Security Analysis of Min Læge

Network Security

Friday 16th December, 2022

Kevork Christian Dounato
Aarhus University
201907831@uni.au.dk

Kristoffer Plagborg Bak Sørensen
Aarhus University
201908140@uni.au.dk

Contents

1. Introduction

The digitization of the Danish public healthcare system has made it easier for citizens to access medical information and services through their mobile devices. However, this has also raised important security concerns. The personal and sensitive nature of this information means that it is subject to strict laws in Denmark. As a result, apps and services that provide access to this data must prioritize security and take steps to protect it from unauthorized access. This includes using cryptographic protection and ensuring that the data is not shared with any malicious actors. Ensuring the security of this information is essential in order to protect the privacy of Danish citizens and prevent the potential misuse of their personal data.

The purpose of this report is to examine the security and privacy of Min Læge. The report is organized as follows. First, the features and data handling of Min Læge will be introduced. Next, a threat model will be outlined based on this information. The software security of Min Læge will then be discussed, followed by a network security analysis of the app. The findings of these analyses will be considered in relation to the proposed threat model. Finally, an overall assessment of Min Læge's security and privacy will be provided.

A. Description of Min Læge

Min Læge is developed by the company Trifork A/S on behalf of the Danish Healthcare Ministry. A Danish citizen that is logged into app with their nationally provided MitID, can:

1) View their full detailed medical record as a timeline.
2) Block any doctor from viewing their medical information.
3) View and book doctors appointments.
4) View opening hours of their doctor.
5) Call their doctor or the duty doctor.
6) Have a video consultation with a doctor.

7) Message their doctor or nurse.

8) Get a list of the closest clinics to their location.

9) View completed and recommended vaccinations.

10) View lab results, referrals and diagnoses.

11) Read important information, e.g. about COVID-19.

Min Læge is a feature-rich app, which means it has a higher potential for security flaws. However, we are not able to in-depth test the video consultancy feature or text messaging functionality present in the app, as that would require approval from both a doctor and app owner.

2. Threat Model

As an app that handles a wide range of personal health information, Min Læge has a wide list of adversaries and a large attack surface.

A. Adversaries

The main adversaries for Min Læge are

- Individual people and cyber criminals: As the app deals with sensitive information, there is a risk that individuals or hackers may try to use this information to harm or extort others. There is also a risk that some people could be discriminated against if their medical information becomes public.
- Government entities: Another government may try to leak sensitive data about high-profile political figures, in order to blackmail them into changing their stance on certain issues, or to interfere in domestic affairs and cause disarray.
- Insider threat: An employee who works on the development of the app, or one of its services may pose a risk, as they could act maliciously and have the ability to introduce an exploit or backdoor.

B. Attack Surface

This report will not cover the attack surface of second, or third party entities such as the data centers' cloud infrastructure. However, the following will be covered:

- The app itself: The app's decompiled codebase. An adversary could gain insight into security shortcomings, by reading its source code.
- Services and APIs: The communication between the app and its endpoints over the internet may be vulnerable to interception and decryption by an adversary if weak encryption is used. Sensitive data sent through these channels could be at risk if the adversary is able to break the encryption.

C. Relevant Security Properties & Policies

Personal health information is very sensitive and under strict laws. Therefore multiple security properties as relevant. The security properties which are considered most relevant are:

1) Confidentiality: Confidentiality is an important security property for a mobile app handling personal health data. The nature of the data is highly private and sensitive, and could cause emotional trauma for the individual if exposed to others without consent. All data therefore has to be encrypted, in such a way that only the individual and relevant medical employee can access it.

2) Privacy: Privacy is an essential consideration for any health app that stores and handles medical information. It allows individuals to control who has access to sensitive personal information about their health, such as diagnoses, treatments, and test results. So it is essential for the app to have strong privacy protections in place.

3) Authenticity: Health data is highly individual, and only you and your appointed doctor should be authenticated to access it.

4) Integrity: It is important that users have trust in the information they access. If a reply from your doctor is tampered with it, might have serious consequences. Like not using the correct dose of a proscription drug.

5) Availability: Availability is crucial for an app that handles medical records, as delays in accessing this time-sensitive information can have serious consequences. Many Danish citizens rely on the app for referrals, lab results, and scheduling appointments, so it is important for the system to have a high level of availability, to ensure users can access the resources they need when needed.

3. Software Security

This section examines the software security of Min Læge, by inspecting its decompiled code.

A. App Decompilation

As part of the investigation into the app's software security, we decompiled it using jadx[1] (version 1.4.5). A decompilation tool for Java programs that has support for APK files, the executable bundle format used by all Android applications. We obtained a copy of Min Læge's APK bundle from the website apkpure.com[2]. A third party site that host many of the apps that can be found on Google Play Store, where Min Læge is distributed.

1) Presence of Obfuscation: With `jadx` we were able to decompile the APK bundle, and reconstruct both source code in Java, and XML files containing the various string constants and configuration options used by the app. The entire code base appears to be written in Kotlin[3], a JVM[1] language that transpiles to Java. As a result there are multiple sections of the decompiled code, where it is hard to make sense of what the code

[1] Java Virtual Machine

```
771 public final java.lang.Object
772 setCprFromAccessToken(kotlin.coroutines.Continuation<? super kotlin.Unit> r6) {
773   /*
774           r5 = this;
775           boolean r0 = r6 instanceof
776   com.trifork.minlaege.auth.AuthController$setCprFromAccessToken$1 if (r0 == 0)
777   goto L14 r0 = r6
778           com.trifork.minlaege.auth.AuthController$setCprFromAccessToken$1 r0 =
779   (com.trifork.minlaege.auth.AuthController$setCprFromAccessToken$1) r0 int r1 =
780   r0.label r2 = -2147483648(0xffffffff80000000, float:-0.0) r1 = r1 & r2 if (r1
781   == 0) goto L14 int r6 = r0.label int r6 = r6 - r2 r0.label = r6 goto L19 L14:
782           com.trifork.minlaege.auth.AuthController$setCprFromAccessToken$1 r0 =
783   new com.trifork.minlaege.auth.AuthController$setCprFromAccessToken$1
784           r0.<init>(r5, r6)
785   L19:
786           java.lang.Object r6 = r0.result
787           java.lang.Object r1 =
788   kotlin.coroutines.intrinsics.IntrinsicsKt.getCOROUTINE_SUSPENDED() int r2 =
789   r0.label r3 = 2 r4 = 1 if (r2 == 0) goto L41 if (r2 == r4) goto L39 if (r2 ≠
790   r3) goto L31 java.lang.Object r0 = r0.L$0
```

Figure 1: Code snippet taken from the decompiled code, in file `sources/com/trifork/minlaege/auth/AuthController.java`. The body of the method `setCprFromAccessToken` is only partially decompiled. Shown in the multiline comment is what appears to some form of bytecode or intermediate language representation of the Java code, as frequent use of jumplabels, e.g. `L19:` on line 785, and `goto` statements, are the only form of control flow used. Instead of using descriptive variable names, all variables have been given semantically meaningless names, such as `r6`.

does. Many of Kotlins syntactical features like its null coalescing operator `?`, gets translated to a much wordier `Intrinsics.checkNotNullParameter(item, ”item”);`, increasing the visual noise when looking at the code. Many places, variable names have been shortened too small constants like `r1`. And for many methods that are not simply “getter” and “setter” methods, the tool failed to fully decompile it to its Java representation. Instead outputting a block of instructions that looks like a textual bytecode instruction format. See Figure 1 for an example that showcases both of these observations. Whether the obfuscation is due to active effort spent by the developers of the app, the build process of the Android framework, or limitations of jadx's ability to decompile code bases written in Kotlin is unclear. But it does make it significantly more difficult for an adversary trying to reverse engineer the app, in order to get insight into how its security mechanisms are implemented.

