Security Analysis of LHV mobile application

Peeter Tarvas

August 15, 2025

1 Abstract

In this report, I describe a security analysis of the LHV mobile application [1] developed by LHV Bank [2]. This app provides banking actions to their clients. A threat model was made and analyzed by looking at software security, network security using commonly available tools, as well as analysis and discussions of authentication and privacy aspects of the app. There were several issues found during the analysis, some of those being weak token generation for biometric login, exposed API keys in client-side code and not using TLS 1.3. Moreover, the report discusses the login methods and assesses the privacy of the application.

2 Introduction

The emergence of digital banking has changed mobile applications into high-value targets for criminals, combining financial assets with vast repositories of personally identifiable information. LHV Pank, as one of Estonia's biggest financial institutions, exemplifies this shift, where security mechanisms are not only optional but critical to maintaining user trust and regulatory compliance. This report evaluates LHV's mobile banking application, analyzing its defenses against data breaches, unauthorized transactions, and privacy violations. In this report, the application will first be described, followed by a documentation of the threat model and important security properties of the application. After that, the application will be analysed and important aspects of the application will be identified. Then an experiment and results section will follow where the application will be decompiled to be able to scan the underlying code. The network activity will be monitored and documented to ensure proper networking done by the application. The authentication mechanisms will be described and scrutinized in detail and the privacy which the application offer will be documented. After this, the results will be discussed. A conclusion will then follow.

3 Description of the mobile application

The mobile app analyzed is LHV's banking application available at Play Store [3]. The application provides general activities for the the customers of the bank. This includes getting an overview of transfers, making transfers, trading investment items, looking at general information like pensions and many more use cases. Thus, the LHV application should be very secure because it holds privacy-sensitive information and grants access to a individual's personal wealth. The application is both offered on iOS and Android platforms. In this report the Android application will be analysed to the underlying application code and emulators.

3.1 Threat model

Banking applications are very sensitive. This is because they have permissions to handle monetary transactions. Moreover, banks require their clients to be validated and for this they gather personal information like ID codes, emails, phone numbers and home addresses. This data gathering has ramped up in the recent decade when vast amounts of data are used for machine learning models [4] and since the Danske Bank/Swedbank money laundering scandal [5], the ramping up of anti-money laundering operations requires much more personal data to be collected. This leads to the situation that banks have to keep a lot of personal information and also makes them a very rewarding but usually hard-to-hack target for bad actors. The threat model then includes a large pool of different adversaries with different goals:

- Individuals/Organizations who mine and gather personal information to sell it online or to use it in an advantageous way for profit. This problem could be realized if it is possible to get information about the user by parameters and responses sent in the network traffic. The worst case scenario here would be that the hackers get their hands on the data warehouse records of a bank through some application exploit.
- Traders/Funds who could benefit from knowing the victim's positions in stocks/bonds/options

and on how those positions could have changed in a given time-frame. Wealthy individuals like CEOs and Board Members would be a very profitable target for criminals in this case since they can have insider information that leads them to make transactions before anyone else knows about them.

- Criminals that are interested in stealing money from the victim by exploiting transfer vulnerabilities in the application.
- Advertisers/Credit companies could use data to identify individuals who take out debt more easily than others to target adds specifically tailored to the victim.

While I'm not a head of an organization, I can still try some of the attack vectors that bad actors might use in achieving their goals. The aim of this report then is to find out if it is possible to identify methods that enable gathering as much personal information as possible and seeing investment positions through some exploit by finding leaks in the code base or performing a man-in-the-middle attack.

3.2 Security policies

As shown in the previous section, banking data is very sensitive and can be used in all kind of ways to profit of security vulnerabilities. Proper policies must be defined to ensure that potential adversarial capabilities are mitigated.

3.2.1 Confidentiality

Financial status and activity must be confidential. This means that the activities performed in the banking application must be kept secret. Outsiders should not be allowed to eavesdrop on non-encrypted financial activities as this could have negative consequences.

3.2.2 Integrity

Financial positions must keep its integrity. This means that there should be assurance that the information cannot be changed by a third party and that financial information is truthful.

3.2.3 Availability

The less physical money we have on our hands and the more online-only solutions there are, like paying debt or rent, the more important availability becomes. Availability is also important to customers who have many bank accounts and use a system of accounts to save money and keep financials in check. It's also important to speculators who could be blocked from the application in a crisis and thus not being able to sell bad/speculative trades because of that.

