Technische Universität München

Department of Informatics

Bachelor's Thesis in Information Systems

A Digital Wallet Implementation for Anonymous Cash

Oliver R. Broome

Technische Universität München

Department of Informatics

Bachelor's Thesis in Information Systems

A Digital Wallet Implementation for Anonymous Cash

Implementierung eines digitalen Wallets for anonyme Währungen

Author	Oliver R. Broome
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Sree Harsha Totakura, M. Sc.
Date	October 15, 2015

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, October 15, 2015

Signature

Abstract

GNU Taler is a novel approach to digital payments with which payments are performed with cryptographically generated representations of actual currencies. The main goal of GNU Taler is to allow taxable anonymous payments to non-anonymous merchants.

This thesis documents the implementation of the Android version of the GNU Taler *wallet*, which allows users to create new Taler-based funds and perform payments with them.

Zusammenfassung

GNU Taler ist ein neuartiger Ansatz für digitales Bezahlen, bei dem Zahlungen mit kryptographischen Repräsentationen von echten Währungen getätigt werden. Das Hauptziel von GNU Taler ist es, versteuerbare, anonyme Zahlungen an nicht-anonyme Händler zu ermöglichen.

Diese Arbeit dokumentiert die Implementation der Android-Version des Taler-Portemonnaies, der es Benutzern erlaubt, neues Taler-Guthaben zu erzeugen und mit ihnen Zahlungen zu tätigen.

Contents

Chapter 1

Introduction

Today's payment market is in a state of upheaval - there is a discernible trend towards digital payment in commerce; this can be witnessed in several areas of the payment sector: Google [1], Apple [2] and Samsung [3] have all launched their own proprietary mobile payment systems coupled to their respective platforms; eBay and their now-independent subsidiary PayPal have been involved for several years already [4]; and German banks scramble to introduce their own digital payment system [5].

Bitcoin [6] and its various derivatives have also established themselves as alternatives to traditional payment systems and are gaining support in sectors that would have dismissed cryptocurrencies only a few years earlier [7].

However, a major drawback of the aforementioned proprietary payment systems is that they trade off anonymity for convenience. Users must have user accounts or other means of identifying themselves to third parties, usually the payment service provider, the merchant and any other parties the first two have data sharing deals with. Payments are performed via non-anonymous channels and are traceable with little to no effort.

Even if the design of the payment system itself provides a certain degree of safety, such as Bitcoin (or the more privacy-minded derivatives, such as Zerocoin [8]), there are other drawbacks to be considered. To continue the example of Bitcoin, blockchain-based cryptocurrencies that keep a centralised record of all transactions made over the network are already starting to show signs of not being able to scale well with large numbers of users simultaneously making transactions or the high amount of processing power required to "mine" new Bitcoins. Additionally, the amount of energy invested into this mining process is increasing at similar rates to the increased demand for computing power.

Cryptocurrencies also suffer from an uncertain legal standing. Unlike traditional payment methods, they are often seen as commodities instead of currencies, for example, enabling law enforcement to auction them off after seizing them. The most prominent case of this was in the aftermath of the Silk Road shutdown [9].

GNU Taler [10] seeks to make an improvement on several of the aforementioned shortcomings. Based on Chaum-style blind signatures [11], it does not require vast computational resources to operate its infrastructure and has privacy by design. Given enough users, payments can be made in a manner that does not enable merchants to link payments to people (when using the GNU Taler system by itself) and outside observers to determine the parties involved in a transaction.

The GNU Taler system is, in contrast to cryptocurrencies, very forthcoming towards states seeking to implement it: payments are taxable and income generated through it can be linked to the merchant receiving the funds.

In order to enable users the use of GNU Taler in an ubiquitous fashion and for GNU Taler to gain adoption, clients for popular mobile electronic devices are necessary. It is out of this need that the Android wallet and the accompanying thesis have been written.

1.1 GNU Taler

The GNU Taler payment system functions through the interactions between three parties: the *customer*, the *mint* and the *merchant*.

The customer is the party willing to make a payment in exchange for a service or product from a merchant. Payment is performed by transferring *deposit permissions* for *coins* they have created earlier.

These coins are created from *reserves* that the customer creates at a Taler *mint*. In order to create a reserve, the customer transfers money to the mint via a bank transfer.

Once the mint has registered the arrival of the funds intended for the customer's reserve, the customer can request coins of different values and currencies from the mint.

With these coins, the customer can pay the merchant by creating a deposit permission from the contract the merchant sends them. This permission is then sent to the mint. If the mint can validate the permission's correctness, it sends the equivalent value of the deposited coins to the merchant's bank account.

These processes are secured through the use of cryptography; for a full explanation of its use in the GNU Taler payment system, refer to the paper by Dold, et al. [10].

1.2 Goals of the thesis

To enable its use on the currently most popular mobile platform - Android - this thesis presents a functioning wallet for the GNU Taler payment system. This implies the ability of the application to create *reserves* at a *mint*, withdraw coins from a reserve and generate *deposit permissions* for *merchants* so that they can, in turn, request the funds required for the fulfilment of the payment contract from the mint.

To enable the use of GNU Taler on Android, it was necessary to bring the cryptographic library *libgcrypt*, the *GNUnet* utility library and the GNU Taler utility library into a state that enables them to be used from inside of the Android operating system's ecosystem. This involves cross-compilation of the existing code to different architectures (mainly ARMv7, but also x86 and its 64-bit variant) and creating a Java-to-native interface for the libraries in addition to the implementation of core functionality.

