/app/src/main/java/nl/rijksoverheid/en/status/
StatusViewModel.kt
[...]

 fun removeExposure() {
        exposureNotificationsRepository.resetExposures()
    }

[...]
```

The `removeExposure` function is called by another function `showRemoveNotificationConfirmationDialog`:

```
nl-covid19-notification-app-android/app/src/main/java/nl/rijksoverheid/en/status/
StatusFragment.kt
[...]

private fun showRemoveNotificationConfirmationDialog() {
        findNavController().navigate(StatusFragmentDirections.
actionRemoveExposedMessage())
        findNavController().currentBackStackEntry?.savedStateHandle?.getLiveData<
Boolean>(
```

[6]https://github.com/google/exposure-notifications-android

```
            RemoveExposedMessageDialogFragment.REMOVE_EXPOSED_MESSAGE_RESULT
        )?.observe(viewLifecycleOwner) {
            if (it) {
                statusViewModel.removeExposure()
            }
        }
    }
[...]
```

However, no indication of automatic removal after 14 days were found in the application code. The 14 days periods is handled by the GAEN API and is not controlled by the application.[7]. The GAEN API allows to upload history of contacts code not older than 14 days by using `getTemporaryExposureKeyHistory` function. Override of this function was only found in the test code:

```
nl-covid19-notification-app-android/en-api/src/test/java/nl/rijksoverheid/en/enapi/
nearby/NearbyExposureNotificationApiTest.kt
[...]
334:                override fun getTemporaryExposureKeyHistory(): Task<List<
TemporaryExposureKey>> =
362:                override fun getTemporaryExposureKeyHistory(): Task<List<
TemporaryExposureKey>> =
381:                override fun getTemporaryExposureKeyHistory(): Task<List<
TemporaryExposureKey>> =
551:        override fun getTemporaryExposureKeyHistory(): Task<List<
TemporaryExposureKey>> =
[...]
```

Another function `provideDiagnosisKeys` is used for storing contact code on the device. And according to the documentation only codes not older than 14 days will be stored. Override of this function was found in the test code:

```
nl-covid19-notification-app-android/app/src/test/java/nl/rijksoverheid/en/
ExposureNotificationsRepositoryTest.kt

[...]
175:            override suspend fun provideDiagnosisKeys(
232:            override suspend fun provideDiagnosisKeys(
286:            override suspend fun provideDiagnosisKeys(
332:                override suspend fun provideDiagnosisKeys(
381:                override suspend fun provideDiagnosisKeys(
434:                override suspend fun provideDiagnosisKeys(
490:                override suspend fun provideDiagnosisKeys(
555:                override suspend fun provideDiagnosisKeys(
[...]

nl-covid19-notification-app-android/en-api/src/test/java/nl/rijksoverheid/en/enapi/
nearby/NearbyExposureNotificationApiTest.kt

[...]
394:    fun `provideDiagnosisKeys without error removes files and returns Success
`() = runBlocking {
400:                override fun provideDiagnosisKeys(
```

[7] https://developers.google.com/android/exposure-notifications/exposure-notifications-api

```
413:        val status = api.provideDiagnosisKeys(
425:    fun `provideDiagnosisKeys with generic error removes files and returns
UnknownError`() =
433:                    override fun provideDiagnosisKeys(
442:                val status = api.provideDiagnosisKeys(
458:    fun `provideDiagnosisKeys with disk io removes files and returns
FailedDiskIo`() =
466:                    override fun provideDiagnosisKeys(
475:                val status = api.provideDiagnosisKeys(
533:        override fun provideDiagnosisKeys(
[...]
```

But also in the production code:

```
nl-covid19-notification-app-android/en-api/src/main/java/nl/rijksoverheid/en/enapi/
NearbyExposureNotificationApi.kt
[...]
    override suspend fun provideDiagnosisKeys(
        files: List<File>,
        configuration: ExposureConfiguration,
        token: String
    ) = suspendCoroutine<DiagnosisKeysResult> { c ->
        client.provideDiagnosisKeys(files, configuration, token).
addOnSuccessListener {
            c.resume(DiagnosisKeysResult.Success)
        }.addOnFailureListener {
            Timber.e(it, "Error while providing diagnosis keys")
            val apiException = it as? ApiException
            c.resume(
                when (apiException?.statusCode) {
                    ExposureNotificationStatusCodes.FAILED_DISK_IO ->
DiagnosisKeysResult.FailedDiskIo
                    else -> DiagnosisKeysResult.UnknownError(it)
                }
            )
        }.addOnCompleteListener {
            files.forEach { it.delete() }
        }
    }

[...]
```

No code related to the adjusting time windows was found. The `provideDiagnosisKeys` function is called by function `processExposureKeySets`:

```
nl-covid19-notification-app-android/app/src/main/java/nl/rijksoverheid/en/
ExposureNotificationsRepository.kt

[...]
/**
 * Downloads new exposure key sets from the server and processes them
 */
@VisibleForTesting
```

```
internal suspend fun processExposureKeySets(manifest: Manifest):
ProcessExposureKeysResult {
        [...]
        }

        val configuration = try {
            getConfigurationFromManifest(manifest)
        } catch (ex: IOException) {
            Timber.e(ex, "Error fetching configuration")
            return ProcessExposureKeysResult.Error(ex)
        }

        Timber.d("Processing ${validFiles.size} files")

        preferences.edit {
            putInt(KEY_MIN_RISK_SCORE, configuration.minimumRiskScore)
        }

        val result = exposureNotificationsApi.provideDiagnosisKeys(
            validFiles.map { it.file\!\! },
            configuration,
            createToken()
        )
[...]
```

The configuration for the GAEN API is retrieved from the manifest by calling `getConfigurationFromManifest` function:

```
 private suspend fun getConfigurationFromManifest(manifest: Manifest):
ExposureConfiguration {
        val riskCalculationParameters =
            api.getRiskCalculationParameters(manifest.riskCalculationParametersId)
        return ExposureConfiguration.ExposureConfigurationBuilder()
            .setDurationAtAttenuationThresholds(*riskCalculationParameters.
durationAtAttenuationThresholds.toIntArray())
            .setMinimumRiskScore(riskCalculationParameters.minimumRiskScore)
            .setTransmissionRiskScores(*riskCalculationParameters.
transmissionRiskScores.toIntArray())
            .setDurationScores(*riskCalculationParameters.durationScores.toIntArray
())
            .setAttenuationScores(*riskCalculationParameters.attenuationScores.
toIntArray())
            .setDaysSinceLastExposureScores(*riskCalculationParameters.
daysSinceLastExposureScores.toIntArray())
            .build()
    }
```

No settings regarding changing the time window is present in the configuration.

4.9.4. Conclusion

The applications do not offer a centralised functionality to configure a expiry duration for keys. However, implicitly the GAEN functionality of providing keys up to 14 days old is being used.

4.10. F9 - Collection of contact codes

Requirement	F9: The application collects periodically and incrementally contact codes from recent exposures from the server.
Conclusion	Secura finds no inconsistencies to this requirement.

4.10.1. iOS

The downloaded exposure TEKs are stored via `createKeySetHolders` in `RequestExposureKeySetsDataOperation`. The method `requestExposureKeySetsOperation` in the `ExposureDataOperationProvider` class invokes this, and is itself called from `ExposureDataController`'s method `fetchAndStoreExposureKeySets`. This in turn is called from `fetchAndProcessExposureKeySets`, which only the `ExposureController` calls, in a method with the same name. That again is called from `updateWhenRequired` in the same class. Section 4.6.0.6 on page 21 discusses that the app schedules a background task that invokes this, which shows that it does download the codes periodically.

`fetchAndProcessExposureKeySets` requests the application configuration first, feeds that to `fetchAndStoreExposureKeySets` which takes the `exposureKeySetsIdentifiers` from the manifest and calls `requestExposureKeySetsOperation` to it. That method applies the operation with the same name to it, and in its `execute` method it removes any keyset identifiers of keysets that are already downloaded or processed. This means that the download of keysets is also incremental.

4.10.2. Android

The Android functionality is similar. Secura sees no issues with this functionality.

4.11. F10 - Central configuration

Requirement	F10: The frequency of collecting data from the server can be centrally configured.
Conclusion	Secura finds no inconsistencies to this requirement.

Both the Android and iOS app contact the server to get the manifest, application configuration, and exposed keys. Because it is not clear what is meant exactly by “the app checking with the server”, we assume it to be all of these operations. Also the requirement does not state who should be able to configure and modify this, but we assume this must be the server.

4.11.1. iOS

All requests that the app makes over the network, are defined in the `NetworkManager` class. Related to this requirement, the class has four methods:

```
getManifest
getAppConfig
getRiskCalculationParameters
getExposureKeySet
```

The `NetworkController` calls these from:

```
var applicationManifest
func applicationConfiguration
func exposureRiskConfigurationParameters
func fetchExposureKeySet
```

For each of these, there is an operation that calls it:

```
RequestManifestExposureDataOperation
RequestExposureConfigurationDataOperation
RequestExposureKeySetsDataOperation
RequestAppConfigurationDataOperation
```

`RequestManifestExposureDataOperation`'s execute method calls `retrieveManifestUpdateFrequency` and uses this value to check if the stored manifest is not older than a certain time. If the manifest is still valid, it will not update it. `retrieveManifestUpdateFrequency` finds this value in the application configuration.

The `RequestExposureKeySetsDataOperation` is called from The ExposureController's method updateWhenRequired. This is run periodically, to a configurable update interval. The method `getAppRefreshInterval` of the `ExposureDataController` retrieves this value from the application configuration.

The `RequestAppConfigurationDataOperation` is called from `requestAppConfigurationOperation`, which in turn is called from `requestApplicationConfiguration`, which the `ExposureDataController` calls in many places.

The operation compares the stored application configuration identifier with the one it retrieved from the manifest, and gets the new application configuration from the server only if the identifier differs. The application configuration update interval is tied to the manifest update interval.

The same holds for the `RequestExposureConfigurationDataOperation` operation.

4.11.2. Android

Within the Android application, whenever a configuration value is requested, the application checks whether a current version is cached:

```
    suspend fun getCachedConfigOrDefault(): AppConfig = getConfigOrDefault {
        cdnService.getAppConfig(
            cdnService.getManifest("only-if-cached,max-stale=${Int.MAX_VALUE}").
appConfigId,
            "only-if-cached,max-stale=${Int.MAX_VALUE}"
        )
    }
```

When this is not the case, a new version will be requested.

4.12. F11 - Automatic exposure detection based on configuration

Requirement	F11: The application warns a user automatically if there was a possible exposure, based on relevant contact (ie 15 minutes within 1,5 meters) with a confirmed infected person.
Conclusion	Secura finds no inconsistencies to this requirement.

4.12.1. iOS

Detection of exposures is described in section 4.9 on page 24. Before persisting the report, the class `ProcessExposureKeySetsDataOperation` calls `createReportAndTriggerNotification`, which according to its name, creates the report and triggers a notification.

It calls `getExposureInformations`, which creates the exposure notifications and feeds them to the queue `DispatchQueue.main`.

4.12.2. Android

On Android the `ExposureNotificationReceiver.kt` registers the following Broadcast Receiver:

```
class ExposureNotificationReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        if (intent.action == ExposureNotificationClient.
ACTION_EXPOSURE_STATE_UPDATED) {
            val token = intent.getStringExtra(ExposureNotificationClient.
EXTRA_TOKEN)
            if (token != null) {
                ExposureNotificationJob.showNotification(context, token)
            } else {
                Timber.e("No token")
            }
        }
    }
}
```

This receiver catches the `ACTION_EXPOSURE_STATE_UPDATED` send by the GAEN when a new exposure is detected.

4.13. Q13 - Warnings on communication errors

Requirement	Q13: When the application is unable to communicate because of networking issues, the user will be notified.
Conclusion	Secura finds no inconsistencies to this requirement.

4.13.1. iOS

All network related methods in the `NetworkManager` expect calls to the network to fail, and return the failure on the stream that made the calls.

To specifically detect network availability, the `NetworkController` implements two methods: `startObservingNetworkReachability` and `stopObservingNetworkReachability`. It uses the Reachability library[8] for this. The app sets two handlers to update the network status stream:

```
reachability?.whenReachable = { [weak self] status in
    self?.mutableNetworkStatusStream.update(isReachable: status.connection != .
unavailable)
}
reachability?.whenUnreachable = { [weak self] status in
    self?.mutableNetworkStatusStream.update(isReachable: !(status.connection == .
unavailable))
}
```

[8]Source: `https://github.com/ashleymills/Reachability.swift`

The `RootRouter` calls these methods from the lifecycle hooks `didEnterForeground` and `didEnterBackground`.

The `ExposureController`'s method `postExposureManagerActivation` uses the `NetworkStatusStream` to check if it can do updates.

The app can give warnings and messages to the user from events via the `UserNotificationCenter`. The code contains a number of methods such as notifyUser that will invoke the UserNotificationCenter to send a message to the user.

The `BackgroundController` has the method `notifyUserENFrameworkDisabled`. It is called from the method `handleENStatusCheck` in the same class.

The method `handleUpdate` calls the `ExposureController`'s method `notifyUserIfRequired`. This is the only call to that method.

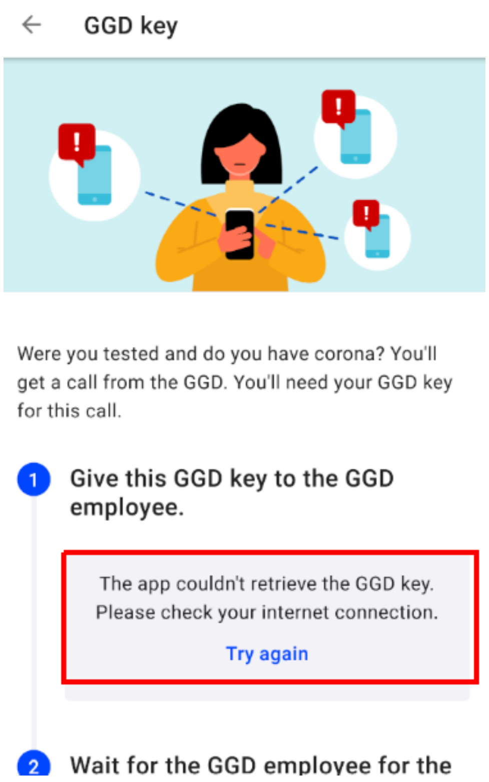
The `ExposureController` also defines the method `notifyUserAppNeedsUpdate`. It is called from `notifyUserIfRequired` in the same class.

The class `ProcessPendingLabConfirmationUploadRequestsDataOperation` defines `notifyUser`. It is called from its `execute` method.

If any of these methods encounters a network error, it will send a notification. All functionality that needs the network is covered by the above.

4.13.2. Android

The application shows an error message when the time-out reached during the network communication:



Were you tested and do you have corona? You'll get a call from the GGD. You'll need your GGD key for this call.

1 Give this GGD key to the GGD employee.

The app couldn't retrieve the GGD key. Please check your internet connection.

2 Wait for the GGD employee for the

Figure 4.2: Communication error message

Additionally, the application generates an error message when the registration was unsuccessful:

```
nl-covid19-notification-app-android/app/src/main/java/nl/rijksoverheid/en/labtest/
LabTestViewModel.kt
[...]

class LabTestViewModel(private val labTestRepository: LabTestRepository) :
ViewModel() {
    sealed class KeyState {
        object Loading : KeyState()
        data class Success(val key: String) : KeyState()
        object Error : KeyState()
    }

    val uploadResult: LiveData<Event<UploadResult>> = MutableLiveData()

    private var usedKey: String? = null

    private val refresh = MutableLiveData<Unit>()
    val keyState: LiveData<KeyState> = refresh.switchMap {
        liveData {
            emit(Loading)
            val result = labTestRepository.registerForUpload()
            if (result is RegistrationResult.Success) {
                usedKey = result.code
                emit(Success(result.code))
            } else {
                emit(Error)
            }
        }
    }
[...]
```

The function `registerForUpload` does not show any specific handling of the communication errors:

```
suspend fun registerForUpload(): RegistrationResult {
        return withContext(Dispatchers.IO) {
            val code = getCachedRegistrationCode()
            if (code != null) {
                return@withContext RegistrationResult.Success(code)
            }
            try {
                val config = appConfigManager.getCachedConfigOrDefault()
                val result = api.register(RegistrationRequest(), config.requestSize
)
                storeResult(result)
                RegistrationResult.Success(result.labConfirmationId)
            } catch (ex: HttpException) {
                Timber.e(ex, "Error registering")
                RegistrationResult.UnknownError
            } catch (ex: IOException) {
                Timber.e(ex, "Error registering")
                RegistrationResult.UnknownError
            }
        }
    }
```

Testing in practice shows that warnings are shown when no communication is possible.

4.14. *Q14 - Warning on loss of required technology*

Requirement	When the application is unable to communicate because of loss of required technology, the user will be notified.
Conclusion	Secura finds no inconsistencies to this requirement.

4.14.1. iOS

The collection of RPIs, done by the GAEN library however, requires Bluetooth. The app has a `BluetoothSettingsListener`, but only uses it during onboarding.

Because the GAEN library collects the RPIs, it would be responsible for detecting errors. Considering that the GAEN code is out of scope of this investigation, we cannot determine whether the library notifies the app of any errors. The app also actively checks the GAEN library status every 24 hours via the method `handleENStatusCheck`. If the library is not working, it will report this to the user.

4.14.2. Android

On Android, the Google Exposure Notification service will show a system notification if enabled and Bluetooth is disabled. Therefore this is outside the scope of the CoronaMelder application.

4.15. Q22 - De-anonymization

Requirement	Q22: Data processed by the application can not be traced to an individual person.
Conclusion	There are two ways how the backend can potentially identify users, both should be covered as discussed in the DPIA. However, since the backend was not in scope for this assessment, no assurances can be given. Additionally, time based observation might allow to distinguish real from decoy traffic.

The design takes care of this, under the assumption that the GAEN library ensures data cannot be tied to an individual. The only data that could be linked to an individual, is the lab confirmation code that a user supplies to the health provider. The health provider does not have access to the uploaded TEKs on the server and is bound by its professional oath and the law not to disclose personal information.

The requests that the app sends over the network are implemented in the `NetworkManager` class. The HTTP requests constructed there contain the minimum amount of information that should be sent. Even the HTTP headers are kept to a minimum.

4.15.1. Anonymity: unlinkability of TEKs to individuals

The daily RPIs are generated on the end-user device, using the TEKs as a seed. Both the RPIs and TEKs are associated with a single device/person and as such pseudonymous identifiers rather than anonymous values. The RPIs and TEKs are intended to not be linkable to individual persons.

Anonymity is by definition[9] bound by three elements:

- a subject (here: a person who uses the app)
- an Item of Interest / IOI (here: an RPI or TEK)
- an adversary (depends on the threat model — open to debate)

The subject can be said to be sufficiently anonymous from the adversary's perspective with regard to an IOI, if the adversary is unable to correctly identify a link between the IOI and the subject (unlinkability). Whether or not that is possible depends on the information position, access position, and capabilities attributed to the adversary.

For the notification app, which once deployed will be considered vital infrastructure, the scenario of an adversary who is able to observe the backend — such as a human operator or software used for performance and reliability monitoring on the backend — may need to be considered. In other words: what if, from a user's perspective, the backend itself is considered untrusted?

Communication between the app and the backend takes place via the public internet, without mediation. When TEKs are sent, the user's real IP address is included as part of the communication. This communication may take place multiple times per day.

An IP address is identifying information, as is also recognised in the DPIA[10] in the context of traffic analysis. If a device is connected via the user's private Wi-Fi connection at the time of this communication, the domestic IP address is disclosed to

[9] See: "A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management", Pfitzmann and Hansen, version 0.34, August 2010: `https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf`
[10] DPIA: `https://github.com/minvws/nl-covid19-notification-app-coordination/tree/master/privacy`

the backend. This is a strong identifier that, similar to a phone number, can be expected to be present in external databases[11] that also contain the user's real name and address.

The backend — which itself is out of scope of this assessment — can perform IP stripping, as is shown the 'Backend overview' architecture diagram[12] If the IP stripping takes place *after* TLS offloading, the combination of the user's IP address and TEKs is accessible, at least temporarily while a request is being processed.

One way to prevent this is to ensure the app does not communicate to the backend directly, but that the communication is brokered via an anonymising network, such as a custom/private onion routing network operated by entities who are independent from the government and the infrastructure hosting party. TLS encryption will still ensure confidentiality and integrity of the communication between the user and the backend.

This is beyond the scope of the assessment, hence no validation on the backend was performed.

REMARK 1	20060524-R1
Topic	The backend can potentially link TEKs an individual person
Applies to	CoronaMelder application
Description	The backend — which itself is out of scope of this assessment — can perform IP stripping, as is shown the 'Backend overview' architecture diagram[13] If the IP stripping takes place *after* TLS offloading, the combination of the user's IP address and TEKs is accessible, at least temporarily while a request is being processed. This finding has not been verified on the backend since that was out of scope for this assessment. For more information see section 4.15.1 on the preceding page
Recommendation	One way to prevent this is to ensure the app does not communicate to the backend directly, but that the communication is brokered via an anonymising network, see the DPIA for more information. TLS encryption will still ensure confidentiality and integrity of the communication between the user and the backend.

4.15.1.1. *Tracking of users*

Bucket id's and confirmation keys have an expiration time. In the code of the Android app this was observed in the `LabTestRepository` class defined in `./app/src/main/java/nl/rijksoverheid/en/labtest/LabTestRepository.kt`:

```
    private fun clearKeyDataIfExpired() {
        val expiration = preferences.getLong(KEY_REGISTRATION_EXPIRATION, 0)
        if (expiration == 0L || expiration < clock.millis()) {
            clearKeyData()
        }
    }

    private fun clearKeyData() {
        preferences.edit {
            remove(KEY_CONFIRMATION_KEY)
            remove(KEY_REGISTRATION_EXPIRATION)
            remove(KEY_LAB_CONFIRMATION_ID)
            remove(KEY_BUCKET_ID)
            remove(KEY_PENDING_KEYS)
            remove(KEY_UPLOAD_DIAGNOSTIC_KEYS)
```

[11] For instance a database of a large e-commerce website, government databases that also include person names and addresses.
[12] See `https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/`.

```
            remove(KEY_DID_UPLOAD)
        }
    }

    suspend fun uploadDiagnosticKeysIfPending(): UploadDiagnosticKeysResult {
        [...]
        clearKeyDataIfExpired()
    [...]
    }
```

Here, every time `uploadDiagnosticKeysIfPending()` is invoked, the `clearKeyDataIfExpired()` is invoked and removes the keys if the expiration time has been reached. The expiration time is obtained via a configuration setting referenced via `KEY_REGISTRATION_EXPIRATION` (which refers to the preference option `registration_expiration`). In the version of the source code that was inspected, no value was found for this parameter. It is not clear what value will be used in the production app.

In the same class file, the method `uploadKeys()` shows the use of the confirmation key to create an `HmacSecret` object (note: the latter class is defined in the `SignedBodyInterceptor` as a simple placeholder: `class HmacSecret(val secret: ByteArray)`):