3.2.4 Authenticity

For keeping information secret, authentication also becomes a security property. The application should only allow people to see their own data but not the data of other people.

3.2.5 Non-repudiation

Users of the mobile application, shops and the bank should be able to trust that they in fact carried out the operation. Therefore, the user should not be able to deny an operation they in fact conducthed through the application.

3.2.6 Reliability

The application should perform in a reliable way. An example of this is that the user should not be able to transfer money to someone else by accident.

4 Analysis

The following section contains analysis of the application. It covers software- and network security, authentication mechanisms and privacy. Existing software tools are used to investigate the application and discover potential security vulnerabilities.

In order to meet the necessary security properties for a well secured mobile application a few expectations have to be met.

Firstly, to have necessary confidentiality, TLS/HTTPS is expected to be used for end-to-end encryption of all communications with the server. This will ensure that eavesdroppers will not be able to make sense of the communication made by the application. Qualys SSL Labs[6] will give us a better picture of the network security implementation and also a grade on how good the security is. Furthermore, tampering with the APK itself should be detected in the application, and it should not be usable anymore. This can be easily tested with the apk-mitm[7] tool which rebuilds the APK in a way that it removes CA authority and allows faulty signed certificates which in turn lets MITM attacks to be carried out.

It is also expected that there are no leaks in the codebase for any keys or other endpoints that might give attackers extra information about the whole IT-system of LHV. This is important because the attackers might get access to cloud databases or other services this way. Decompiling and scanning the code can give us a better overview of this.

The authentication mechanisms of the application are expected to be of very good quality which will ensure that only authenticated users can have access to their information. This is due to the obvious privacy and theft factors discussed in the Threat Model section. Authentication can be done in many ways, especially since there are various authentication providers in Estonia. The current best practice is to use at least two authentication layers to increase the difficulty of malicious use by attackers. Moreover, it is to be expected that transactions should have an authentication mechanism in place as it adds another layer of protection. This is important because if a bad actor gets access to their account then the amount of damage they can do can be limited only to information gathering.

The main ways to authenticate the user are the following:

- Something you know (eg. a password/pin)
- Something you have (eg. a physical device displaying a code/readable ID card)
- Something you are (eg. fingerprint or face scan)

Finally, the mobile application should not spread sensitive information to third-parties that don't need it. Therefore, it is expected to have a small amount of third-party integrations, and they are expected not to carry sensitive information like account balance and similar.

Android APK was installed from APKMirror[8] with the version 3.54.14 of the application.

4.1 Software security

To conduct the software analysis Apktool[9] was used in order to decompile the application and then Android Studio to analyze the code base. As it turns out, the application is made in Kotlin[10], a popular language built on JVM[11].

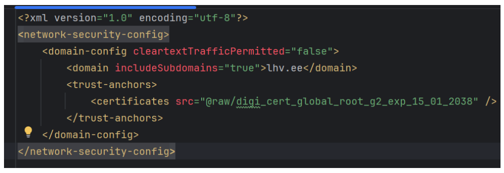
After decompiling the application, it was observed that the whole application had been obfuscated and was a hard read. The code logic was mostly in .smail files. The only things that were readable were some hardcoded strings, such as exceptions and constant variables. There is also a lot of asset files so that the application does not have to ask for those in the server.

I ran a check on the code base with the following words:

- Password
- Token
- Bearer
- Key
- Hash
- JWT

It is usually an Estonian standard for applications to use JWT from my experience and thus it could be assumed there are files related to JWT token handling. As it turns out, there were files related to JWT token handling. From the files, it can be observed that some version of firebase library is used which means that Firebase Authentication[12] service is used for token management. Later on, in the MobSF [13] scan we will discover that there are in fact places where this starts to play a bigger role.

Even though the code was unreadable, the libraries used can be observed in the folder named "unknown" where it was observed that OkHttp [14] is most likely used for TLS and certificate pinning because there was a folder named `okhttp3` where a configuration reference was held. I also discovered a file named `network_security_config.xml`.



```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">lhv.ee</domain>
        <trust-anchors>
            <certificates src="@raw/digi_cert_global_root_g2_exp_15_01_2038" />
        </trust-anchors>
    </domain-config>
</network-security-config>
```

Figure 1: Network security configuration file

From this file it can be deducted that the configure enables:

- Certificate Pinning is implemented and lhv.ee is specified for a trusted certificate, also **digi_cert_global_root_g2_exp_15_01_2038** confirms that OkHttp is used for certificate pinning.
- It's also cleared from the part **cleartextTrafficPermitted="false"** that the application prohibits unencrypted HTTP communication with the lhv.ee domian, enforcing HTTPS connections only.
- **Custom Trust Anchor**, rather than relying solely on the system's certificate store, the application defines a specific trusted certificate authority, which enhances protection against fraudulent certificates.