Thus, an additional, yet essential goal was to implement the necessary prerequisites for the use of GNU Taler functionality on platforms other than the standard Linux-on-x86.

1.3 Outline

The thesis is structured as follows:

In chapter 2 the existing code base and the necessary steps taken to adapt them to the Android platform will be examined. An overview of the application architecture will be given in chapter 3 and the offered functionality shown in chapter 4. An overview of potential security risks and their mitigations will be given in chapter 5. Finally, the thesis is concluded in chapter 6, where the findings of this thesis will be summarised and an outlook to the future of the GNU Taler wallet will be given.

Chapter 2

Implementation prerequisites

The Android Taler wallet reuses several frameworks for its implementation of the cryptographic primitives and their applications within the Taler payment system.

The three main components are Libgcrypt for the primitives and the GNUnet and Taler utility libraries for the implementation of the necessary data formats and cryptographic routines.

By using these libraries, a significant amount of code is shared between the Android wallet and existing implementations, thus preventing, in the long term, maintenance issues and redundancy, which would be introduced by a reimplementation of the respective libraries.

2.1 Native libraries

2.1.1 Libgcrypt

GNUnet and Taler both rely on libgcrypt [12] for its implementation of cryptographic primitives. In particular, the two former libraries make use of RSA for blind signing, EdDSA signatures for other signing purposes and a combination of ECDH, AES and Twofish for symmetric encryption. Hashing is performed via SHA-512 and key derivation functionality relies on SHA-256.

While there are comparable cryptography libraries that are more accessible to Android, such as Spongy Castle [13], or the Android-provided [1]implementation of *java.security*, Bouncy Castle [14]; using libgcrypt reduces the overhead necessary for transferring data from the Dalvik VM (which is the Android project's implementation of a Java VM) back to the native components. Additionally, GNUnet and Taler are tightly coupled to libgcrypt due to their use of its S-Expressions, which are used in several critical locations, such as the de- and encoding of RSA keys [15].

Another aspect that tilted the decision in favour of using an external library was the fact that the Android platform has a reputation for a lack of updates, even on fairly new devices. Devices usually receive updates for 2 years at most, rendering devices that remain on older Android API Levels[2] inherently vulnerable to exploitation. The recently discovered flaw [17] in the random number generation of Bouncy Castle that affected the Android API levels up to 19 is a good example of this phenomenon [18]. Devices that have reached the end of their commercial shelf life and no longer receive updates from their manufacturer are left with a broken randomiser.

The main drawback of using libgcrypt on Android, however, is that it is not the main focus of libgcrypt's development. While the library will compile under most conditions, there are certain aspects that can hinder its compilation and usage; these will be described in Section 2.2.2. Also, libgcrypt is not widely used on Android, possibly because of the added difficulty in the use of native code on the platform.

Due to the efforts of the Guardian Project, an existing, albeit somewhat outdated build script [19] could be adapted for use under the application's development environment, which enables the execution of tests for the libgcrypt code in an Android environment.

2.1.2 GNU unistring

In order to be able to compile GNUnet for Taler wallets, it is necessary to compile *GNU unistring*. This proved to be a straightforward matter, as it was possible to cross-compile the library without any issues.

GNU unistring is a comparatively large code dependency, requiring roughly 3 MiB of storage alone. Unfortunately, the library is not an optional dependency and must, therefore be included in the wallet project files. It may be possible to reduce the size of the compiled library by removing functionality from the library, such as support for unused feature sets of Unicode.

2.1.3 GNUnet utility library

The GNUnet project is a modular approach to enhancing communications security and has, in order to provide that security, implemented many convenience functions that wrap around libgcrypt. The Taler utility library is tightly coupled with the GNUnet library, making it essential for the operation of the wallet. In particular, the Taler library

[1]This is no longer entirely true, Google are currently migrating to their fork of OpenSSL, BoringSSL [16]

[2]The numbers used in conjunction with API Levels represent the revision number of the API. API Level 21 is, for example, more commonly known as Lollipop.

relies on the data structures and the cryptographic routines for SHA-512 hashing, RSA, RSA blind signing, EdDSA signatures, and ECDH.

Due to its modular nature, however, the effort involved in order to make the wallet-only functionality cross-compile to Android was not an issue and swiftly performed by the maintainers. There were some initial problems that prevented cross-compilation, such as checks in the build phase of the project, where code was compiled and executed before the actual library was created. This test code was, due to the nature of the build scripts, already compiled with the Android compilers, and would, therefore, not run on machines with different configurations and processor architectures. A patch that removes these checks from the configuration scripts is present in the source code repository of the Android wallet.

2.1.4 Taler utility library

The Taler utility library, taken from what is currently maintained as the Taler mint [20], provides us with additional cryptographic functions related to the various tasks that arise from the procedures dictated by the Taler protocol. It has also been modified by the maintainers to include only the codebase necessary for the wallet itself.

Also specified by this library are the various data structures that are used when communicating with a Taler mint.

2.1.5 JNI dispatch library / Java Native Access

In the course of the implementation of the Android wallet application, a decision was made to use the Java Native Access framework [21] to delegate calls to native functionality from the parts of the application written in Java.