```
    private suspend fun uploadKeys(
        requestedKeys: List<TemporaryExposureKey>,
        bucketId: String,
        confirmationKey: ByteArray
    ) {
        val config = appConfigManager.getCachedConfigOrDefault()
        val request = PostKeysRequest(requestedKeys.map {
            nl.rijksoverheid.en.api.model.TemporaryExposureKey(
                it.keyData,
                it.rollingStartIntervalNumber,
                it.rollingPeriod
            )
        }, bucketId)
        api.postKeys(request, HmacSecret(confirmationKey), config.requestSize)
    }
```

If the value of `confirmationKey` — or of a bucket id for that matter — remains constant over multiple uploads before the expiration triggers, an attacker who has access to the backend (prior to IP addresses being stripped) may be able to infer uniquely identifying patterns of times and IP addresses linked to a known bucket id or confirmation key. For instance, a user's device may be connected to a private home internet connection in the morning and evening, and be connected to a school or employer Wi-Fi network during the day. If this pattern is sufficiently unique, an attacker who has access to the backend may be able to infer links between a user's past and future confirmation keys by matching patterns of changes in IP addresses. This may enable linkability where unlinkability is desired from a privacy perspective.

As said, whether or not this threat is sufficiently mitigated by measures documented in the DPIA, is beyond the scope of this assessment.

REMARK 2	20060524-R2
Topic	The backend can potentially link confirmation keys or bucket id's to an individual
Applies to	CoronaMelder application
Description	An attacker who has access to the backend (prior to IP addresses being stripped) may be able to infer uniquely identifying patterns of times and IP addresses linked to a known bucket id or confirmation key. This finding has not been verified on the backend since that was out of scope for this assessment. For more information see section 4.15.1.1 on page 37.
Recommendation	One way to prevent this is to ensure the app does not communicate to the backend directly, but that the communication is brokered via an anonymising network, such as a custom/private onion routing network operated by entities who are independent from the government and the infrastructure hosting party. TLS encryption will still ensure confidentiality and integrity of the communication between the user and the backend.

4.15.2. Decoy uploads

In order to thwart traffic analysis, the application sends out decoy messages to `stopkeys` instead of `postkeys`. Secura found no issues in the creation of these decoy requests that could lead to a direct identification of decoy versus normal traffic. However, the documentation on `https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/Traffic Analysis Mitigation With Decoys.md` states the following:

> *"Note: the above algorithm will also generate decoy traffic outside Health Authority office hours. Any traffic outside office hours can be distinguished as decoy traffic. However, this is not a privacy issue. Any upload during office hours is not distinguishable as real or decoy traffic."*

However, both the iOS and Android application are limited to sending decoy traffic between 7:00 and 19:00, contradicting this statement. Essentially, allowing an observer to know that data send outside these hours is not decoy traffic.

REMARK 3	20060524-R3
Topic	Decoy traffic is only sent during office hours.
Applies to	CoronaMelder application
Description	Decoys are limited to be send during office hours (7:00 to 19:00). Any traffic observed outside this time window must be real traffic. This violates the description in the documentation which states that: "Note: the above algorithm will also generate decoy traffic outside Health Authority office hours."
Recommendation	Send decoy traffic at random moments during the entire day to thwart attackers who can observe network traffic.

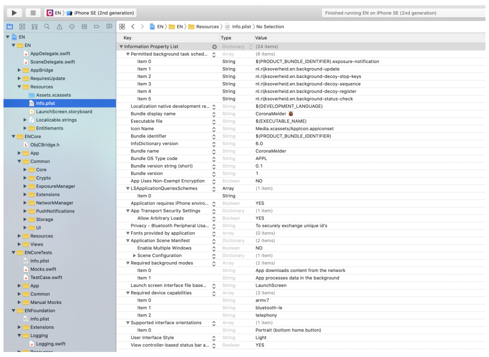
4.16. Q23 - No location Data

Requirement	Q23: The application does not use location data.
Conclusion	Secura finds no inconsistencies to this requirement.

The following research questions were applicable to this functionality:

4.16.1. iOS

The file `EN/Recources/Info.plist` contained the following information:



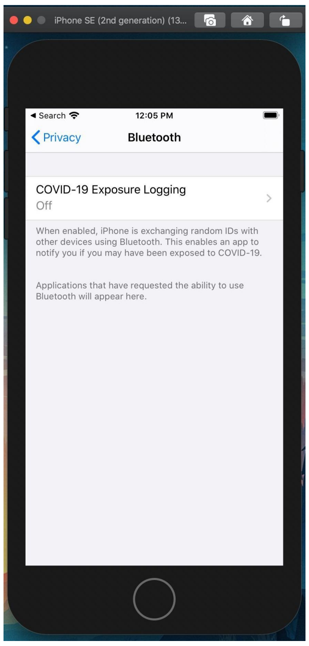
Key	Type	Value
▼ Information Property List	Dictionary	(24 items)
▼ Permitted background task sched...	Array	(6 items)
Item 0	String	$(PRODUCT_BUNDLE_IDENTIFIER).exposure-notification
Item 1	String	nl.rijksoverheid.en.background-update
Item 2	String	nl.rijksoverheid.en.background-decoy-stop-keys
Item 3	String	nl.rijksoverheid.en.background-decoy-sequence
Item 4	String	nl.rijksoverheid.en.background-decoy-register
Item 5	String	nl.rijksoverheid.en.background-status-check
Localization native development re...	String	$(DEVELOPMENT_LANGUAGE)
Bundle display name	String	CoronaMelder 🐞
Executable file	String	$(EXECUTABLE_NAME)
Icon Name	String	Media.xcassets/AppIcon.appiconset
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	CoronaMelder
Bundle OS Type code	String	APPL
Bundle version string (short)	String	0.1
Bundle version	String	1
App Uses Non-Exempt Encryption	Boolean	NO
▼ LSApplicationQueriesSchemes	Array	(1 item)
Item 0	String	
Application requires iPhone enviro...	Boolean	YES
▼ App Transport Security Settings	Dictionary	(1 item)
Allow Arbitrary Loads	Boolean	YES
Privacy - Bluetooth Peripheral Usa...	String	To securely exchange unique id's
▼ Fonts provided by application	Array	(0 items)
▼ Application Scene Manifest	Dictionary	(2 items)
Enable Multiple Windows	Boolean	NO
▶ Scene Configuration	Dictionary	(1 item)
▼ Required background modes	Array	(2 items)
Item 0	String	App downloads content from the network
Item 1	String	App processes data in the background
Launch screen interface file base...	String	LaunchScreen
▼ Required device capabilities	Array	(3 items)
Item 0	String	armv7
Item 1	String	bluetooth-le
Item 2	String	telephony
▼ Supported interface orientations	Array	(1 item)
Item 0	String	Portrait (bottom home button)
User Interface Style	String	Light
View controller-based status bar a...	Boolean	YES

Figure 4.3: Example of the properties file `EN/Recources/Info.plist`

The following permission was defined in the file:

- *Bluetooth*

A check was made in the iOS emulater, which confirmed that the *Bluetooth* and *Health* permissions were active.



iPhone SE (2nd generation) (13...

◂ Search 12:05 PM

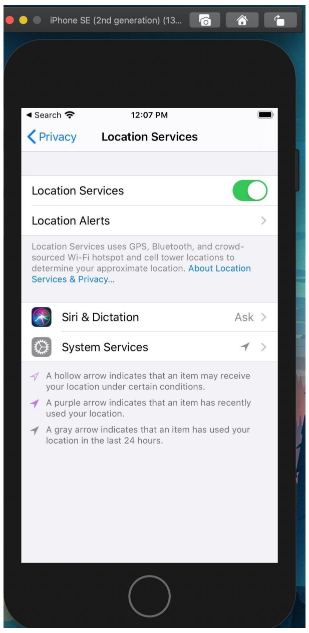
Privacy — Bluetooth

COVID-19 Exposure Logging
Off

When enabled, iPhone is exchanging random IDs with other devices using Bluetooth. This enables an app to notify you if you may have been exposed to COVID-19.

Applications that have requested the ability to use Bluetooth will appear here.

Figure 4.4: *Bluetooth* permission



iPhone SE (2nd generation) (13...

◂ Search 12:07 PM

Privacy — Location Services

Location Services: On

Location Alerts

Location Services uses GPS, Bluetooth, and crowd-sourced Wi-Fi hotspot and cell tower locations to determine your approximate location. About Location Services & Privacy...

Siri & Dictation — Ask

System Services

A hollow arrow indicates that an item may receive your location under certain conditions.

A purple arrow indicates that an item has recently used your location.

A gray arrow indicates that an item has used your location in the last 24 hours.

Figure 4.5: *Location Services* permission

4.16.2. Android

The application requires the Bluetooth permission, but no access to the precise GPS location is needed:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:
versionCode="1" android:versionName="0.4.1-acc" android:compileSdkVersion="29"
android:compileSdkVersionCodename="10" package="nl.rijksoverheid.en"
platformBuildVersionCode="29" platformBuildVersionName="10">
    <uses-sdk android:minSdkVersion="23" android:targetSdkVersion="29"/>
    <uses-feature android:name="android.hardware.bluetooth_le" android:required="
true"/>
    <uses-feature android:name="android.hardware.bluetooth"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.REORDER_TASKS"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
[...]
```