This configuration gives a fair protection against man-in-the-middle attacks, which is one of the main concerns mentioned in the treat model. By implementing certificate pinning, the application rejects connections of an attacker attempting to intercept traffic using a different certificate, even if the certificate is signed by a trusted authority.

It should also be noted that if the application here communicates with other domains than lhv.ee then the same security guarantees do not hold. This will be investigated also in the networking section, but in advance, all the requests are made to lhv.ee domain and no other request destination was found.

4.1.1 MobSF scan

Finally, since the code is heavily obfuscated and detailed analysis could not be done then MobSF [13] framework was used to conduct static analysis on the APK file. Worryingly, the application received security score of 58 out of 100 and a risk rating of B, that being medium risk. The tool also was able to convert files to Java code where the readability is better but most of the code is still obfuscated.

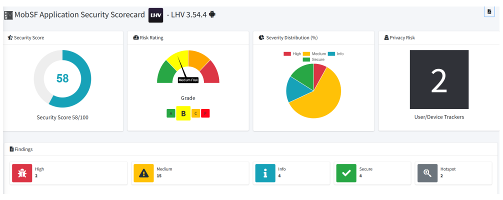
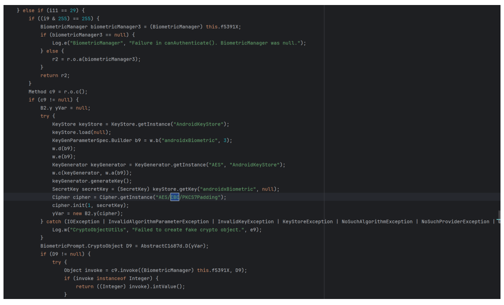


Figure 2: MobSF score

Two high severity vulnerabilities were found:

- "Application is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues." - This is used for signing the application by the Head of Digital Banking in LHV, Margus Holland. This is most likely some release/development cycle related approval with Estonian ID certificate.
- "The App uses the encryption mode CBC with PKCS5/PKCS7 padding. This configuration is vulnerable to padding oracle attacks." - It also referenced a file called R3/a.java which was obfuscated and I could not tell if it was a package built by LHV or some dependency that they are using but from the code it was deductible that the encryption is CBC with PKCS7 which is vulnerable to Oracle Padding attacks [15]. The problematic encryption is used for biometric login. This violates confidentiality and integrity from threat model by enabling potential decryption of biometric tokens through padding oracle attacks.



```
} else if (i11 == 29) {
    if ((i9 & 255) == 255) {
        BiometricManager biometricManager3 = (BiometricManager) this.f5391X;
        if (biometricManager3 == null) {
            Log.e("BiometricManager", "Failure in canAuthenticate(). BiometricManager was null.");
        } else {
            r2 = r.o.a(biometricManager3);
        }
        return r2;
    }
    Method c9 = r.o.c();
    if (c9 != null) {
        B2.y yVar = null;
        try {
            KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
            keyStore.load(null);
            KeyGenParameterSpec.Builder b9 = w.b("androidxBiometric", 3);
            w.d(b9);
            w.e(b9);
            KeyGenerator keyGenerator = KeyGenerator.getInstance("AES", "AndroidKeyStore");
            w.c(keyGenerator, w.a(b9));
            keyGenerator.generateKey();
            SecretKey secretKey = (SecretKey) keyStore.getKey("androidxBiometric", null);
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
            cipher.init(1, secretKey);
            yVar = new B2.y(cipher);
        } catch (IOException | InvalidAlgorithmParameterException | InvalidKeyException | KeyStoreException | NoSuchAlgorithmException | NoSuchProviderException |
            Log.w("CryptoObjectUtils", "Failed to create fake crypto object.", e9);
        }
        BiometricPrompt.CryptoObject D9 = AbstractC1687d.D(yVar);
        if (D9 != null) {
            try {
                Object invoke = c9.invoke((BiometricManager) this.f5391X, D9);
                if (invoke instanceof Integer) {
                    return ((Integer) invoke).intValue();
                }
```