In order to enable the use the JNA, an additional component was necessary: the JNI dispatch library or *libjnidispatch*. This library allows the Java VM executing the application code to use a minimalistic JNI wrapper around *libffi* (the Foreign Function Interface library) to call native-only code without the need for glue code around every single native library and function (more on this subject in Section 2.2.3).

2.2 Android integration

Porting the functionality of the existing implementation of a Taler wallet in Python can be broken down into several distinct fields, as will be described in the following.

2.2.1 Data Persistence

The currently existing Python- and JavaScript-based wallets use an SQLite database for persistence. On Android, applications can store their data in the same format, making the adaptation of the existing table structures fairly straightforward. This also allows for the sharing of the database schema across platforms, which in turn is useful for possible later changes and allows users to migrate from one wallet client to another with a comparatively small amount of effort.

The Android framework allows its applications to access its own databases in two ways, either directly via a *SQLiteDatabaseHelper* or indirectly via a *ContentProvider*. In order to keep the scope of the project within reasonable boundaries, the current implementation forgoes the Content Provider approach for the simplicity of the database helper.

2.2.2 Cross-compilation

Build environment

In order to be able to use C code with the desired processor architecture and API Level combinations, the code must be compiled with the appropriate tooling. Android offers the *NDK* or *Native Development Kit* for this task. It gives developers many different toolchain combinations covering various host architectures, platforms and targets.

Development of the wallet so far has focused on ARMv7, due to its broad market share, but the NDK allows for an easy expansion to x86 and MIPS, if the need arises. Most non-ARM devices feature a compatibility layer that allows them to emulate the ARM instruction set, which should make cross-platform use a non-issue, even if no native versions are compiled for the other platforms.

While the Android operating system is based on Linux, it does not, however, offer a complete implementation of all the regular APIs one might expect from a Linux kernel. Not all APIs of the Linux kernel are included or considered stable and are also subjected to changes across different API Levels.

In the process of cross-compiling the libraries mentioned in Section 2.1, there were several problems that could be encountered, depending on which Android API Level was targeted:

- on API Levels 7 and below, there is no implementation of regular expressions
- API Levels 8 and below lacked an implementation for the *pthread_rwlock* functions
- API Levels below 13 do not have a *pthread_atfork* implementation

The latter two examples demonstrate that the Android APIs differ in their implementation of what can be considered "basic" functionality, such as multi-threading. In practice, this means that C developers that wish to have their code ported to Android must work around these limitations.

With the given set of libraries for the wallet, the only problems so far were caused by libgcrypt and one of its tests. The *t-lock* test requires a C type by the name of *pthread_t*, which is also omitted from the API header files. Currently, the solution to this issue is to simply prevent the test from being compiled, since this is the only instance of the pthread_t type to be used within the libgcrypt source when compiling for Android. A patch for this issue has been added to the wallet source code repository.

Build automation

To actually compile the required libraries for Android, the NDK offers a framework with a set of conventions with which native code is supposed to be integrated into an Android project. For simple projects the process looks like this:

1. Add the C source files to the *jni/* folder in the project folder
2. Define which files to compile:
    - Define the library source files
    - Define which files represent JNI wrapper code
3. Automatically compile the library source files to a static library
4. Compile the static library and the compiled wrapper code into a shared library.

This procedure makes several assumptions about the code that should be compiled, most notably:

- the project uses the JNI for native code access
- the code is already adapted for use with the Android platform and the NDK's automatic building scripts

While it was possible to include pre-built shared libraries with the NDK (by simply copying them to the appropriate location in the project structure), this approach still assumes the manual creation of JNI wrappers. These are not used for the wallet; the reasoning for this is described in Section 2.2.3.

None of the libraries' sources are pre-configured for the Android platform; they all require (automated) configuration via standard GNU tools before they can be compiled. It is, theoretically, possible to introduce changes to the NDK build scripts, since they are based on those tools as well, but the resulting modifications would be subjected to future updates of the Android NDK and thus harder to maintain. Therefore, these steps must be performed manually outside of the Android build scripts.

The building issues aren't limited to differences in the building process itself, they can also be seen in the existence of different conventions, e.g. the organisation of shared libraries: a typical Linux kernel will expect shared libraries to be versioned (i.e. have their respective version numbers appended to their file name, like so: *libgcrypt.so.1.6.3*), while Android systems expect their shared objects to be unversioned. In addition to this, these versioned files aren't usually real files, but symbolic links to the actual library, which is located elsewhere. Since Android applications do not, by default, have access to locations outside of their owned directories, this scheme is not useful in an Android environment.[3] Since all included libraries originate from the GNU/Linux operating system, their build scripts do not account for this quirk and have to be manually adjusted.

These factors led to the decision that the given Android infrastructure should be completely ignored. A considerable amount of work has already been done by the Guardian Project and their customised version of the Android build scripts for libgcrypt, upon which the wallet's build scripts are based.

2.2.3 Native code interface

During the early planning phase for the wallet, the Java Native Interface was used for accessing the native parts of the application. This approach is encouraged by the Android documentation, which fails to mention that there are alternative methods to call native methods from Java [22].

The Java Native Interface is a low-level approach to foreign function interfacing, allowing access to code written in C(++) from Java objects and vice versa. It has been part of Java specifications since Java 2 [23]. This approach requires the creation of specialised wrapper code which must be integrated into the library during compilation in order to be functional.