2) Manual Code Analysis: The decompiled code base is of considerable size. We measured it to be 168291[2] SLOC[3] when disregarding third party dependencies. Given the time frame and scope of the project, we deemed it insurmountable to manually analyze the program in its entirety. Even more so given the obfuscation and visual noise apparent in the output, see subsection 3.1. Instead we changed our methodology to look for appearances of specific keywords. Whose potential presence and context might make us able to do inferences in regards to the software security properties of the app.

The following keywords were searched for, but did not show up, or was of any significance:

- TLS
- X509
- 2FA
- HMAC
- RSA
- SHA{1,2,3}, SHA{256,512}
- MitID

[2]Result gathered using `tokei` [4] in the directory sources/com/trifork/minlaege.

[3]Source Lines of Code. A common measure for the size of a program's source code. Blank lines and comments are excluded, making it more accurate that just counting the number of lines in each file.

- NemID

The following keywords were searched for, and did show up 1 or more times:

1) AES

   In `auth/BiometricsHandlerHelper.java` AES is mentioned multiple times, see Figure 8. This utility class is used as part of the authentication procedure using the users stored biometric information in `auth/BiometricsCipherKeystore.java`. What is worth paying notice to is that CBC[4] is used as the AES block mode, and PKCS7 is used as the padding scheme. The combination of these two, can be exploited by an attack known as the "Padding Oracle Attack"[5]. Where an adversary can decrypt obtained ciphertext, without knowing the shared symmetric key, by using any error information obtained by the receiver that tells him whether the applied padding is correct or not. Although it is unclear to us, how an adversary could exploit this through the given attack surface, as laid out in subsection 2.2, it is nevertheless a good idea, to change it to a lesser exploitable block mode, such as AES-GCM[5], or use another type of padding scheme, not susceptible to the attack.

2) Initialization Vector

   In `auth/authBLL.java` a class is defined to do symmetric encryption using AES-CBC. CBC needs an initialization vector (IV) of equal size to the key-size, to initialize the block mode encryption and decryption. Instead of using a new random IV each time encryption is performed, the same static IV is used. This can be seen in Figure 9. An IV does not have to confidential, but should be unique, even more so when using CBC, as an adversary can be able to infer that the cleartext of two ciphertexts must be identical, if there ciphertext is identical, and they have been encrypted under the same key and IV[6]. Instead it would be better to use a cryptographically random IV instead each time a new block message is to be encrypted.

3) API keys

   Two static keys were found, `google_api_key` and `google_crash_reporting_api_key` in `resources/res/values/strings.xml`. Used to interact with Google specific services. Not deemed a security risk, as the service is related with crash analytics, and not anything related to users of the app.

[4]Cipher Block Chaining
[5]Galois Counter Mode

3) Real Time Protocol used for Video Consultation: Although we did not have a chance to inspect the data being transferred and received as part of the video consultation feature. We looked at its implementation, and the libraries it uses. WebRTC is used as the real-time communication protocol. It is a good choice in terms of security, as connections after the initial setup are peer to peer. Encryption of exchanged media streams are a requirement, and the collection of protocols are open source with full support from most mobile and browser providers[7]. WebRTC does not specify how the signalling process i.e. the establishment of a peer to peer duplex channel between two communicating parties, should be done, which can be exploited by a man in the middle attack, if no message authentication is utilized. From the decompiled code alone, it is not clear to us which security measurements are utilized as part of the signalling process[7]. A protocol scheme using TLS is the recommended way of doing it in a secure manner[7].

4) Static Analysis using MobSF: In addition to our own manual review, we also used MobSF[8] to statically analyze the APK file.

The overall security scores provided by MobSF are low, as it was rated 57/100, along with a medium risk grade. The scorecards can be observed in Figure 7 This contrasts our own findings, that are more positive. After looking into MobSF's findings more closely, we discovered that some of the raised issues are false positives, such as the following:

- "The app uses an insecure Random Number Generator". This is a false positive because it is only found to be used in an audio visualization library.
- "Application contains Privacy Trackers" This is a false positive as it is simply their Firebase crash reporting tracker.
- "Files may contain hardcoded sensitive information, such as usernames, passwords, and other confidential data". This rule is triggered whenever there are string literals containing keywords such as "key" or "password" that could indicate the presence of sensitive information. In the files we examined, we found no obvious security risks associated with this rule being triggered. The reason it was triggered, was because the files contained mock classes used for integration testing, and variables suffixed with the word "key", but their usage was limited to routing between different activities or fragments in the app.

MobSF completely missed out on more important security issues, such as the static IV, which we found by our own independent code inspection.

However, the static analysis tool found that the Min Læge app uses certificate pinning, which is a good thing, as it

prevents man in the middle attacks. This will be further expanded upon in subsection 4.1. Furthermore the app also found the same hardcoded API keys that we found in subsubsection 3.1.2.

Disregarding the false positives provided by MobSF, the provided security scores would likely align closer to the result of our analysis.

4. Network Security

In this section, we examine the app's network security and its ability to withstand man in the middle attacks. We evaluate the TLS configuration of the services the app is found to communicate with over HTTPS.

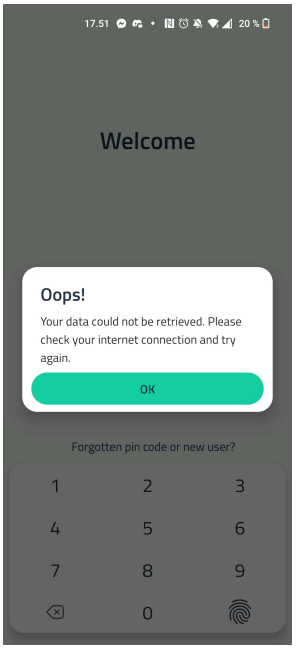


Figure 2: Failure to login under a man in the middle attack, when Min Læge's APK file has not been patched.