Figure 3: CBC with PKCS7

While scanning the medium security risks made by MobSF the analysis revealed that there were some hard-coded endpoints, keys and tokens left in the code in plaintext which I did not detect while using Apktool. The following endpoints and keys were detected in the code base after further inspection.

settings.xml:

- `com.google.firebase.crashlytics.mapping_file_id = b9bbfd74228f4a03b3bdba631f333612`
- `firebase_database_url = https://lhv-mobile-001.firebaseio.com`
- `google_api_key = AIzaSyAZ8UCSah90aJ-Q5-q2ZsJS7skoC3NYjgY`
- `google_api_id = 1:987892835981:android:7b132db9441eaf80`
- `google_crash_reporting_api_key = AIzaSyAZ8UCSah90aJ-Q5-q2ZsJS7skoC3NYjgY`
- `google_storage_bucket = lhv-mobile-001.appspot.com`

DefaultSettingsSpiCall.smail:

- `X-CRASHLYTICS-DEVELOPER-TOKEN = 470fa2b4ae81cd56ecbcda9735803434cec591fa`

Config.smail:

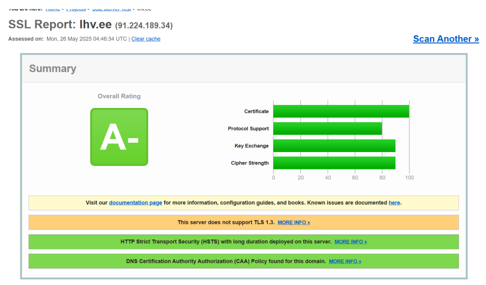
- `GOOGLE_DIRECTIONS_API__BASE = https://maps.googleapis.com/maps/api/`
- `GOOGLE_DIRECTIONS_API_KEY = QU16YVN5QUt2TGVtZkFSaE1WTE5xNWZva2V3bHVTaTJyazJsV`

This is a bad discovery since not only are some keys in the text but also now we know the exact endpoints for the services that LHV uses like Firebase database url or their storage bucket url. While I got permission denied for all of these endpoints, which is good, it still is a clear mismanagement of how to store sensitive information. Furthermore, since none of these endpoints are directly used in the application to send requests then they should not be there at all since all of the requests go through lhv.ee domain. This compromises confidentiality by revealing cloud service endpoints (lhv-mobile-001.firebaseio.com), directly enabling for example data miners to target backend infrastructure.

4.2 Network Security

4.2.1 Qualys SSL Labs

To start off with, I ran an Qualys SSL Labs analysis on lhv.ee domain to see if there are any clear security flaws that can be detected.



You are here: Home > Projects > SSL Server Test > lhv.ee

SSL Report: lhv.ee (91.224.189.34)

Assessed on: Mon, 26 May 2025 04:46:34 UTC | Clear cache

Scan Another »

Summary

Overall Rating

A-

Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server does not support TLS 1.3. MORE INFO »

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. MORE INFO »

DNS Certification Authority Authorization (CAA) Policy found for this domain. MORE INFO »

Figure 4: Qualys SSL Labs

As it can be seen, the scan gave a grade of A- which means "Acceptable security, some significant issues" [6]. While most of what SSL report showed was positive there is one exception to this. This being that the server does not support TLS 1.3 protocol and uses the TLS 1.2.

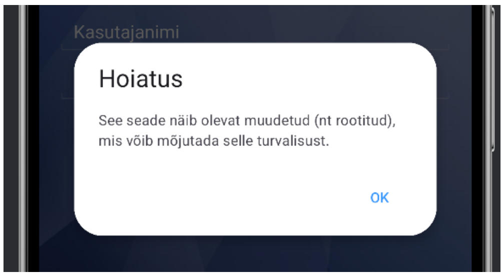
While the server supports strong cipher suites like TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, the lack of TLS 1.3 support means it cannot benefit from the protocol's improved security features, including forward secrecy improvements, simplified cipher suite selection, and removal of legacy features [16]. For a banking application, achieving an A+ rating with TLS 1.3 support should be the minimum standard.

4.2.2 Man-in-the-middle attack

In this section, Burp Suite [17] was used for network monitoring, a certificate from "http://burpsuite" and a proxy to try to intercept traffic coming from the application. I used Android Studios Android Emulator running Android 16.0 to run the application in a local environment to make it easier to run a patched APK file. Apk-mitm [7] tool was used to patch the APK so that the security configuration would be removed.