As a consequence of this close connection between the library code and the interface code, changes to the signatures of methods, the memory layout or Java objects require re-implementation of the wrapper layer, which can be particularly cumbersome while a library is still under development. An automatic generation of the wrapper code is, therefore, more practical.

For JNI code, there exists a framework called SWIG [24], which is able to produce the necessary code by means of a custom script, which is similar to a C-language header file. It has a fairly complicated syntax and several caveats that need to be observed when compiling the libraries.

[3]This is a result of Android's security policy, which mandates that each application has its own user account assigned to it.

For the actual implementation of the wrapper, a different approach was selected. Instead of the JNI, the native code is now accessed through the *Java Native Access* framework (JNA), which dynamically accesses the shared library without the use of custom wrapper code. Instead, the libraries and their expected data structures need to be described within the Java code.

The developers of the JNA framework also offer an automatic generator for this Java-based wrapper code, called JNAerator. Its approach to code generation is similar to the way SWIG handles C code: it parses the C header files of the native library and (in theory) produces Java-based definitions ready for consumption within the application.

Although the JNAerator project can be considered mature, it still had issues with the GNUnet code.

For example, fairly simple C structure definitions such as

Listing 2.1: GNUnet, src/include/gnunet_crypto_lib.h

```c
/**
 * @brief A 512-bit hashcode
 */
struct GNUNET_HashCode
{
  uint32_t bits[512 / 8 / sizeof (uint32_t)];   /* = 16 */
};
```

will not be interpreted correctly by JNAerator due to the use of the *sizeof()* function, instead resulting in conversion errors that have to be resolved manually.

While other, more modern alternatives to the combination of JNA and JNAerator exist [25], it was ultimately decided to stay with these tools as they are the most mature of their kind, thus possibly reducing the probability of unforeseen programming errors or incompatibilities in the framework.

Chapter 3

Architecture

3.1 Application model

3.1.1 Communication

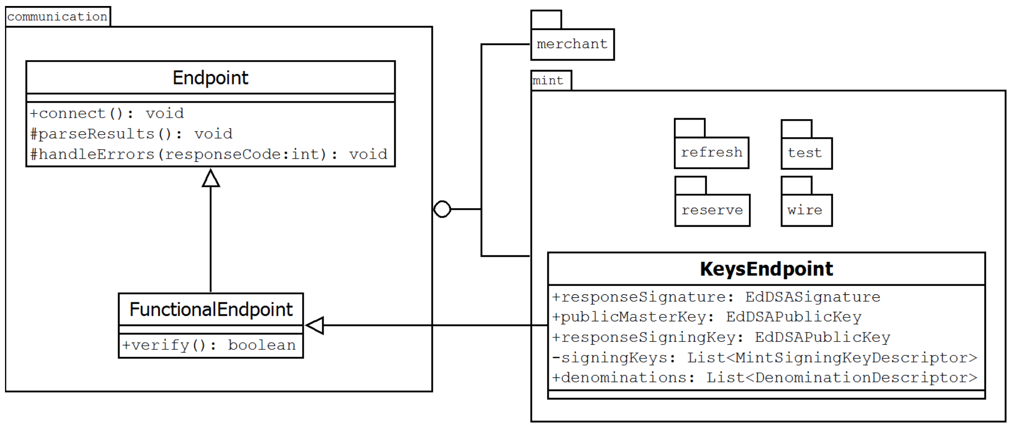

communication
Endpoint
+connect(): void
#parseResults(): void
#handleErrors(responseCode:int): void
FunctionalEndpoint
+verify(): boolean
merchant
mint
refresh
test
reserve
wire
KeysEndpoint
+responseSignature: EdDSASignature
+publicMasterKey: EdDSAPublicKey
+responseSigningKey: EdDSAPublicKey
-signingKeys: List<MintSigningKeyDescriptor>
+denominations: List<DenominationDescriptor>

Figure 3.1: Communication package

The wallet application models its communication functionality around the Taler REST APIs. [26, 27] Each endpoint is represented as its own class in its respective package: mint or merchant.

An abstract *Endpoint* class handles all communication basics such as connection establishment and management, while implementing classes take care of handling the contents of messages sent to and from the API endpoints. This encompasses serialisation and deserialisation, verification of messages and the creation of signatures over values that should be sent.

Figure 3.1.1 shows an exemplary inheritance chain for the */keys* endpoint of the mint

API. It inherits an additional abstract function, *verify()* from the *FunctionalEndpoint* class, which is responsible for determining the validity of a response from the mint. This is verified by checking that the signatures added to a response by the mint actually sign the data they are supposed to sign. The wallet can determine whether the mint is behaving dishonestly by checking these results of the signature verification, or, in some cases, offering verification results to the end user for comparison with other wallet users.

Similar classes for other endpoints are contained within the other packages shown within the *mint* package.