The application checks if Bluetooth ans Location services are enabled:

```
nl-covid19-notification-app-android/app/src/main/java/nl/rijksoverheid/en/
ExposureNotificationsRepository.kt
[...]

 suspend fun getCurrentStatus(): StatusResult {
        val result = exposureNotificationsApi.getStatus()
        return if (result == StatusResult.Enabled) {
            if (isBluetoothEnabled() && isLocationEnabled()) {
                statusCache.updateCachedStatus(StatusCache.CachedStatus.ENABLED)
                StatusResult.Enabled
            } else {
                statusCache.updateCachedStatus(StatusCache.CachedStatus.
INVALID_PRECONDITIONS)
                StatusResult.InvalidPreconditions
            }
        } else {
            if (result == StatusResult.Disabled) {
                statusCache.updateCachedStatus(StatusCache.CachedStatus.DISABLED)
            }
            result
        }
    }
[...]
```

However, no code was found that requests current location of the device. Enabling location is required for the correct functionality of the Google Exposure notification. Note that this requirement will be removed in future versions of the Google Exposure Notification on Android 11.[14]

[14] https://blog.google/inside-google/company-announcements/update-exposure-notifications/

4.17. Q26 - Anonymous contact codes

Requirement	Q26: Contact codes are not based on personal information.
Conclusion	Secura finds no inconsistencies to this requirement.

The GAEN ensures this functionality and is out of scope for this assessment.

4.18. Q28 - Use only after explicit consent

Requirement	Q28: Use of the application is only possible after explicit consent by the user.
Conclusion	Secura finds no inconsistencies to this requirement.

4.18.1. iOS

As mentioned in section 4.6 on page 18, if the user has not given consent, the state on the exposure stream is `notAuthorized`. None of the operations will work until the app is authorised.

The file `OnboardingViewController.swift`[15] contains the class responsible for this functionality:

```
final class OnboardingViewController: NavigationController,
OnboardingViewControllable, Logging {

[...]

    // MARK: - OnboardingConsentListener

    func consentClose() {
        listener?.didCompleteOnboarding()
    }

    func consentRequest(step: OnboardingConsentStepIndex) {
        router?.routeToConsent(withIndex: step.rawValue, animated: true)
    }

[...]

    private weak var listener: OnboardingListener?
    private let onboardingConsentManager: OnboardingConsentManaging
}
```

Giving consent to start the app looks as follows:

[15] `nl-covid19-notification-app-ios/Sources/ENCore/App/Features/Onboarding/OnboardingViewController.swift`

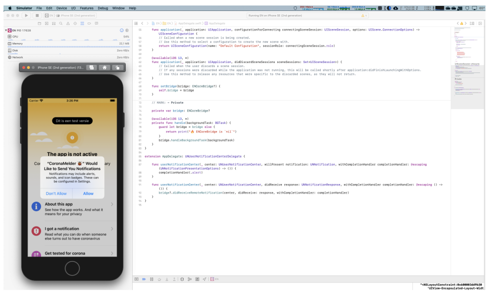

The app is not active
"CoronaMelder" Would Like to Send You Notifications
Notifications may include alerts, sounds, and icon badges. These can be configured in Settings.
Don't Allow
Allow
About this app
See how the app works. And what it means for your privacy
I got a notification
Read what you can do when someone else turns out to have coronavirus
Get tested for corona
Dit is een test versie

Figure 4.6: Example of the prompt that is shown after the 'start app' button is pressed

The conclusion is made that the CoronaMelder application asks for explicit consent before the app becomes active.

4.18.2. Android

Similar to iOS, the application can not be used without explicit consent. The following code in `In MainActivity.kt` checks whether consent was given, or requests it from the user:

```
viewModel.notificationsResult.observe(this, EventObserver {
when (it) {
    is ExposureNotificationsViewModel.NotificationsStatusResult.ConsentRequired ->
{
        startIntentSenderForResult(
            it.intent.intentSender,
            RC_REQUEST_CONSENT,
            null,
            0,
            0,
            0
        )
    }
[...]
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == RC_REQUEST_CONSENT && resultCode == Activity.RESULT_OK) {
        viewModel.requestEnableNotifications()
    }
    // If user canceled the forced update, do not allow them to use the app
    if (requestCode == RC_UPDATE_APP && resultCode != Activity.RESULT_OK) {
        finish()
    }
}
```

4.19. Generation, storage and transport of logging data.

Requirement	Secure generation, storage and transmission of local log files
Conclusion	Secura finds no inconsistencies to this requirement.

Both the logs of the iOS and Android application were inspected during use. Additionally, no sensitive logged information was discovered outside of appropriate places. Additionally, no online upload of log information was present. For more information on logging see section 4.3.1 on page 12.

4.20. *Protection of data in local storage*

Requirement	Secure local storage of data and secure deletion of contact codes
Conclusion	Secura finds no inconsistencies to this requirement.

The local storage for both iOS and Android was inspected. No data was places here that was out of context or more sensitive than necessary.

4.21. *External libraries*

Requirement	Correct use of software libraries and library versions
Conclusion	At the time of writing, a number of libraries were out of date. Specifically one OpenSSL library used by the iOS application, which was released on 10 September 2019. Since its release, two security bugs were made public that affect this version: CVE-2019-1551 and CVE-2020-1967. These bugs do not affect the security of the CoronaMelder app, because the library is only used to validate certificates and the bugs do not affect that process. It is nonetheless recommended to upgrade to the most current version, as a general best practice.

Both iOS and Android applications make use of a number of external libraries. While these are out of scope for the assessment, Secura has assessed the versions that are in use for publicly known security vulnerabilities.

4.21.1. iOS

The iOS version on the app CoronaMelder application makes use of the following third party software libraries (Cocoapods):

- CocoaLumberjack version 3.6.1 (*CocoaLumberjack is a fast and simple, yet powerful and flexible logging framework for Mac and iOS.*)- BSD-3-Clause license[16]
- Lottie version 3.18 (*Lottie is a mobile library for Android and iOS that natively renders vector based animations and art in realtime with minimal code.*)-Apache-2.0 license[17] -
- Reachability version 5.0.0. - BSD license(*This is a drop-in replacement for Apple's Reachability class. It is ARC-compatible, and it uses the new GCD methods to notify of network interface changes.*)[18] Recent version 5.0.0
- SnapKit version 5.0.0 (*SnapKit is a DSL to make Auto Layout easy on both iOS and OS X*) - MIT license[19]
- SnapshotTesting version 1.7.2 (*Delightful Swift snapshot testing*) - MIT license[20]
- ZipFoundation version 0.9.11 (*ZIP Foundation is a library to create, read and modify ZIP archive files*)- MIT license[21]

A check of various internet sources showed that recent versions were in use by the app. No public vulnerabilities were discovered in these libraries. The following framework was also included in the CoronaMelder application

[16]`https://github.com/CocoaLumberjack/CocoaLumberjack`
[17]`https://cocoapods.org/pods/lottie-ios`
[18]`https://github.com/tonymillion/Reachability`
[19]`https://github.com/SnapKit/SnapKit`
[20]`https://cocoapods.org/pods/SnapshotTesting`
[21]`https://cocoapods.org/pods/ZIPFoundationview`

- OpenSSL version 1.1.1D - OpenSSL & SSLeay license[22]

OpenSSL version 1.1.1D is used by the iOS application, which was released on 10 September 2019. Since its release, two security bugs were made public that affect this version: CVE-2019-1551 and CVE-2020-1967. These bugs do not affect the security of the CoronaMelder app, because the library is only used to validate certificates and the bugs do not affect that process. It is nonetheless recommended to upgrade to the most current version, as a general best practice.