Before I move on to patching the application, I also tried to man-in-the-middle attack the app without using apk-mitm, only with Burp Suite, and pinning was in place so I could not intercept any of the network requests inside the application.

After applying the patch and running the faulty APK, on opening the application I got a warning that the device has been rooted and it might affect the security of the application in Estonian.



Kasutajanimi

Hoiatus

See seade näib olevat muudetud (nt rootitud), mis võib mõjutada selle turvalisust.

OK

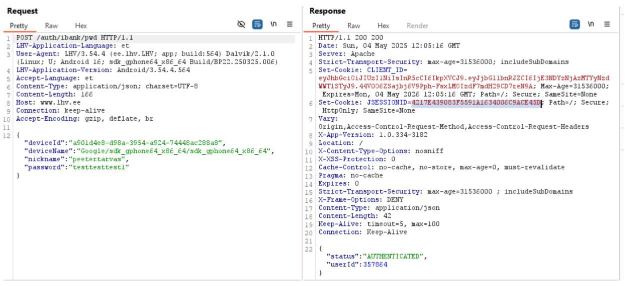
Figure 5: Warning

This was not the expected result, as ideally the application should have either blocked further use or at least displayed a warning indicating that it had been tampered with. The fact that it continued to function suggests weaknesses in runtime integrity checks. The success of the man-in-the-middle, hereafter MITM, attack [18], despite the app's original use of TLS and certificate pinning, was made possible by modifying the APK using the apk-mitm tool. This tool removes the certificate pinning logic from the app's network configuration, thereby disabling the app's verification of server certificates. Under normal circumstances, certificate pinning ensures that the app only trusts a specific certificate or public key, rejecting even valid CA-signed certificates if they don't match. By stripping out these checks, the patched version of the app accepted the Burp Suite self-signed root certificate, allowing interception of HTTPS traffic. This indicates that while the original security state was efficient, the lack of anti-tampering mechanisms allowed the app to operate in a compromised state, making it vulnerable to advanced attackers who can manipulate the APK.

As this is the case, I could continue with my MITM attack. I decided that since the password login is most likely to be vulnerable then I would target that. The biometric login option is not testable on the emulator so this is why I didn't try to attack that method as well.

I adjusted the proxy settings on both my phone and the Burp Suite proxy using this [19] guide.

While logging in I got a "/auth/ibank/pwd" log in my http history where my account details can be seen.



```
POST /auth/ibank/pwd HTTP/1.1
LHV-Application-Language: et
User-Agent: LHV/3.54.4 (ee.lhv.LHV; app; build:564) Dalvik/2.1.0
(Linux; U; Android 16; sdk_gphone64_x86_64 Build/BP22.250325.006)
LHV-Application-Version: Android/3.54.4.564
Accept-Language: et
Content-Type: application/json; charset=UTF-8
Content-Length: 166
Host: www.lhv.ee
Connection: keep-alive
Accept-Encoding: gzip, deflate, br

{
  "deviceId":"a901d4e8-d98a-3954-a924-74448ac288a8",
  "deviceName":"Google/sdk_gphone64_x86_64/sdk_gphone64_x86_64",
  "nickname":"peetertarvas",
  "password":"testtesttest1"
}
```

```
HTTP/1.1 200 200
Date: Sun, 04 May 2025 12:05:16 GMT
Server: Apache
Strict-Transport-Security: max-age=31536000; includeSubDomains
Set-Cookie: CLIENT_ID=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjE3NDYzNjAzMTYyNzd
WWT1STyJ9.44V006ZSajbj6V9Pph-FsxLM0IzdF7mdH29CD7reN9A; Max-Age=31536000;
 Expires=Mon, 04 May 2026 12:05:16 GMT; Path=/; Secure; SameSite=None
Set-Cookie: JSESSIONID=4217E439083F5591A1634006C9ACE45D; Path=/; Secure;
 HttpOnly; SameSite=None
Vary:
Origin,Access-Control-Request-Method,Access-Control-Request-Headers
X-App-Version: 1.0.334-3182
Location: /
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
Content-Type: application/json
Content-Length: 42
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive

{
  "status":"AUTHENTICATED",
  "userId":357864
}
```

Figure 6: Login request

This proves that MITM attacks are still possible with some difficulty. I also received an email notification that a new device has been used to log in to my account, which is positive because the user can now report that their account might have been hijacked. It also means that the application is gathering user data on what devices they are using to add awareness for the user about their accounts activities.