A. Man In The Middle Attack

A man-in-the-middle (MITM) attack was performed using the mitmproxy tool[9]. The setup consisted of a Oneplus Nord running Android 12 with Min Læge installed, and a laptop running mitmproxy in transparent proxy mode on an ordinary home network. The phone was configured to route traffic through the laptop. We first attempted to inspect app traffic without adding an additional X.509 certificate to the phone's list of trusted certificates. However, this did not succeed, as opening Min Læge and authenticating with a pin or fingerprint immediately led to an error prompt, shown in Figure 2, preventing further authenticated use of the app. Presumably because either Android or Min Læge itself were able to detect the proxy being a distrusted device. Next up we escalated the attack by installing a certificate generated by `mitmproxy` on the device, as a trusted user CA certificate. Adding the certificate did not change the outcome. The reason for


```
{
    "BooleanValue": null,
    "CitizenID": "e8144473-[redacted]",
    "Code": "DNK35312",
    "CodeMeaning": "SARS-CoV-2 (RNA), Borger\n",
    "CodeMeaningSimplyfied": "SARS-CoV-2(RNA);Sekr(spec.)",
    "Created": null,
    "DateTimeValue": null,
    "Documents": {},
    "DurationValue": null,
    "End": null,
    "Initials": "",
    "LabSampleID": "1ea68b46-4245-43a6-8df4-7d4c2c808f5a",
    "NumericValue": null,
    "Operator": 0,
    "ReferenceFrom": null,
    "ReferenceTo": null,
    "RelevantType": "Text",
    "Start": "2022-10-03T09:14:00Z",
    "TextValue": "Påvist",
    "Updated": null,
    "Value": "Påvist"
},
```


Figure 3: Example of a lab result data received in Min Læge. Proving the user has tested positive for Corona. Part of `CitizenID` 's value has been redacted on purpose.

this is the apps use of certificate pinning. As mentioned in subsubsection 3.1.4, certificate pinning[6] is used. This has been standard to have in apps' manifest file since Android 7[10]. At this point, it is fair to conclude that Min Læge is not vulnerable to man in the middle attacks, given the considered attack surface that would be exposed under normal conditions.

1) Patching APK with a Malicious Root Certificate: As a last resort to circumvent the found usage of certificate pinning, we modified the manifest of the APK file, and reinstalled it onto the device. The modification consisted of adding the certificate generated by `mitmproxy` to the list of pinned certificates, using a program called `apk-mitm` [11]. This enabled us to successfully execute the attack, and advance beyond the authentication mechanism. From here we were now able to see all traffic going to and from the app in clear text. The success of this approach should not be seen as critique of the app, as it is not a realistic attack surface. It requires a malicious actor to trick the victim into downloading and installing their patched version of the APK, and later sniff the traffic generated by it. Nevertheless it is worthwhile to do in order, to inspect which services it communicates with, and the nature of the data being communicated.

2) Sniffed Communication: The first thing noted about the communication is that all data for the initial application view is fetched every time the app is restarted. Figure 10 shows sniffed traffic being sent right after the user has authenticated. Meaning that, none of the users

[6]Certificate pinning is the practice of associating a host with its expected X509 certificate or public key to prevent man-in-the-middle attacks.

```
[
    {
        "appVersion": "2.6.5.2503",
        "category": "login",
        "device": "OnePlus AC2003",
        "deviceId": "e2197b52-[redacted]",
        "logLevel": "INFO",
        "message": "Login gennemført med biometri (issuer: https://oidc.sundhedsdatastyrelsen.dk/auth/realms/sds",
        "osVersion": "31",
        "timestamp": "2022-12-15T17:36:25.603+01:00"
    }
]
```


Figure 4: The only data that Min Læge was found to log. Part of `deviceId`'s value has been redacted on purpose.

data such as vaccination, lab results, referrals, diagnoses, etc. are cached to the device's storage. Furthermore, only the data that is needed in the current activity is actually received. An example of which can be demonstrated by pressing the more tab, then lab results, this initiates a GET request to receive the lab results as seen in Figure 3. We assess this decision to be beneficial for the privacy of the user. But only applies so long the communication between the app and the services are transmitted securely, as they are now.

Little data is sent from the client to external services. The only logging type of data we observed being sent to an external service, happens after logging in. The contents of this packet can be observed in Figure 4, it contains information about the method of authentication used, being biometric in this case, app version, device model, device ID, and a timestamp. Considering the threat model presented in section 2, we deem this level of logging acceptable, as it is quite tame and does not include any private information, other than the device type.

B. Examination of Services

The app was found to be communicating with numerous endpoints. We collected and analyzed all such endpoints and services found during our static analysis and `mitmproxy` attack. A compilation of endpoints can be observed in Table I. One type of analysis made use of Qualys SSL Labs' Server Testing tool, which can be used to test SSL/TLS configurations of any public facing web server. The tool provides a detailed description of supported protocols by the web server, and a summarizing grade from A+ to F. The endpoints, grades and a short description is compiled into Table I. The findings show that the endpoints are compliant with most industry standards, as all endpoints received an A or A+.

5. Authentication

As hinted at in the network security section, see section 4, a user has to be authenticated before being able to access any of their data.

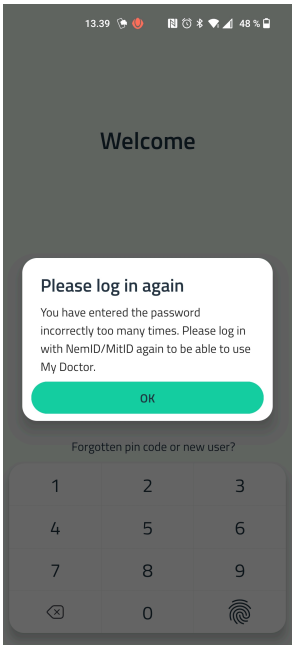


Figure 5: Maximum PIN login attempts exceeded, after 3 tries.

Authentication in Min Læge is split into two parts. Initial authentication the first time the app is started after it being installed, and subsequent authentication mechanisms after a user has proved their identity.

Min Læge makes use of MitID or NemID to authenticate the citizen the first time the app is used. We will only investigate the integration with MitID, as NemID is being phased out at the time of this writing. After the initial login, the user is prompted to create a 4-digit PIN code, and optionally add a biometric login method for subsequent logins. The user can at any point revoke the biometric option or completely sign out as a user.