The conclusion is that the app uses recent versions of third party libraries. However, the version off the OpenSSL framework in use of the CoronaMelder application was outdated and contained public vulnerabilities.

RISK NOTE 2	20060524-N2
Topic	Outdated library used in iOS application: OpenSSL
Applies to	CoronaMelder application (iOS)
Description	OpenSSL version 1.1.1D is used by the iOS application, which was released on 10 September 2019. Since its release, two security bugs were made public that affect this version: CVE-2019-1551 and CVE-2020-1967. These bugs do not affect the security of the CoronaMelder app, because the library is only used to validate certificates and the bugs do not affect that process. It is nonetheless recommended to upgrade to the most current version, as a general best practice. For more information see the description in section 4.21 on the preceding page.
Risk	When this library would be used for TLS connections, it would be possible to perform a denial of service of a client under specific circumstances via the use of a publicly known vulnerability. In order to prevent issues in future iterations when this library might be used in another way, it is nonetheless recommended to upgrade to the most current version.
Recommendation	Upgrade the software to a version without known vulnerabilities.
Reproduction	See the description in 4.21.1 on the previous page Cehck the version of the OpenSSL framework in use by the CoronaMelder application.
Metadata	Likelihood: Low, Impact: Low, CVSS-Score:0.0 CVSS:3.0/AV:A/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:N

4.21.2. Android

The Android application makes use of a large number of dependencies. At the time of writing, the following libraries were not up to date:

- `"com.diffplug.spotless:spotless-plugin-gradle:3.30.0"`
- `"com.osacky.flank.gradle:fladle:0.10.2"`
- `'com.squareup.moshi:moshi-kotlin:1.9.2'`
- `'com.squareup.moshi:moshi-kotlin-codegen:1.9.2'`
- `'com.squareup.okhttp3:logging-interceptor:4.8.0'`
- `"com.squareup.okhttp3:mockwebserver:4.8.0"`
- `'com.squareup.okhttp3:okhttp:4.8.0'`
- `'com.squareup.retrofit2:converter-moshi:2.8.1'`
- `'org.bouncycastle:bcpkix-jdk15to18:1.65'`
- `'org.bouncycastle:bcprov-jdk15to18:1.65'`
- `"org.mockito:mockito-android:3.4.4"`

While not up-to-date, no public security vulnerabilities were discovered in these versions.

[22] https://www.openssl.org/news/openssl-1.1.1-notes.html

4.22. Protection of data in transit

Requirement	Protected communication with the backend servers and protection against Man-in-the-Middle attacks
Conclusion	Secura finds no inconsistencies to this requirement.

The following research questions are applicable to this section:

- Protection of communication with the backend servers.
- Protection against Man-in-the-Middle (MitM) attacks.

The logic for TLS pinning is implemented in `ENCore/Common/NetworkManager/NetworkManagerURLSessionDelegate.swift`. It appears that the root certificate within the chain is the single pinned certificate:

```
final class NetworkManagerURLSessionDelegate: NSObject, URLSessionDelegate {
    [...]

    func urlSession(_ session: URLSession, didReceive challenge:
URLAuthenticationChallenge, completionHandler: @escaping (URLSession.
AuthChallengeDisposition, URLCredential?) -> ()) {

        guard let localSignature = configurationProvider.configuration.sslSignature
(forHost: challenge.protectionSpace.host),
            challenge.protectionSpace.authenticationMethod ==
NSURLAuthenticationMethodServerTrust,
            let serverTrust = challenge.protectionSpace.serverTrust else {
            // no pinning
            completionHandler(.performDefaultHandling, nil)
            return
        }

        let policies = [SecPolicyCreateSSL(true, challenge.protectionSpace.host as
CFString)]
        SecTrustSetPolicies(serverTrust, policies as CFTypeRef)

        let certificateCount = SecTrustGetCertificateCount(serverTrust)

        guard
            SecTrustEvaluateWithError(serverTrust, nil),
            certificateCount > 0,
            let serverCertificate = SecTrustGetCertificateAtIndex(serverTrust,
certificateCount - 1), // get topmost certificate in chain
            let signature = Certificate(certificate: serverCertificate).signature
else {
            // invalid server trust
            completionHandler(.cancelAuthenticationChallenge, nil)
            return
        }

        guard localSignature == signature else {
            // signatures don't match
            completionHandler(.cancelAuthenticationChallenge, nil)
            return
        }

        // all good
        completionHandler(.useCredential, URLCredential(trust: serverTrust))
    }

    [...]
}
```

The term ‘SSL signatures’ used in the code refers to SHA-256 hashes (‘fingerprints’) of certificates. Aside from this pinning hash, standard certificate validation of the `URLSession` class is performed. This contains a check to match the certificate’s CN or ANs with the hostname of the connection target.

Domain names and pinning hashes for the production environment are defined in `ENCore/Common/NetworkManager/NetworkManagerURLSessionDelegate.swift`:

```
    static let production = NetworkConfiguration(
        name: "Production",
        api: .init(
            scheme: "https",
            host: "coronamelder-api.nl",
            port: nil,
            path: ["v1"],
            sslSignature: Certificate.SSL.apiSignature,
            tokenParams: [:]
        ),
        cdn: .init(
            scheme: "https",
            host: "productie.coronamelder-dist.nl",
            port: nil,
            path: ["v1"],
            sslSignature: Certificate.SSL.cdnSignature,
            tokenParams: [:]
        )
    )
```

Pinning constants are defined in `ENCore/Common/Crypto/Certificates.swift`:

```
extension Certificate {
    struct SSL {
        static let apiSignature: Certificate.Signature = "PE+
wuVq4swAy9DK4b1Nf4XLBhdD9OYZYN882GH+m9Cg="
        static let cdnSignature: Certificate.Signature = "PE+
wuVq4swAy9DK4b1Nf4XLBhdD9OYZYN882GH+m9Cg="
    }
}
```

After connecting to `coronamelder-api.nl`, it is noted that these fingerprints match the root certificate `Staat der Nederlanden Root CA - G3`.

The root certificate will expire on 14 November 2028, after which the current version of the app would stop working. This leaves sufficient time to update this key pin using an app update.

4.22.1. Android

For the Android app, the build file `api/build.gradle` sets the `FEATURE_SSL_PINNING` to `true`:

```
    defaultConfig {
        minSdkVersion 23
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles "consumer-rules.pro"

        // encoded query string for public read access on the cdn
        buildConfigField "String", "CDN_BASE_URL", "\"https://test.coronamelder-
dist.nl/\""
        buildConfigField "String", "API_BASE_URL", "\"https://test.coronamelder-api
.nl/\""
        buildConfigField "boolean", "FEATURE_RESPONSE_SIGNATURES", "true"
        buildConfigField "boolean", "FEATURE_SSL_PINNING", "true"
    }
```

This value is used in `api/src/main/java/nl/rijksoverheid/en/api/services.kt`:

```
private const val CDN_PIN = "sha256/Y9mvm0exBk1JoQ57f9Vm28jKo5lFm/woKcVxrYxu80o="
private const val API_PIN = "sha256/QiOJQAOogcXfa6sWPbI1wiGhjVS/dZlFgg5nDaguPzk="
[...]

internal fun createOkHttpClient(context: Context): OkHttpClient {
    return okHttpClient ?: OkHttpClient.Builder()
        // enable cache for config and resource bundles
        .cache(Cache(File(context.cacheDir, "http"), 32 * 1024 * 1024))
        .apply {
            addNetworkInterceptor(CacheOverrideInterceptor())
            addNetworkInterceptor(SignedResponseInterceptor())
            addInterceptor(PaddedRequestInterceptor())
            addInterceptor(SignedBodyInterceptor())
            if (Timber.forest().isNotEmpty()) {
                addInterceptor(HttpLoggingInterceptor(object :
HttpLoggingInterceptor.Logger {
                    override fun log(message: String) {
                        Timber.tag("OkHttpClient").d(message)
                    }
                }).setLevel(HttpLoggingInterceptor.Level.BODY))
            }
            if (BuildConfig.FEATURE_SSL_PINNING) {
                connectionSpecs(
                    listOf(
                        ConnectionSpec.MODERN_TLS
                    )
                )
                certificatePinner(
                    CertificatePinner.Builder()
                        .add(Uri.parse(BuildConfig.CDN_BASE_URL).host , CDN_PIN)
                        .add(Uri.parse(BuildConfig.API_BASE_URL).host , API_PIN)
                        .build()
                )
            }
        }.build().also { okHttpClient = it }
}
```

The result is behaviour that is identical to that of the iOS app.