An interesting thing is that once the user has logged in, a lot of network requests are made right after. I assume that this is because the application wants to cache some data so it would run faster when switching from one view to another.

Endpoints like:

- "/auth/ibank/users" - provides userId(possibly database id), and privileges that the user has
- "/services/external/mobile/1.7/portfolio/get_user_data" - returns portfolio_id, iban numbers, amount of money the user has on all in their bank accounts and also theri card details. Note here though that the credit card number is actually delivered like so: "number": "**** **** **** 6079", so some anti intrusion policy has been used here.
- "/b/investment/balances" can be used to get info on the victims investments.
- "/b/customers/private" can be access to get person information on the user such as their ID number, birth country, currently registered living address, email, phone number, occupation, employer, income range, gender and if they have a dual citizenship or not.

This shows that sensitive information, with some difficulty, can be eavesdropped on. It is also the case that a lot of this information is not needed in the mobile application since the database id, and most of the information that can be gotten with "/b/customers/private" is not visibly used anywhere. My suspicion is that the same endpoint is used in the web application where this information can be edited but since this is not the case with the mobile app, then data-transfer objects should not include this information and separate information sending methods should be implemented so that the principle of the user only getting the data that they needed would hold.

There is a second layer of protection when it comes to making transactions and also viewing transactions made in that you cannot view or make them when logged in with a password and username. Transactions require a second confirmation as well which are handled through different login methods like SmartId [20], MobiilID [21] or RIA DigiDoc [22] - all interlinked with Estonian ID card and Estonian ID certificates [23], hereafter eID. This goes beyond the scope of this analysis since it could be a separate paper on its own.

Furthermore, all of the requests are made against lhv.ee domain which indicates that no outside requests are made. This assures us that the security configuration file from the Security Analysis section is set up correctly.

4.3 Authentication

When opening the application, the main ways to authenticate the user are the following:

- Mobiil-ID
- Smart-ID
- Pin calulator
- Biometric login
- Username and Password

4.3.1 Mobiil-ID and Smart-ID

I will discuss Mobiil-ID and Smart-ID in the same section since their workflows are similar and are based on the logic.

Mobiil-ID, hereafter mID, was and still is a government-provided authentication system which is based on the original ID card based security functionality used in Estonia provided by RIA DigiDoc. The functionality has to be ordered by the user via a mobile operator like Telia [24] or Elisa [25] where they will provide you with a specialized mobile SIM card that holds your digital signature electronic id that supports mID login. When applying for mID, a proof of identity is needed such as an ID card or Passport. The user will be given two pins, one for sign-in actions and the other for transaction related actions which they can use to login.

Login workflow:

- User chooses to login via mID. The user is asked to enter their phone number and a personal code.
- After entering the code, user will see a control/verification code appearing in the LHV bank app.
- After a moment the user will see the same code appearing on the mobile phone screen. The user confirms if the code on their mobile is the same as on the application.
- The user enters their PIN1 code.
- The user is authenticated.

Smart-ID, hereafter sID, is an alternative to mID, the main difference being that with sID does not need a special SIM card to work. Instead of this, there is a separate mobile app, SmartID, which will verify your identity. The registration for sID can be done physically at a bank or online at sID's website, in both instances id card or passport verification is required. Once the account has been made, then the user has to register their device so that the application would work. The requests made by the sID application are device based. The user will generate two keys just like in mId, but this time whenever they login they will be redirected to the sID app.

Login workflow:

- Choose Smart-ID authentication and enter username and personal ID code.
- Verification code is shown.

- User is redirected to sID application
- User selects the verification code from three other verification codes
- User enters PIN1
- The user is authenticated.

4.3.2 Pin calculator

Pin calculator can be requested from the bank for both login and transaction authentication. A proof of identification has to be shown when request is made. This can be either ID card or a passport [26].

Login workflow:

- User unlocks their PIN device.
- User has to enter a 5 digit PIN-code to unlock the device
- "APPLI-" is promted on the screen and when user presses the key 1 then an 8 digit code will be displayed that the user can insert in the mobile application.

4.3.3 Biometric login

Biometric login with a fingerprint needs to be enabled in the settings. It also requires a second layer email/-phone number code confirmation to be done.

Biometric login on the phone uses fingerprint as an authentication mechanism. The users also has full rights with this login, meaning that transactions are visible and they can make payments. For the transactions, they still have to provide sID PIN2 sometimes but usually the transaction can just be confirmed by fingerprint scan as well.