A. Security of Authentication

Due to the usage of MitID and NemID, two factor authentication is utilized to initially sign in. Subsequent logins only require single factor, as only a PIN or a biometric signature is needed. However, the app disables

URL	Grade	Reason
https://minlaegeapp.dk	A	SSL Labs was able to get a certificate from the TLS server without using SNI in the ClientHello message of the TLS handshake. SNI[12] is an acronym for Server Name Identicator, which is an extension to the TLS protocol. It allows a physical host to host multiple domains on the same IP address, without having the domains sharing the same digital certificate. Not using SNI can be problematic, as it means that the server will not be able to present the correct certificate, potentially allowing an attacker to impersonate the server and carry out a man-in-the-middle attack.
https://dawa.aws.dk	A	No SNI support.
https://oidc.hosted.trifork.com	A	Only supports TLS v1.2, and multiple weaker cipher suites like TLS_RSA_WITH_AES_128_CBC_SHA
https://oidc.sundhedsdatastyrelsen.dk	A	Only supports TLS v1.2, and multiple weaker cipher suites like TLS_RSA_WITH_AES_128_CBC_SHA
https://prod.mlapp.dk	A	Supports both TLS v1.2 and v1.3. When using v1.2 the cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 can be selected, which is labelled as WEAK by SSL Labs.
https://rooms.vconf.dk	A+	N/A
https://testpatientportal.egclinea.com	A	Supports TLS v1.2 but not v1.3.
https://trifork.com	A	Supports TLS v1.2 but not v1.3.
https://www.sundhed.dk	A+	N/A

Table I: A combined list of HTTPS URLs found using the decompiled APK and the `mitmproxy` analysis, which the application communicates with. Each URL has been tested using Qualys SSL Labs Server Test[13]. The overall assessment score is given in column 2, and the reason for why it did not achieve the highest grade, that is A+, is listed in the third column.

the biometric option after 5 rejected fingerprint/FaceID attempts, falling back to only providing the PIN option. The PIN option only allows 3 attempts, failing at this, the app will completely sign the user out of the app as seen in Figure 5, requiring the initial two-factor authentication using MitID or NemID once again. We deem this to be a good design, as it disallows brute forcing attempts, which would otherwise be feasible due the low number of symbols in the input domain i.e. $10^4 = 10000$. With 3 different attempts only $3/10^4 = 0.03\%$ of the domain can be searched.

As covered in subsubsection 3.1.2 the way MitID is being used is unclear from looking at the decompiled code, as there is no explicit mention of it. As part of the man in middle attack, see subsection 4.1, we tested the initial authentication step with MitID, to see if it would be possible to intercept the messages being sent. This failed, even when using the patched APK file. However, this attempt was unsuccessful, likely due to the MitID login prompt being loaded in an external WebView that does not utilize the list of pinned certificates specified in the app's manifest.

Inspecting the sniffed traffic gave us insight into how the authenticated session is established after the user has logged in. Three requests are sent at the start of a session to two different servers subdomain `oidc` [7]. The three requests appears to be used to prove that the client has previously authenticated with MitID, and are marked in red in Figure 10. The first request is sent to the domain `oidc.hosted.trifork.com` where two parameters are passed, `longsecret` and `keyid`. The server responds with a value called `key`, and can be seen in Figure 11. Leading us to believe that the initial MitID authentication generates the two parameters, which the app then stores in the internal app-specific storage provided by Android[15]. To then later be used for subsequent sessions. With the key the client then sends a request to `oidc.sundhedsdatastyrelsen.dk`, and gets the response seen in Figure 13. Similar to a TLS handshake the server responds with a record of the types of asymmetric and symmetric encryption it supports for the cipher suite negotiation. To which the client replies and finishes the handshake. Getting as response the list of ephemeral tokens shown in Figure 12. From here, the tokens are used in every request until the users closes the app, or the session duration exceeds `expires_in` = 1800 seconds.

[7] oidc is short for OpenID Connect[14]. A open standard to use trusted third party identity providers, such as MitID, for authentication.

B. What Could be Improved

In our analysis of the current authentication mechanism, we found that MitID making use of OpenID Connect, an open and verified standard, effectively ensures the security of the system. We did not in our analysis identify any potential improvements that could be made, to further enhance the mechanism.

Additionally, the implementation of a limited number of login attempts, adds an additional layer of protection against unauthorized access. Based on these findings,

we believe that the current authentication mechanism is sufficient for the needs of the system.

6. Privacy

Min Læge is mindful of the users' right to privacy. No integrations with external unrelated services were found during the assessment of the software and network security. The only data found to be logged about the user, are characteristics about the physical device the app is running on, see section 4.

A. Android Permissions

Applications running under the Android operating system, are required to declare which device resources they want to access, in order to perform their intended functionality. This is done by listing the permissions in the AndroidManifest.xml file[16]. The permissions requested by Min Læge are shown in Table II. Included in the table is our assessment of the soundness of the permissions given the functionality of the app.

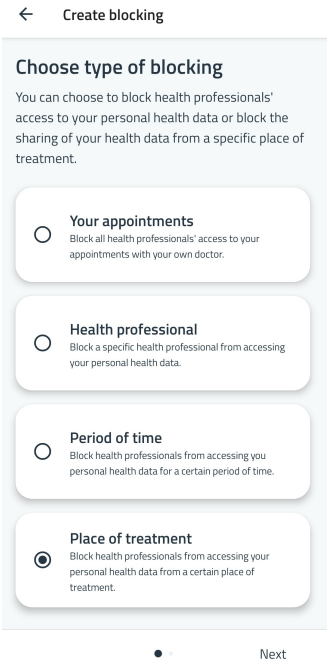


Figure 6: The types of blocks that a citizen can set.

Overall there are no egregious use of permissions. As they are all sensible. However `READ_EXTERNAL_STORAGE` and the corresponding `WRITE_EXTERNAL_STORAGE` seems to be unnecessary. But since these are categorized by Android as "Runtime Permissions"[8], therefore seen as harmless when considering their security impact. The app is in compliance with the minimal viable security principle, that dictate to request as few permissions as possible to complete a task. Permissions that a user might want to decline such as `CAMERA` are only prompted for, when the user activates the part of the app where the permission is needed.

[8]Permissions that requires the user to explicitly opt-in by accepting a prompt during runtime [17].

B. Privacy Features

The app allows the user to have fine-grained control, over which health professionals should be blocked from accessing their personal health data, appointments, and data from certain places of treatment. Figure 6 also shows that the block can be set to only be active for a set period of time.

7. Conclusion

To sum up, the Min Læge app has strong software security measures, including obfuscation and protection against decompilation. While some issues were identified during manual code analysis, such as the use of a static IV and the combination of AES, CBC, and PKCS#7 cipher, which is vulnerable to the padding oracle attack, the app still demonstrated a high level of security overall.

The app's network security is also robust, with protection against man-in-the-middle attacks using certificate pinning. In addition, very little data is logged about the user, ensuring a high level of privacy.

The app's authentication system, which uses MitID, is secure and allows very few attempts for logging in by PIN and biometric authentication, before singing the user out.

Finally, the app respects users' privacy and allows users to block selected health professionals from accessing their data. It also has sensible Android permissions in place. Overall, we deem the Danish Min Læge app to be a secure digital health platform.

References

[1] skylot, "jadx," 2014. [Online]. Available: https://github.com/skylot/jadx

[2] "Min læge - download - apppure." [Online]. Available: https://apkpure.com/min-l%C3%A6ge/com.trifork.minlaege

[3] JetBrains, "Kotlin programming language," 2022. [Online]. Available: https://kotlinlang.org/

[4] XAMPPRocky, "tokei," 2022. [Online]. Available: https://github.com/XAMPPRocky/tokei

[5] D. F. Aranha, "Network security - security in practice - slides 2," 2022, lecture slides presented at Aarhus University, pp. 15-16.

[6] ——, "Network security - crash course on cryptography - slides 1," 2022, lecture slides presented at Aarhus University, pp. 47.