4.23. TEK signing

Requirement	TEK signatures should not be forgeable by an attacker
Conclusion	Any leaf certificate recently issued by KPN as part of the PKIOverheid service would be accepted by the application. If an attacker would manage to obtain a key corresponding to such a certificate, they could forge TEK signatures. However, no exploitable scenario could be devised for this weakness.

While not part of the original research questions, Secura discovered a potential vulnerability in the TEK signing functionality. Section 2.8.2 of the architecture document `Crypto Raamwerk.md`[23] details the two signatures set on sets of TEKs used for diagnosis. The first signature follows the GAEN specification and is intended to be verified by the on-device Apple/Google API. The second signature will is validated by the CoronaMelder app itself.

Section 2.8.6 of said document states that the second signature, which uses a certificate chain and a CMS/PKCS#7 format, should be validated based on its issuing CA and subject name.

The signature validation code is located in `ENCore/Common/Crypto/SignatureValidator.swift`:

```
final class SignatureValidator: SignatureValidating {
private let openssl = OpenSSL()

func validate(signature: Data, content: Data) -> Bool {
    guard let rootCertificateData = validatedRootCertificateData() else {
        return false
    }

    return openssl.validatePKCS7Signature(signature,
                                          contentData: content,
                                          certificateData: rootCertificateData,
                                          authorityKeyIdentifier:
SignatureConfiguration.authorityKeyIdentifier)
}

private func validatedRootCertificateData() -> Data? {
    guard let certificateData = SignatureConfiguration.rootCertificateData else {
        return nil
    }

    guard openssl.validateSerialNumber(SignatureConfiguration.rootSerial,
                                       forCertificateData: certificateData) else {
```

[23]https://github.com/minvws/nl-covid19-notification-app-coordination/blob/0b9d8330c35b2aee98b1203c3e224eb8863d37a8/architecture/

```
        return nil
    }

    guard openssl.validateSubjectKeyIdentifier(SignatureConfiguration.
rootSubjectKeyIdentifier,
                                               forCertificateData: certificateData)
 else {
        return nil
    }

    return certificateData
}
}
```

The signature validator checks the chain based on a particular root certificate (which receives two additional checks) and a specific `authorityKeyIdentifier` in the leaf certificate (this normally matches the `subjectKeyIdentifier` of the issuing CA certificate). No subject name validation is observed, neither in this class nor in `ENCore/Common/Crypto/OpenSSL.m`. This is inconsistent with the documentation.

The root certificate and `authorityKeyIdentifier` are defined in `ENCore/Common/Crypto/SignatureConfiguration.swift`:

```
final class SignatureConfiguration {
    static var rootCertificateData: Data? {
        guard let localUrl = Bundle(for: SignatureConfiguration.self).url(
forResource: "nl-root", withExtension: "pem") else {
            return nil
        }

        return try? Data(contentsOf: localUrl)
    }

    static var rootSubjectKeyIdentifier: Data {
        // 04:14:54:AD:FA:C7:92:57:AE:CA:35:9C:2E:12:FB:E4:BA:5D:20:DC:94:57
        return Data([0x04, 0x14, 0x54, 0xad, 0xfa, 0xc7, 0x92, 0x57, 0xae, 0xca, 0
x35, 0x9c, 0x2e, 0x12, 0xfb, 0xe4, 0xba, 0x5d, 0x20, 0xdc, 0x94, 0x57])
    }

    static var authorityKeyIdentifier: Data {
        // 04:14:c3:9a:a6:7b:5e:74:2b:82:b6:c6:72:fd:74:4e:85:d2:97:cd:fd:18
        return Data([0x04, 0x14, 0xc3, 0x9a, 0xa6, 0x7b, 0x5e, 0x74, 0x2b, 0x82, 0
xb6, 0xc6, 0x72, 0xfd, 0x74, 0x4e, 0x85, 0xd2, 0x97, 0xcd, 0xfd, 0x18])
    }

    static var rootSerial: UInt64 {
        return 10003001
    }
}
```

The certificate located in `ENCore/Resources/nl-root.pem` is `Staat der Nederlanden Root CA - G3`, the same national root certificate pinned during TLS connections. The `rootSubjectKeyIdentifier` and serial number match it.

The `authorityKeyIdentifier` corresponds to that of the following KPN intermediate certificate[24]:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7b:74:85:b9:f0:51:4a:55:21:e5:2d:39:1a:1a:ae:db:93:6d:8f:8c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = NL, O = Staat der Nederlanden, CN = Staat der Nederlanden
Organisatie Services CA - G3
        Validity
            Not Before: Apr 16 08:40:16 2019 GMT
            Not After : Nov 12 00:00:00 2028 GMT
        Subject: C = NL, O = KPN B.V., organizationIdentifier = NTRNL-27124701, CN
= KPN BV PKIoverheid Organisatie Server CA - G3
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:cd:[...]
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            Authority Information Access:
                CA Issuers - URI:http://cert.pkioverheid.nl/
DomOrganisatieServicesCA-G3.cer
                OCSP - URI:http://domorganisatieservicesocsp-g3.pkioverheid.nl

            X509v3 Subject Key Identifier:
                C3:9A:A6:7B:5E:74:2B:82:B6:C6:72:FD:74:4E:85:D2:97:CD:FD:18
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Authority Key Identifier:
                keyid:43:EB:4D:00:D3:95:93:CE:A6:7C:40:0D:6D:11:BE:39:D1:32:AE:E2

            qcStatements:
                0.0...+.......0.......I..
            X509v3 Certificate Policies:
                Policy: 2.16.528.1.1003.1.2.5.6
                  CPS: https://cps.pkioverheid.nl

            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl.pkioverheid.nl/DomOrganisatieServicesLatestCRL-G3.
crl

            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Extended Key Usage:
                TLS Web Client Authentication, OCSP Signing, TLS Web Server
Authentication
    Signature Algorithm: sha256WithRSAEncryption
```

[24]See `https://certificaat.kpn.com/installatie-en-gebruik/installatie/ca-certificaten/kpn-g3-ca/`