It is also apparent that this might have some sort of security vulnerability considering that in the Software Security section it was discovered that there is an encryption flaw in that the application uses CBC with PKCS7 padding to encrypt this data. While I could not use the vulnerability myself, a more sophisticated actor could potentially break this encryption and use it for their benefit.

4.3.4 Username and Password

The user also has an option to to login with a password. The password can be only set in either at the 1) bank provided the customer has an valid ID and also access to the ID card PINs. 2) Online web portal where also login with ID card, mID, sID or PIN calulator is already needed. There are two types of passwords: one for login named "Sisselogimisparool" and another for making time critical trades called "Kauplemisparool". Both passwords have requirements that they have to consist of at least 12 characters, and include both numerics and letters [27].

The passwords here are the most insecure option since they passwords can be reused and leaked in a different application database leakage. This threat seems to be taken into account in that the password login has less authorized actions than the other login methods. Password login does not support making transactions and viewing previously made transactions. However, you can still view users investments, credit cards and the total amount of capital they have.

It should also be mentioned that passwords only have a half-year lifetime after which the user needs to generate a new password to continue using password login functionality and also after three consecutively failed login attempts the password is blocked for a unspecified amount of time.

Login workflow:

- User selects the password option and enters their credentials.
- User is logged in.

Here, even though passwords already have some limitations set, the login can be improved on. For one, the Multi-Factor Authentication [28] could be used in order for a login confirmation. Also, the password length could be larger for more security since most up-to-date advice is that passwords should be at least 14 characters long [29].

4.4 Privacy

In this section, I describe the different privacy properties of the LHV app. I will cover the private data that the app handles, the services it integrates with and how they affect privacy, and the privacy trackers and required permissions of the app.

As also explained in the Threat Model section, banks by nature require a lot of personal information such as id codes, living addresses, birthdays, phone numbers, emails, personal incomes, occupations and employers. They also have a lot of data, mainly through transaction history, that can be used to make reasonable guesses such as including habits, social circle or daily routines. This should be disclosed in the privacy policy of the bank.

As discovered in the Networking section, the application also communicates a lot of this information through the API but does not really use them in a meaningful way in the mobile application itself.

By reading LHV personal policy [30] it is stated that they collect a lot of personal data, which includes:

- Personal details – name, preferred name, ID verification details (passport or other ID document, photograph, video), and residential status.

- Contact details – correspondence and billing address, telephone number, email address, online messaging details, and social media details.
- Financial details – employment status, annual income, residential status, and PEP status.
- Demographic information – gender, date of birth / age, nationality, salutation, title, and language preferences.
- Customer service logs – details of conversations with us through email, website or app.
- Consent records – records of consents you've given, with date, time, method, and relevant context.
- Purchase details – records of purchases and prices.
- Payment information – invoice records, payment method and amount, billing address, bank or card details, payment date, Bacs, SWIFT and IBAN details, and mobile banking authentication.
- Data relating to website or app usage – device type, operating system, browser info, IP address, cookies (if consented), language settings, usage times, mobile network, behavioural biometrics, mobile advertising ID, and statistical data.
- Google Analytics and app usage – anonymised app usage data used for performance tracking, behavioural insights, and marketing optimisation.
- Message history – disclosures through in-app chat, secure messaging, and email.
- Technical data – login data, browser type and version, time zone and location, plug-ins, OS, and device technology.
- Employment status – collected when interacting with the service.
- Content and advertising data – records of interactions with ads and content, including forms completed or partially completed, mouse or touch interactions.
- Views and opinions – any personal opinions or feedback sent to or posted about the service.

The privacy policy is quite detailed as is expected from a bank and a mismanagement of this policy was also not detected.

4.4.1 Permissions

The application can use the following permissions on Android:

- Access contacts - for payment history details if possible
- Camera - if bank asks for details, such as ID it can be sent via the application
- Notifications - for sending information on the phone to the user about their bank, like money has been received.
- Location - used for a navigation system in the application which shows the offices where the bank operates and customers can go to.

4.4.2 Third party software

The application's third-party software was analyzed with Exodus Privacy [31]

Two trackers have been found in the code signature: Google CrashLytics [32] and Google Firebase Analytics [33] as was also stated in the privacy policy.