[7] "Webrtc security," accessed on 11-12-2022. [Online]. Available: https://webrtc-security.github.io/

[8] MobSF, "Mobile security framework (mobsf)," accessed on 11-12-2022. [Online]. Available: https://mobsf.github.io/Mobile-Security-Framework-MobSF/

[9] mitmproxy, "mitmproxy," 2022. [Online]. Available: https://mitmproxy.org/

Permission	Soundness
INTERNET	Valid. All of the listed functionalities in subsection 1.1 are dependent on making requests over the internet.
ACCESS_NETWORK_STATE	Valid. As shown in section 4, the app does not cache data local on the client device, but fetches it dynamically upon the user authenticating successfully.
READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE	Unclear. The combination of both permissions allows an app to read and write to the external storage. The permissions are checked in a utility class for saving an image, which is used by the inbox's new message form, to add an attachment. When we try to fill out the form ourselves, we do not see an option to select a file attachment. Making the permissions unnecessary.
WAKE_LOCK	The decompiled code doesn't make it clear where the permission is required, as it's only specified in the manifest. We think it's probably necessary for the video consultation feature to prevent the screen from locking during longer consultations.
CAMERA RECORD_AUDIO	Valid. The video consulting feature needs access to the camera and microphone in order for the doctor or nurse to meaningfully consult the user.
MODIFY_AUDIO_SETTINGS BLUETOOTH BLUETOOTH_CONNECT	Requested in the Video Conference Activity for when the user uses a Bluetooth headset and microphone. Not clear why this would have to be managed at the application level, and not by the operating system itself.
ACCESS_COARSE_LOCATION ACCESS_FINE_LOCATION	Valid. Used for the duty doctor feature. When granted the feature, will connect you to the nearest medical clinic when you call the duty doctor.
USE_BIOMETRIC USE_FINGERPRINT	Valid. To authenticate after initial authentication the users' fingerprint can be used to login. See section 5 for more detail.

Table II: Android permissions requested by Min Læge. The first column contains the permissions grouped by related functionality. The second column contains our assessment of the permissions validity relative to the functionality of the app, as described in subsection 1.1.

[10] M. Pandey, "Ssl pinning in android," 2022, accessed on 16-12-2022. [Online]. Available: https://mailapurvpandey.medium.com/ssl-pinning-in-android-90dddfa3e051

[11] shroudedcode, "apk-mitm," 2019. [Online]. Available: https://github.com/shroudedcode/apk-mitm

[12] W. contributors, "Server name indication," 2021, [Online; accessed 7-December-2022]. [Online]. Available: https://en.wikipedia.org/wiki/Server_Name_Indication

[13] "Ssl labs server test - qualys," [Online; accessed 7-December-2022]. [Online]. Available: https://www.ssllabs.com/ssltest/

[14] O. Foundation, "Openid connect," 2022, accessed on 16-12-2022. [Online]. Available: https://openid.net/connect/

[15] Google, "App-specific storage in android," 2022, accessed on 16-12-2022. [Online]. Available: https://developer.android.com/training/data-storage/app-specific

[16] "Declaring permissions in android," accessed on 16-12-2022. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission

[17] "Android runtime permissions," accessed on 16-12-2022. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview#runtime