3.1.2 Persistence

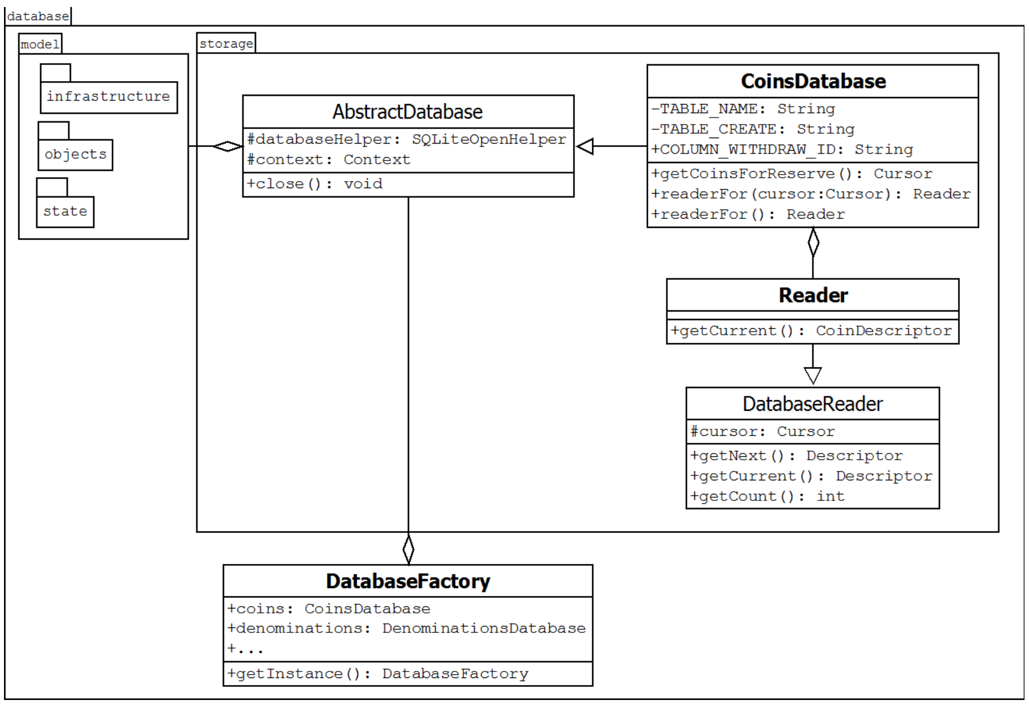


Figure 3.2: Database package

The persistence infrastructure shown above is a result of the way the Android APIs handle database access. All the classes representing the different database tables (e.g. *CoinsDatabase*) define their own SQLiteOpenHelper. It is for this reason that they are all accessed from a singleton instance of a *DatabaseFactory*. The Factory is responsible for all database installation, configuration and access tasks, supplying the class specific to a certain table of the database when needed.

If necessary, a database implementation class may choose to implement a *DatabaseReader*, which handles the conversion of rows from a database query into a Descriptor object of their choosing. These objects can be found in their respective *model* subpackages.

3.1.3 Native library wrappers

All code necessary for access to the native libraries is contained within the *nativeLibs* package. Complex data structures are stored in separate class files, while constant values, native function bindings and data structures that simply wrap pointers to other data types are contained within a monolithic library class.

When applicable, native functions that take variable-length arguments (or *varargs*) have been moved to their own correspondingly named classes, since JNA does not permit the use of statically bound native functions and variable-argument functions in the same class.

3.1.4 Android-specific code

User interface

All user-interface related code is kept in the *ui* package. This includes Android Activities and Fragments.

Tasks

(Async)Tasks [28] are a convenience feature for Android applications in which computationally expensive code can be executed without affecting the main UI thread. AsyncTasks are, as the name implies, performed asynchronously and in the order they are called.

The wallet uses these tasks for generating cryptographic keys, hashing, database access and other expensive functionality.

Services

The *services* package contains Android Services [29] that are used to provide application-wide functions that can run even when there are no visible activities belonging to the application running. Currently, this package only contains the KeyCachingService, which will be used to cache the encryption key used for securing private keys when they are stored in the application database.

3.2 Application data storage

Because Android offers built-in support for SQLite databases [30] and the original Taler wallet written in Python uses it, too, the adaptation of the existing database schema was fairly straightforward. Most of the existing code could be reused, but changes had to be made in the way the data is accessed, because the Android guidelines encourage an encapsulated approach over raw SQL queries.

An SQL statement such as

Listing 3.1: Sample SQL INSERT statement

```sql
INSERT INTO
        mints (name, url, pubkey)
VALUES
        (?, ?, ?)
```

would translate to this:

Listing 3.2: Taler Android wallet, MintsDatabase.java

```java
/**
* Adds a new mint to the database
*
* @param mintName the mint's name
* @param mintUrl the mint's URL
* @param publicKey the mint's public EdDSA key
* @return the row id of the newly inserted mint
*/
public long addMint(String mintName, URL mintUrl, EdDSAPublicKey publicKey) {
        ContentValues content = new ContentValues(3);

        content.put(COLUMN_NAME, mintName);
        content.put(COLUMN_URL, mintUrl.toExternalForm());
        content.put(COLUMN_MASTER_PUB, publicKey.getStruct().q_y);

        try (SQLiteDatabase database = databaseHelper.getWritableDatabase()) {
                return database.insert(TABLE_NAME, null, content);
        }
}
```

While there are more sophisticated approaches available in the Android ecosystem, such as *ContentProviders* that would enable cross-application sharing of data (e.g. to a finance application, such as GNUcash [31] or a banking application for allowing bank transfers to a mint), they were not implemented for this iteration of the wallet in order to enable basic functionality first.

To protect the private keys of reserves and coins in the application's database, it is essential that this data is not simply accessible to third parties so that these cannot use the coins stored by the wallet. To this effect, either the private keys themselves, or - ideally - the entire database should be encrypted.