It is evident that Google CrashLytics is used for crash reporting for the phone application. Moreover, I assume that Google Firebase Analytics is also used for the reasons that is stated in the privacy section but it is not verifiable since all of the requests go though lhv.ee domain and the data being gathered can not be tracked through a network logger.

5 Discussion

While LHV implements strong authentication methods (Smart-ID, certificate pinning), several flaws that damage the security of the application were found.

The CBC-PKCS7 implementation in biometric authentication creates a risk of a potential padding oracle attack. While not testable in the scope of this report, it is clear that this could be a potential weakness for more sophisticated actors who are willing to go the extra mile. This would make the authenticity of the application potentially vulnerable.

Exposed third-party endpoints and keys violate OWASP MASVS-STORAGE [34] guidelines, potentially enabling movement and mapping of LHV's cloud infrastructure. This potentially could impact the confidentiality of the whole LHV IT infrastructure.

The TLS 1.2 implementation, while currently secure, could be improved by upgrading to TLS 1.3.

The detection of tampering could be improved. A simple warning would be a good start, indicating that the application has been tampered. An even better way of resolving this would be to block the application from usage after it has been patched or edited by another party. Man-in-the-middle attacks are possible due to the fact that the tampered app can still be used. Granted, this requires that the attacker can manipulate the victim to install a patched application and also their certificates. The tampering and MITM attacks violates integrity, confidentiality and if password login is used then also authenticity of the application.

The network requests, as it turns out, give the mobile application too much information. Most of the information sent over the requests is not used for anything visible or informative. This could be improved by making more data limiting data-transfer objects for the mobile application.

6 Conclusion

LHV banking application was analyzed in this project. Threat model was made and an analysis was conducted where the application was deemed to be of high value to bad actors requiring it to have a good security standards implemented. Some flaws during the security analysis were found, such as exposed endpoints and keys in the code base, potentially flawed security in the biometric token, use of TLS1.2 not 1.3, sending too much information and patching the application.

References

[1] . Lhv android application.

[2] LHV Pank. Lhv.

[3] . Google play store.

[4] SAS Institute. Big data: What it is and why it matters, 2024.

[5] CBS Research. The danske bank money laundering scandal: A case study. *Copenhagen Business School Research*, 2019.

[6] Qualys, Inc. Key changes in ssl labs grading and qualys certview, February 2025. Accessed: 2025.

[7] shroudedcode. apk-mitm: A cli tool for automatic interception of android apk files, 2024.

[8] APKMirror. Apkmirror - free apk downloads, 2024.

[9] Connor Tumbleson. Apktool - a tool for reverse engineering android apk files, 2024.

[10] JetBrains. Kotlin programming language, 2024.

[11] Oracle Corporation. Java virtual machine (jvm), 2024.

[12] Google. Firebase authentication, 2024.

[13] Ajin Abraham. Mobile security framework (mobsf), 2024.

[14] Square, Inc. Okhttp - an http client for android and java applications, 2024.

[15] Arnold K. L. Yau, Kenneth G. Paterson, and Chris J. Mitchell. Padding oracle attacks on cbc-mode encryption with secret and random ivs. *Fast Software Encryption*, pages 299–319, 2005.

[16] Jaee Moharir. Key changes in ssl labs grading and qualys certview, 2025.

[17] PortSwigger. Burp suite professional, 2024.

[18] Imperva. Man in the middle (mitm) attack, 2025.

[19] Bhuvan Gandhi. Capture network traffic of mobile device in pc using burp suite, 2024.

[20] SK ID Solutions AS. Smart-id official website, 2024.

[21] Signicat. About mobiil-id, 2024.

[22] RIA. Ria digidoc application.

[23] Republic of Estonia Information System Authority. Electronic identity (eid) and trust services, 2024.

[24] Telia Company. Telia estonia, 2024.

[25] Elisa Corporation. Elisa estonia, 2024.

[26] LHV Pank. Pin-kalkulaatori kasutusjuhend, 2024.

[27] LHV Pank. Kkk - internetipank: Paroolid, 2024.

[28] NIST. Multi-factor authentication (mfa), 2024.

[29] Bitwarden. How long should my password be?, 2024.

[30] LHV Pank. Lhv privacy policy, 2024.

[31] Exodus Privacy. Exodus privacy - android application privacy audit platform, 2024.

[32] Google. Google firebase crashlytics.

[33] Google. Google firebase analytics.

[34] OWASP Foundation. Owasp mobile application security verification standard (masvs).