8. Appendix

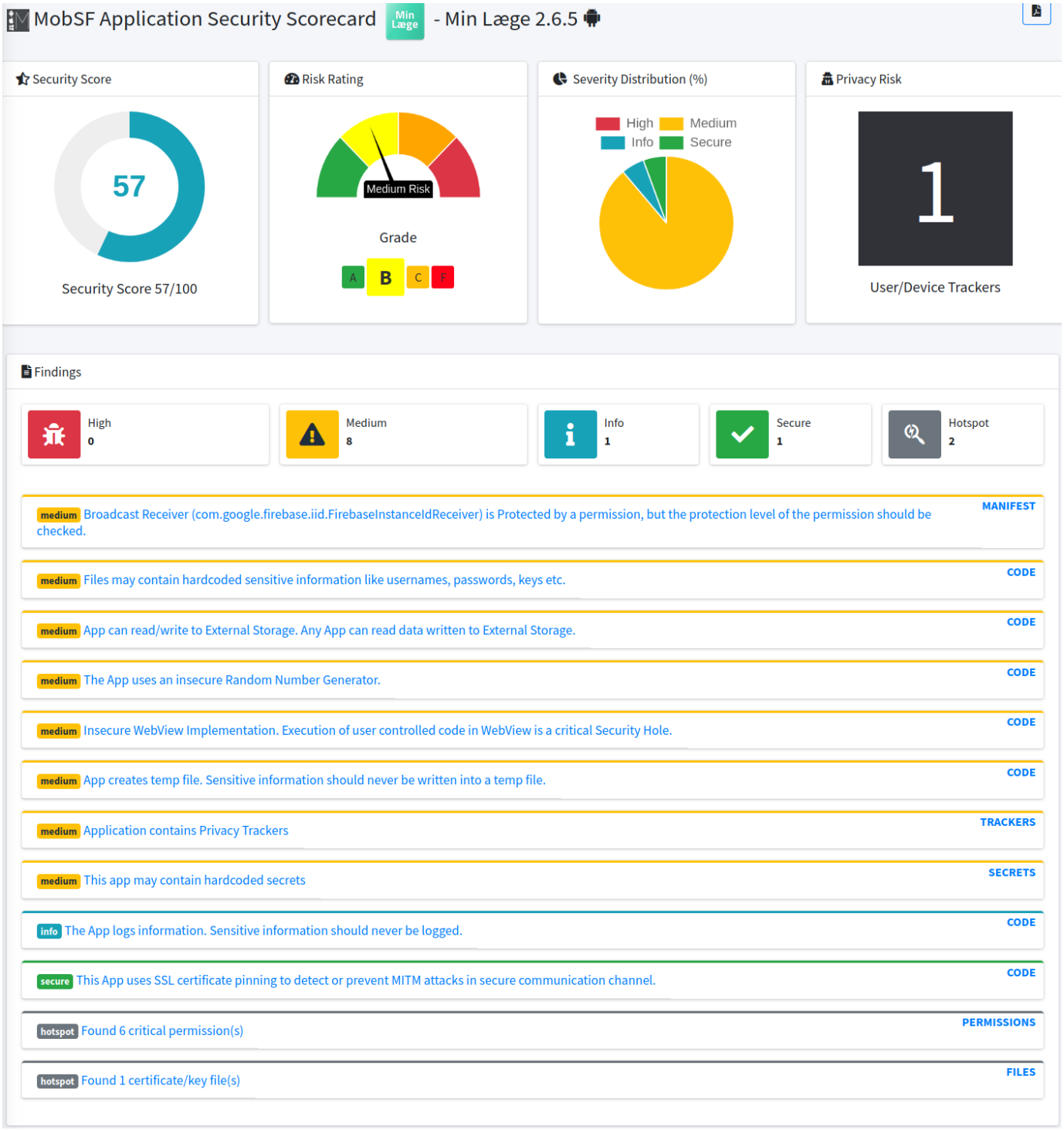


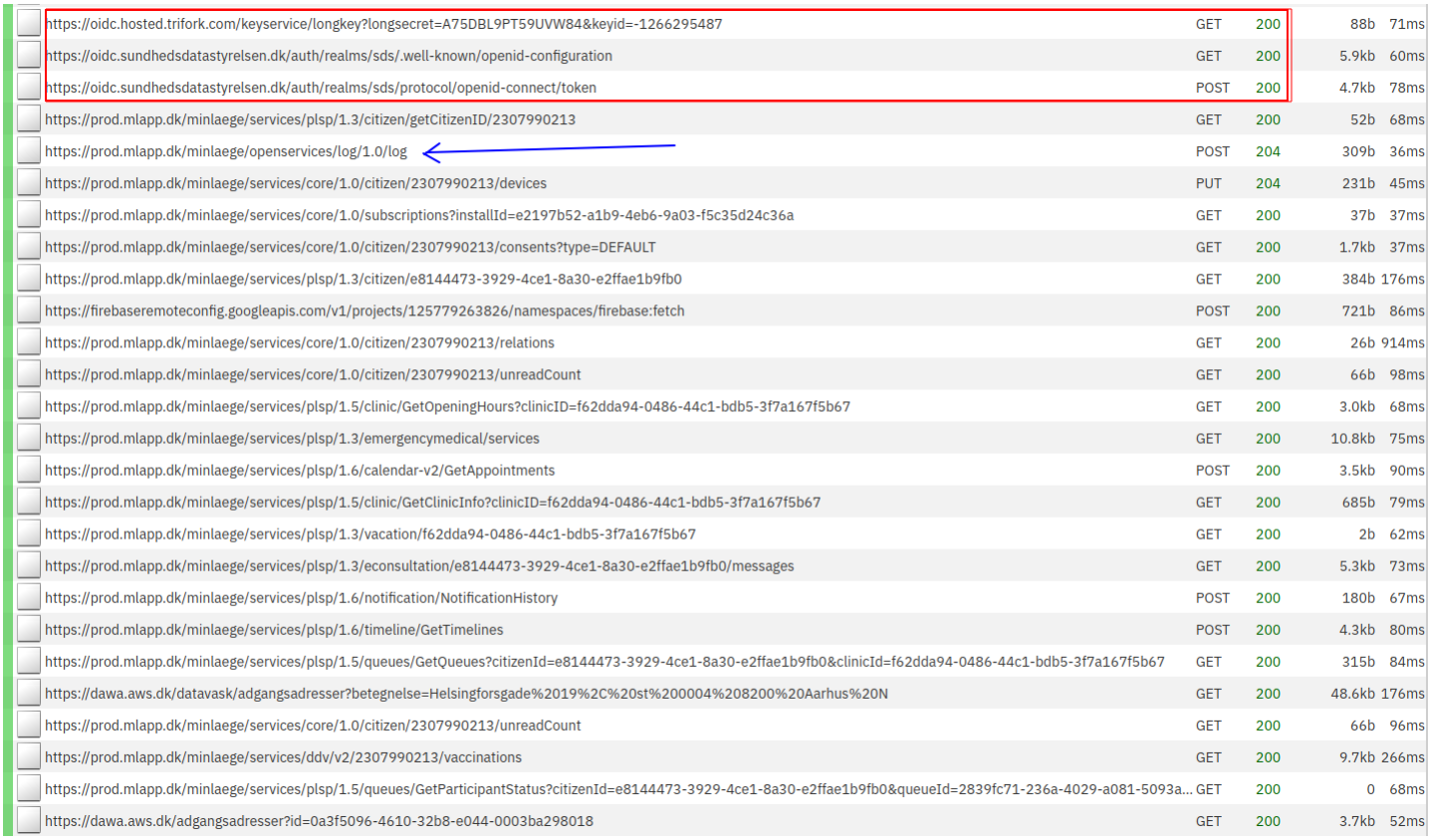
Figure 7: Security score and grade provided by MobSF. Along with the most prominent security risks found by its static analysis.

```
22 private final String getAesAlgorihm() { return "AES/CBC/PKCS7Padding"; }
23
24 public final KeyStore getKeystore() {
25   try {
26     KeyStore store = KeyStore.getInstance(Constants.keyStoreName);
27     store.load(null);
28     Intrinsics.checkNotNullExpressionValue(store, "store");
29     return store;
30   } catch (KeyStoreException e) {
31     throw new RuntimeException("Failed to get an instance of KeyStore", e);
32   }
33 }
34
35 public final Cipher createKeystoreCipher() {
36   Cipher cipher = Cipher.getInstance(getAesAlgorihm());
37   Intrinsics.checkNotNullExpressionValue(cipher,
38                                          "getInstance(getAesAlgorihm())");
39   return cipher;
40 }
41
42 public final KeyGenerator createCipherKeyGenerator() {
43   try {
44     KeyGenerator cipherKeyGenerator =
45         KeyGenerator.getInstance("AES", Constants.keyStoreName);
46     KeyGenParameterSpec.Builder encryptionPaddings =
47         new KeyGenParameterSpec.Builder("alias", 3)
48             .setBlockModes("CBC")
49             .setUserAuthenticationRequired(true)
50             .setEncryptionPaddings("PKCS7Padding");
51     Intrinsics.checkNotNullExpressionValue(
52         encryptionPaddings,
53         "Builder(\"alias\", keyProp…ENCRYPTION_PADDING_PKCS7)");
54     cipherKeyGenerator.init(encryptionPaddings.build());
55     Intrinsics.checkNotNullExpressionValue(cipherKeyGenerator,
56                                            "cipherKeyGenerator");
57     return cipherKeyGenerator;
58   } catch (Exception e) {
```

Figure 8: Found in auth/BiometricsHandlerHelper.java

```java
public final class authBll {
  private static final String aesCipher = "AES";
  private static final String aesCipherCbc5Padding = "AES/CBC/PKCS5Padding";

  /* renamed from: iv */
  private static final IvParameterSpec f304iv;

  static {
    byte[] bArr = new byte[16];
    for (int i = 0; i < 16; i++) {
      bArr[i] = 0;
    }
    f304iv = new IvParameterSpec(bArr);
  }

  public static final byte[] encrypt(KeyModel keyModel, byte[] data) {
    Intrinsics.checkNotNullParameter(keyModel, "<this>");
    Intrinsics.checkNotNullParameter(data, "data");
    Key aesKey = aesKey(keyModel);
    if (aesKey == null) {
      return null;
    }
    Cipher it = Cipher.getInstance(aesCipherCbc5Padding);
    it.init(1, aesKey, f304iv);
    Intrinsics.checkNotNullExpressionValue(it, "it");
    return doFinalNoException(it, data);
  }

  public static final byte[] decrypt(KeyModel keyModel, byte[] data) {
    Intrinsics.checkNotNullParameter(keyModel, "<this>");
    Intrinsics.checkNotNullParameter(data, "data");
    Key aesKey = aesKey(keyModel);
    if (aesKey == null) {
      return null;
    }
    Cipher it = Cipher.getInstance(aesCipherCbc5Padding);
    it.init(2, aesKey, f304iv);
    Intrinsics.checkNotNullExpressionValue(it, "it");
    return doFinalNoException(it, data);
  }
```