For the initial iteration of the wallet, the former approach was selected. This leaves meta-data, such as payment amounts, the existence of reserves, coins, etc. open to access if another application has elevated privileges. For more information on security considerations of the wallet, see Chapter 5.

The main problem of integrating a fully-encrypted database is the absence of a ready-to-use solution that is compatible with free software (due to the constraints set by the licensing of the Taler project). The makers of SQLite offer an extension to their product in the form of the SEE (*SQLite Encryption Extension*), but it is proprietary [32] and therefore ineligible.

A second alternative is the SQLCipher project [33], which is Open Source, but - like the SEE - requires manual integration and compilation of the source code into the application and replacing of Android's existing SQLite implementation. This approach had to be abandoned due to time constraints but will be added in future revisions.

3.3 Communication

The wallet application communicates with Taler mints via Android's standard HTTP(S) APIs, namely the HttpURLConnection classes [34].

Parsing and generation of JSON data is performed with the Google GSON library [35], which offers more performance and flexibility than the built-in JSON processing libraries.

The communication protocol is defined by the mint and merchant REST API specifications provided by the Taler documentation [26].

3.4 Cryptography

The critical cryptography implementations are all provided by the libgcrypt [12] library, which is used instead of the built-in Spongy Castle framework offered by Android [13].

In order to allow the high-level Java code to interact with the cryptographic underpinnings of the application and to use the resulting data, the inputs and outputs of the libgcrypt and GNUnet libraries have been wrapped to conform with object-oriented paradigms. Because using native code with the JNA exposes memory management intricacies to the normally garbage-collected upper levels, the wrapper objects also have methods to call low-level functions that free the underlying memory areas. These must be triggered manually, since the Java garbage collector is not usually aware of memory allocated outside of its own heap [36].

3.5 Testing

In order to account for the different processor architectures that can run the Android operating system, all testing is currently performed on-device. It is equally possible to perform these tests from within an emulator, should this be necessary (e.g. for continuous integration). The tests are written using the custom JUnit 4 implementation provided by Android [37]. For advanced testing, the Mockito library [38] is used, which provides facilities such as mocking and spies.

Currently, the tests cover the communication classes and the wrapper code for the shared libraries.

Chapter 4

Functionality

4.1 User interface

Due to time constraints, the user interface for the wallet is incomplete. Images here show a preliminary state that does not mark the final state of the wallet. Future iterations will have improved and fully functional versions of what is depicted in this thesis. These images are mainly intended to give the reader a general idea of the layout of the application.

The Taler wallet is launched by tapping on an icon on the home screen (also known as the *launcher*) of an Android device. (Figure 4.1)

The entry screen prompts the user for their passphrase to the wallet database. (Figure 4.2)

After having entered the wallet passphrase, the user has access to their wallet balance, the mints section and their payment history. (Figure 4.3)

From the mint detail screen (figure 4.4), which is accessed by opening a mint from the mints section, the user can create a new reserve for a mint. Once the user has entered the desired amount, the wallet generates a new keypair for the reserve and displays the information necessary for a bank transfer to the mint.