```
        bb:b0:[...]
```

This CA certificate is widely used for government organisations using the PKIOverheid service through KPN. Any (web) application that uses this will have a `authorityKeyIdentifier` value matching this certificate. An example is the server certificate in use on `https://duo.nl/`.

If an attacker manages to obtain a private key belonging to any recent PKIOverheid certificate issued by KPN, they will be able to set signatures that pass this local validation.

RISK NOTE 3	20060524-N3
Topic	The subject name in the certificate used for TEK signing is not checked.
Applies to	CoronaMelder application
Description	When validating the second signature (using the certificate based CMS/PKCS#7 format) on a TEK list received by the server, the root and issuing CA certificates within the chain are checked. The subject name in the leaf certificate is not validated. See section 4.23 on page 49 for more information.
Risk	Any leaf certificate recently issued by KPN as part of the PKIOverheid service would be accepted by the `SignatureValidator` class. If an attacker would manage to obtain a key corresponding to such a certificate, they could forge TEK signatures. This violates the integrity requirements defined in the architecture documentation. Because TEKs are also protected in transit by TLS connection, and have additional protection due to the GAEN signature, it has not become clear during this assessment how this flaw could be practically exploited by an attacker.
Recommendation	Validate that the Subject of the signature certificate contains the correct Common Name (CN).
Reproduction	See the description in section 4.23 on page 49
Metadata	Likelihood: Low, Impact: Low, CVSS-Score:1.9 CVSS:3.0/AV:L/AC:H/PR:H/UI:N/S:U/C:L/I:N/A:N

4.24. *Correct use of the Google and Apple API's (GAEN)*

Requirement	Correct use of the Google and Apple API's (GAEN)
Conclusion	Secura finds no inconsistencies to this requirement.

The communication with the GAEN is central to the correct functioning of both applications. No vulnerabilities were discovered in how the CoronaMelder apps (Android and iOS) make use of the GAEN APIs.

4.25. *Limited permissions*

Requirement	Correct use of application permissions
Conclusion	Secura finds no inconsistencies to this requirement.

As detailed in section 4.16 on page 39, both the Android and iOS application only request permissions that are essential for the correct execution of the application. No superfluous permissions were discovered.

4.26. Root / Jailbreak detection

Requirement	Protection of the integrity of the application: No installation on rooted/jailbroken devices
Conclusion	The application can be installed on rooted/jailbroken devices. This is in line with a concious choice made by the developers and discussed in the documentation. However, end users should be informed via a message of the risks of using the application on such a device.

Both the iOS and Android application do not include functionality to detect an installation on a rooted or jailbroken device. This was a concious design decision as detailed in the documentation[25]:

> *"The Google Reference Implementation of a backend for exposure notification suggests the use of DeviceCheck (iOS) and Safetynet Attestation (Android) to validate if a request comes from a genuine android device and/or from the official app:* `https://github.com/google/exposure-notifications-server/blob/master/docs/server_functional_requirements.md`
>
> *The documentation for DeviceCheck and Safetynet Attestation can be found here:* `https://developer.apple.com/documentation/devicecheck`, `https://developer.android.com/training/safetynet/attestation`.
>
> *We have decided not to apply these platform specific checks. First, it relies on a server API at Apple and Google, which can be down and could be a privacy risk.*
>
> *Second, the Android Developer blog states:*
>
> *"In other words, not all users who fail attestation are necessarily abusers, and not all abusers will necessarily fail attestation. By blocking users solely on their attestation results, you might be missing abusive users that don't fail attestations. Furthermore, you might also be blocking legitimate, loyal customers who fail attestations for reasons other than abuse" (NOTE: https://android-developers.googleblog.com/2017/11/10-things-you-might-be-doing-wrong-when.html)*
>
> *The safetynet attestation documentation further states about attestation failure: "Most likely, the device launched with an Android version less than 7.0 and it does not support hardware attestation. In this case, Android has a software implementation of attestation which produces the same sort of attestation certificate, but signed with a key hardcoded in Android source code. Because this signing key is not a secret, the attestation could have been created by an attacker pretending to provide secure hardware" (NOTE: https://developer.android.com/training/articles/security-key-attestation)*
>
> *This leads us to believe that when applying these checks, we introduce risks and dependencies while not gaining a substantial amount of security."*

While the reasoning for this use case is sound, it still leaves users of rooted/jailbroken devices at a higher risk level. The application should notify users of this fact.

[25] `https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/`

REMARK 4	20060524-R4
Topic	No notification for installation on jailbroken / rooted devices
Applies to	CoronaMelder application
Description	The application does not detect if it is installed on a jailbroken/rooted device. All of the application's functionality can be used after installation on a jailbroken/rooted device. This is a concious decision as detailed in the architecture documentation. For more information see section 4.25 on page 52.
Recommendation	Detect if the application is installed on a jailbroken device and notify the user about the possible implications.

A. TESTING APPROACH AND BACKGROUND

A.1. Technical Security Assessment Types

To achieve the goal stated above, Secura uses a methodology that is derived from the Open Web Application Security Project (OWASP), the Information Systems Security Assessment Framework (ISSAF) and the Open Source Security Testing Methodology Manual (OSSTMM). It provides the correct execution of, amongst others, the following types of projects:

- Technical security assessment
  (this can relate to an infrastructure, networks and network components, systems and combinations hereof);
  - Black box *(with minimal information in advance and no credentials);*
  - Grey box *(using credentials supplied in advance);*
  - Crystal box *(having full access to all information pertaining the system to be assessed);*
- Code inspection;
- Vulnerability scan.

A.2. System Changes

We recommend that changes to facilitate the investigation are reverted. For instance changes to the firewall or IDS/IPS infrastructure, or newly created users and data in applications.

A.3. Assessment Criteria

Secura assessed the configuration and management of the systems to be examined against best practices as applicable for the type of systems at the time of the investigation.

A.4. Classification of Findings

A risk is defined by the impact multiplied by the likelihood of the risk occurring.

This section describes how we determine the likelihood and impact of risks.

A.4.1. Likelihood

The likelihood of exploitation is influenced by a number of variables, such as the (technical) knowledge that is needed to abuse a vulnerability and the availability (public or not) of programs *(exploits)* to take advantage of a vulnerability. Historical data can also be used to estimate the likelihood. Our classifications take into account the access level to the system that is needed, the difficulty of the techniques used, and the knowledge that is available or needed for the attacker. The classification of the likelihood can be found in table A.1 on the following page.

Classification	Description
Low	Requires a skilled attacker dedicated to exploiting this weakness, significant effort or resources, inside knowledge and/or circumstances outside of the control of the attacker (i.e. the use of a specific browser version by the victim).
Medium	Requires considerable skill, fair effort or resources, customised attacks and/or circumstances which an attacker may skillfully arrange (i.e. by means of social engineering).
High	Requires low or medium skill, minimal effort or resources and the use of publicly available knowledge or tools.

Table A.1: Likelihood of abuse

A.4.2. Impact

The impact concerns the consequences of an event. The impact includes financial damages and loss of reputation. Due to a lack of detailed knowledge of the customers business we are unable to determine the business impact. Instead, we determine the impact from a technical perspective and leave the estimation of the business impact to the customer. The classification we use is included in table A.2.

Classification	Description
Low	Very limited negative consequences for confidentiality, integrity and/or availability of the system under investigation.
Medium	Limited negative consequences for confidentiality, integrity and/or availability of the system under investigation.
High	Great negative consequences for confidentiality, integrity and/or availability of the system under investigation.

Table A.2: (Technical) impact of consequences

A.4.3. Risk

Based on the estimates of both likelihood and impact, the risk can be determined. We include a matrix to show how the risk is determined from these factors.

The risks are categorised in the following risk levels: critical, high, medium, low, and note.

We recommend to resolve all risks, but especially those with a classification of critical, high or medium. In the case of critical risks, we advise to take the application or infrastructure off-line and resolve the risk immediately. Furthermore, it would be prudent to investigate whether these risks have been exploited. In the case of high risks, we advise to resolve the risk as quickly as possible.

A.5. Remarks

Apart from ‘risk’, Secura also uses the term ‘remark’.

A remark is something that does not lead to a risk, often a functional problem. Whether or not this is resolved, is up to the customer.

A.6. Dossier

During the investigation extensive logs were kept. This information was used to create this report and is stored in a dossier belonging to this investigation. This information will be kept securely on the systems of Secura for a limited period of time.

After the final version of this report has been delivered to the customer, the digitally stored data will be overwritten multiple times before it is removed from the system (the end result can be compared to a secure erase as specified in “DoD 5220”). Information on paper, CD-ROM and/or DVD-ROM will be destroyed using a shredder complying to DIN standard 66399/2012 level 3[1].”

[1]This standard applies to the destruction of confidential documents.

B. USED ACRONYMS

In our profession acronyms are often used. Because the meaning may not always be directly clear, this appendix tries to provide an overview of all acronyms used in this document.

RPI	Rolling Proximity Identifiers
GAEN	Google Apple Exposure Notification
TEK	Temporary Exposure Key
AN	Alternative Name
API	Application Programming Interface
BSD	Berkeley Software Distribution
CA	Certificate Authority
CMS	Content Management System
CN	Common Name
DIN	Deutsches Institut für Normung e.V.
DoD	Department of Defense
GPS	Global Positioning System
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IP	Internet Protocol
ISSAF	Information Systems Security Assessment Framework
MitM	Man-in-the-Middle
OSSTMM	Open Source Security Testing Methodology Manual
OWASP	Open Web Application Security Project
PDF	Portable Document Format
PKCS	Public Key Cryptography Standards
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security
USB	Universal Serial Bus