Figure 9: Found in `auth/authBll.java`



https://oidc.hosted.trifork.com/keyservice/longkey?longsecret=A75DBL9PT59UVW84&keyid=-1266295487	GET	200	88b	71ms
https://oidc.sundhedsdatastyrelsen.dk/auth/realms/sds/.well-known/openid-configuration	GET	200	5.9kb	60ms
https://oidc.sundhedsdatastyrelsen.dk/auth/realms/sds/protocol/openid-connect/token	POST	200	4.7kb	78ms
https://prod.mlapp.dk/minlaege/services/plsp/1.3/citizen/getCitizenID/2307990213	GET	200	52b	68ms
https://prod.mlapp.dk/minlaege/openservices/log/1.0/log	POST	204	309b	36ms
https://prod.mlapp.dk/minlaege/services/core/1.0/citizen/2307990213/devices	PUT	204	231b	45ms
https://prod.mlapp.dk/minlaege/services/core/1.0/subscriptions?installId=e2197b52-a1b9-4eb6-9a03-f5c35d24c36a	GET	200	37b	37ms
https://prod.mlapp.dk/minlaege/services/core/1.0/citizen/2307990213/consents?type=DEFAULT	GET	200	1.7kb	37ms
https://prod.mlapp.dk/minlaege/services/plsp/1.3/citizen/e8144473-3929-4ce1-8a30-e2ffae1b9fb0	GET	200	384b	176ms
https://firebaseremoteconfig.googleapis.com/v1/projects/125779263826/namespaces/firebase:fetch	POST	200	721b	86ms
https://prod.mlapp.dk/minlaege/services/core/1.0/citizen/2307990213/relations	GET	200	26b	914ms
https://prod.mlapp.dk/minlaege/services/core/1.0/citizen/2307990213/unreadCount	GET	200	66b	98ms
https://prod.mlapp.dk/minlaege/services/plsp/1.5/clinic/GetOpeningHours?clinicID=f62dda94-0486-44c1-bdb5-3f7a167f5b67	GET	200	3.0kb	68ms
https://prod.mlapp.dk/minlaege/services/plsp/1.3/emergencymedical/services	GET	200	10.8kb	75ms
https://prod.mlapp.dk/minlaege/services/plsp/1.6/calendar-v2/GetAppointments	POST	200	3.5kb	90ms
https://prod.mlapp.dk/minlaege/services/plsp/1.5/clinic/GetClinicInfo?clinicID=f62dda94-0486-44c1-bdb5-3f7a167f5b67	GET	200	685b	79ms
https://prod.mlapp.dk/minlaege/services/plsp/1.3/vacation/f62dda94-0486-44c1-bdb5-3f7a167f5b67	GET	200	2b	62ms
https://prod.mlapp.dk/minlaege/services/plsp/1.3/econsultation/e8144473-3929-4ce1-8a30-e2ffae1b9fb0/messages	GET	200	5.3kb	73ms
https://prod.mlapp.dk/minlaege/services/plsp/1.6/notification/NotificationHistory	POST	200	180b	67ms
https://prod.mlapp.dk/minlaege/services/plsp/1.6/timeline/GetTimelines	POST	200	4.3kb	80ms
https://prod.mlapp.dk/minlaege/services/plsp/1.5/queues/GetQueues?citizenId=e8144473-3929-4ce1-8a30-e2ffae1b9fb0&clinicId=f62dda94-0486-44c1-bdb5-3f7a167f5b67	GET	200	315b	84ms
https://dawa.aws.dk/datavask/adgangsadresser?betegnelse=Helsingforsgade%2019%2C%20st%200004%208200%20Aarhus%20N	GET	200	48.6kb	176ms
https://prod.mlapp.dk/minlaege/services/core/1.0/citizen/2307990213/unreadCount	GET	200	66b	96ms
https://prod.mlapp.dk/minlaege/services/ddv/v2/2307990213/vaccinations	GET	200	9.7kb	266ms
https://prod.mlapp.dk/minlaege/services/plsp/1.5/queues/GetParticipantStatus?citizenId=e8144473-3929-4ce1-8a30-e2ffae1b9fb0&queueId=2839fc71-236a-4029-a081-5093a...	GET	200	0	68ms
https://dawa.aws.dk/adgangsadresser?id=0a3f5096-4610-32b8-e044-0003ba298018	GET	200	3.7kb	52ms

Figure 10: Initial content fetch upon login. The 3 requests contained within the red border, make up the authentication setup. The POST request pointed at by a blue arrow highlights data about the users device, that are logged to a server.

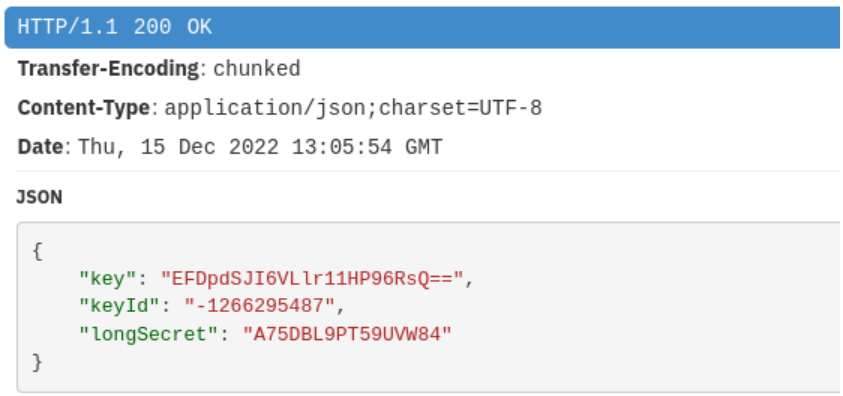


Figure 11: The response from the GET request to `oidc.hosted.trifork.com/keyservice/longkey?longsecret=A75DBL9PT59UVW848keyid=-1266295487`. The `keyId` and `longSecret` are stored on the client, and sent at the start of a session to get the `key`.