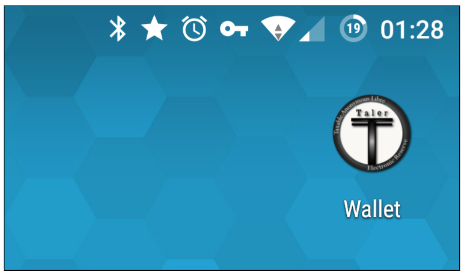


Figure 4.1: Launcher icon on an Android launcher

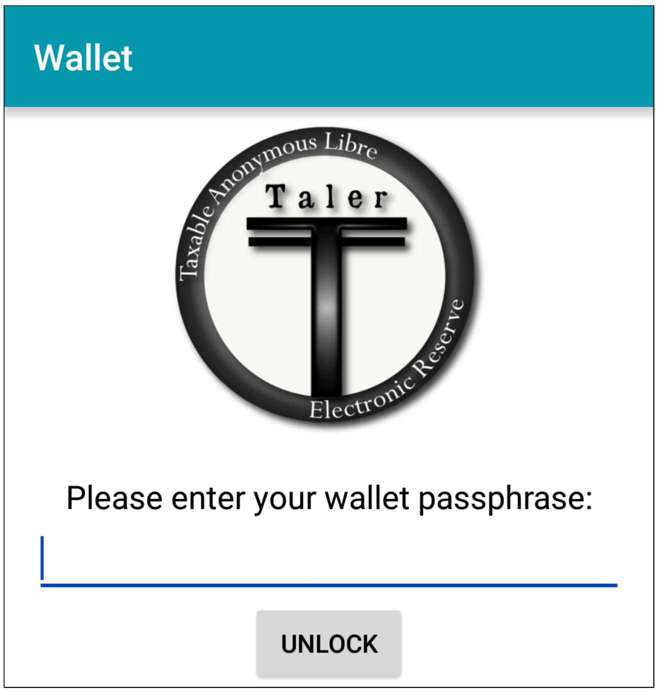


Figure 4.2: Passphrase prompt

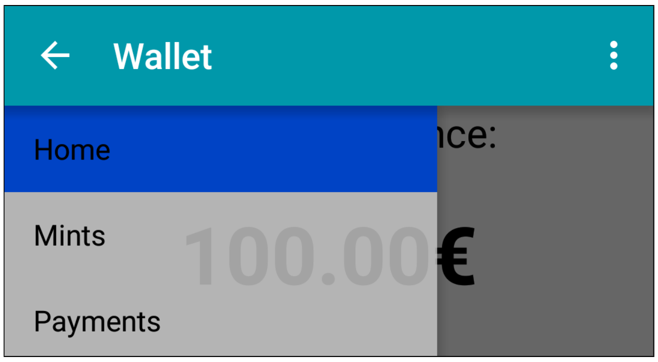


Figure 4.3: Navigation menu

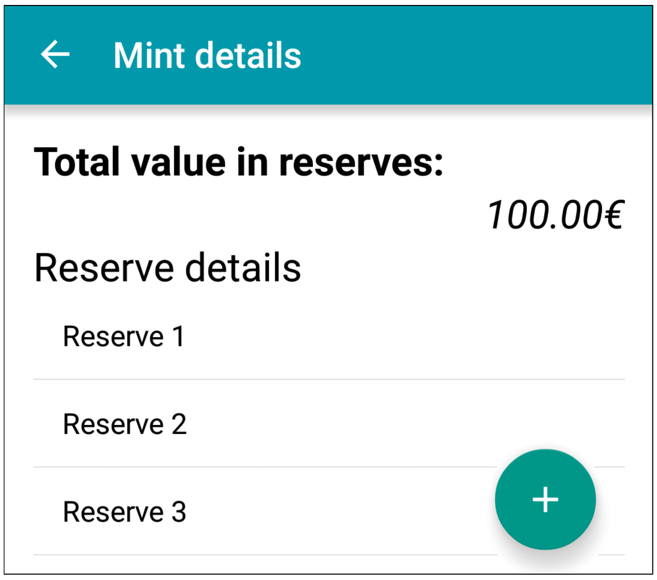
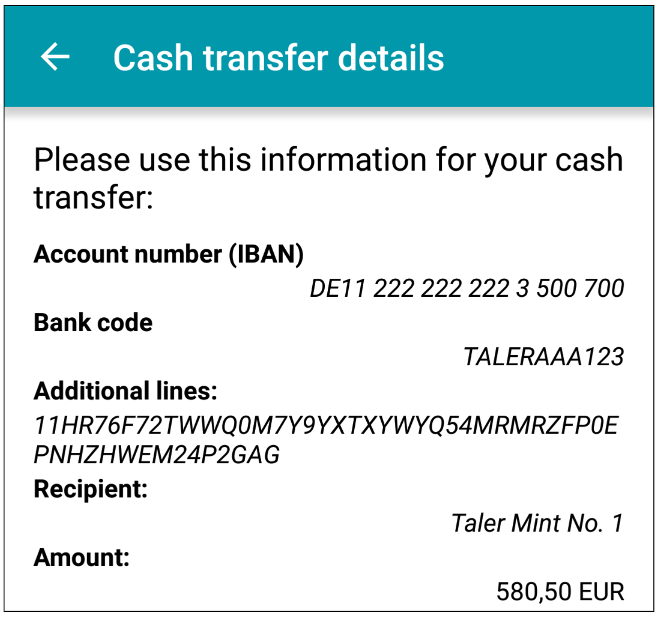


Figure 4.4: Mint detail screen with "floating action button" for adding reserves


Cash transfer details

Please use this information for your cash transfer:

Account number (IBAN)
DE11 222 222 222 3 500 700
Bank code
TALERAAA123
Additional lines:
11HR76F72TWWQ0M7Y9YXTXYWYQ54MRMRZFP0E PNHZHWEM24P2GAG
Recipient:
Taler Mint No. 1
Amount:
580,50 EUR

Figure 4.5: Wire transfer details for a reserve

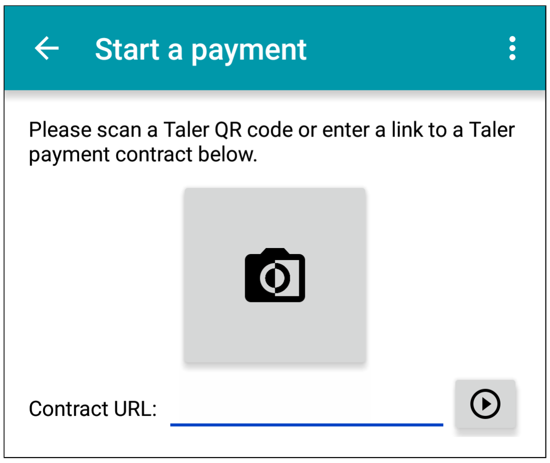

Start a payment
Please scan a Taler QR code or enter a link to a Taler payment contract below.
Contract URL:

Figure 4.6: QR code scanning activity. The large button starts the camera.

In the case of an already existing reserve, which can be opened from the details of a mint, the wallet displays the current value of funds remaining in a reserve.

In order to enable payments, the wallet lets users scan a specialised QR code which contains or links to a contract from a merchant (Figure 4.6). For more information on how this interaction should behave, see chapter 4.2.2.

4.2 Enabling mobile payments

As of writing, the Taler merchant does not provide a method for mobile wallets to perform purchases directly. Therefore, the author proposes a new payment process based on the use of QR codes [39].

4.2.1 Existing payment method

The current method for a merchant to supply a wallet with a contract is based entirely on the interaction of a merchant-hosted website and the Taler browser add-on [27]. Through a series of interactions with the add-on, the merchant implementation is made to generate a *contract*, which the wallet can then use to generate a deposit permission for the merchant, which is subsequently sent to a mint for processing.