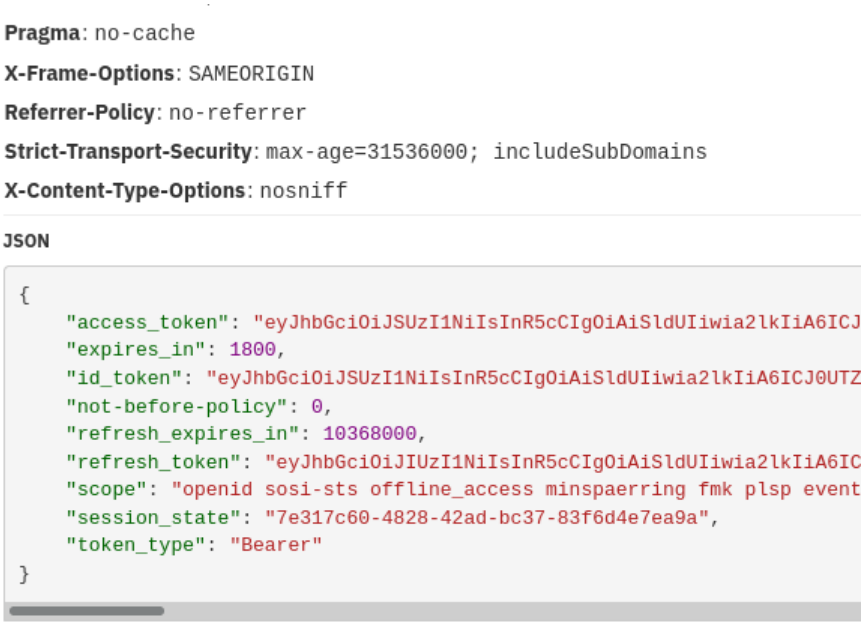
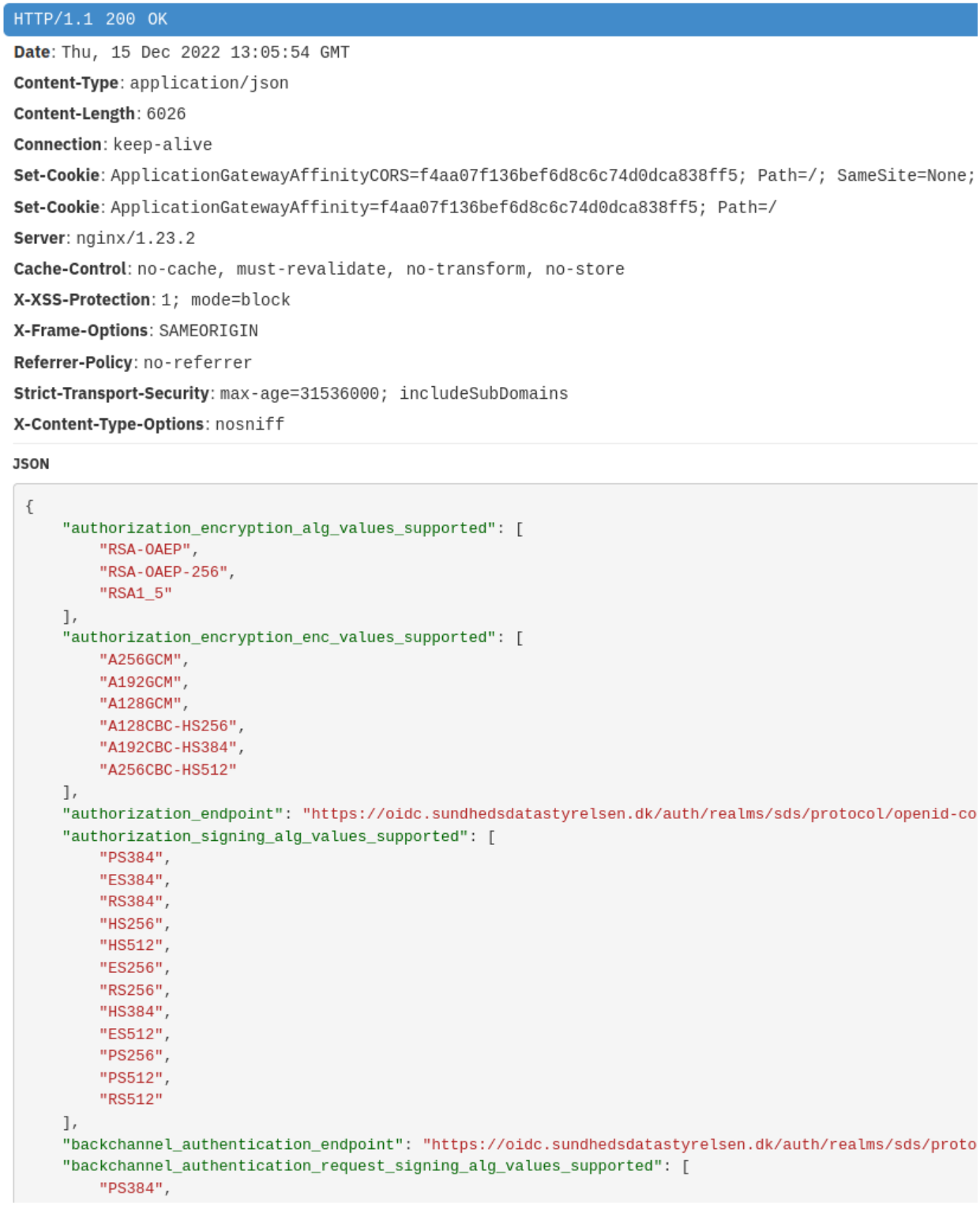


Figure 12: Reply from the GET request to `oidc.sundhedsdatastyrelsen.dk/auth/realms/sds/protocol/openid-connect/token`. The response contains the ephemeral tokens that proves the client is authenticated during the session.



```
HTTP/1.1 200 OK
Date: Thu, 15 Dec 2022 13:05:54 GMT
Content-Type: application/json
Content-Length: 6026
Connection: keep-alive
Set-Cookie: ApplicationGatewayAffinityCORS=f4aa07f136bef6d8c6c74d0dca838ff5; Path=/; SameSite=None;
Set-Cookie: ApplicationGatewayAffinity=f4aa07f136bef6d8c6c74d0dca838ff5; Path=/
Server: nginx/1.23.2
Cache-Control: no-cache, must-revalidate, no-transform, no-store
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-Options: nosniff

JSON

{
    "authorization_encryption_alg_values_supported": [
        "RSA-OAEP",
        "RSA-OAEP-256",
        "RSA1_5"
    ],
    "authorization_encryption_enc_values_supported": [
        "A256GCM",
        "A192GCM",
        "A128GCM",
        "A128CBC-HS256",
        "A192CBC-HS384",
        "A256CBC-HS512"
    ],
    "authorization_endpoint": "https://oidc.sundhedsdatastyrelsen.dk/auth/realms/sds/protocol/openid-co
    "authorization_signing_alg_values_supported": [
        "PS384",
        "ES384",
        "RS384",
        "HS256",
        "HS512",
        "ES256",
        "RS256",
        "HS384",
        "ES512",
        "PS256",
        "PS512",
        "RS512"
    ],
    "backchannel_authentication_endpoint": "https://oidc.sundhedsdatastyrelsen.dk/auth/realms/sds/proto
    "backchannel_authentication_request_signing_alg_values_supported": [
        "PS384",
```

Figure 13: Part of the reply from the GET request to `oidc.sundhedsdatastyrelsen.dk/auth/realms/sds/.well-known/openid-configuration`. The response is a record of which asymmetric and symmetric cipher suites it supports for the handshake.