4.2.2 Proposed mobile payment method

In order to allow the processing of payments without necessitating the inclusion of a web browser into a mobile Taler wallet (thus introducing more complexity and additional attack surface), a different approach is necessary. Instead of directly communicating with a browser add-on, the merchant should generate a QR code with the necessary data for the wallet, which can be interpreted by a specialised QR code reader application and then passed on to the actual wallet application.

Because the merchant API is still under development, the following proposal should not be regarded as a definite solution to the matter of transferring contracts from a merchant to a mobile wallet, but as a suggestion. Implementation details are intentionally left vague, as they may become unnecessary or conflict with future changes.

A merchant could accommodate for mobile wallets as follows:

- collect items to be purchased, as usual
- offer a mobile Taler payment option in addition to the regular one
- upon selection of this option, generate a QR code containing either the link to the contract URL or the contract itself

The mobile wallet can then, using the device's camera, scan the generated QR code and then access the contract. Should the user not be able or willing to trust the camera or code reader applications, the merchant could introduce an additional factor, such as a code in form of a number or a sequence of colours which could then be used by the wallet to decrypt the contents of the scanned QR code.

The wallet made during the course of this thesis contains stubbed code for interpreting encoded URLs, which can be extended in response to the implementation of the functionality mentioned above into the merchant.

Chapter 5

Security

5.1 Application security

The key security risk of any wallet application is that of attackers gaining access to the user's reserve and coin key pairs, which would enable them to spend existing coins and withdraw new ones from existing reserves, while pretending to be the user. An additional threat to the user is loss of anonymity via meta-data leakage.

Assuming no errors in the implementation of the cryptographic functionality of the wallet, there are several avenues of attack to be considered. The wallet's security is compromised when the attacker

- gains access to the database while the data is not encrypted
- learns the passphrase of the wallet
- can brute-force the key used for encryption of the private keys

In order to prevent an attacker from gaining access to unencrypted keys in memory, the wallet relies on the operating system to prevent access to its own memory [40]. It does not protect itself against attacks that enable attackers to access memory regions in an unprivileged manner, such as direct hardware access. Should the key exist in memory in an unencrypted form and its location is known to the attacker, it is currently not possible to protect the key material. This attack is somewhat mitigated by keeping the derived key for the wallet in memory for a limited time only. When the set time limit expires, the memory containing the key material is cleared and should no longer be recoverable.

When a password manager such as Keepass2Android or LastPass is used or the user saves their passphrase elsewhere on the phone and copies it to the Android operating system's clipboard, it is possible for an attacker to retrieve the passphrase from said clipboard. Access to the clipboard is not secured by the operating system, and any application running on an Android device can monitor the clipboard for changes [41]. To mitigate this attack vector, users of password managers should use (if available) specialised software keyboards that are distributed along with their password manager and refrain from copying sensitive information to the operating system's clipboard altogether. A positive side-effect of using such specialised keyboards is keylogger protection. It is for this reason that password manager use is encouraged. The wallet will warn users against using insecure methods of passphrase entry.

Normally, when an Android device is set up without any modifications, administrative ("*root*") access to the phone is prohibited. Each application - by default - is run under an individual Linux user account which has access only to its own code and resources and external storage, if present [40]. Foreign applications should therefore not have access to the wallet's database. This protection can, however, be removed. There are several factors responsible for the granting of root privileges on Android:

- advanced users wishing to access to features such as advertisement blockers or the removal of unwanted provider-supplied applications installed to the system partition
- accidentally installed malicious software
- exploited security vulnerabilities

The same kind is, within certain limitations, possible with developer access to the device via the *Android Debug Bridge*. While access to the database alone does not immediately present a threat to the user in terms of financial loss, it can result in meta-data leakage.

With access to the database files it is also possible to brute-force the encryption key. This should not, assuming current processing power, prove to be a problem if the key derivation function used for the key was set up to use secure parameters, such as a high amount of iterations and a secure passphrase, as recommended by the standard defining the *scrypt* key derivation function [42].

Access to the files of the application is not the only avenue attackers can take to access the application. With access to the Linux underpinnings of the Android operating system, it is - for example - possible to access the device's frame buffer directly and create an image of any application currently visible on the screen [43]. The same applies for many other methods of accessing the hardware of the device running the wallet directly, e.g. keyloggers or custom code that can access the device's memory [44].

There is no direct mitigation for attacks based on privileged access. The Taler wallet will, therefore, warn users about the dangers of having root access on the device running the wallet in order to make the user aware of the risks involved, but will not prevent them from using it.

5.2 Communications security

To protect the customer from meta-data leakage, it is important for the mints and merchants to implement HTTPS encryption. While the Taler protocol and the use of blind signatures do not necessitate the use of additional encryption, possibly sensitive information such as a reserve's transaction history are transmitted in the clear. The contents of this communication, depending on the user and their situation, may be useful to active observers and could be used against the user. The decision of enabling additional encryption lies outside of the scope and responsibility of the wallet, but should be an important consideration for both merchants and mints.

One of the main aspects of the Taler payment system is the anonymity of the customer. This can, however, only be guaranteed if the user is able to perform transactions over a channel that itself provides both confidentiality and anonymity.

At present, the most viable option for achieving both of these goals would be *Tor* project, which provides network level anonymity by routing data through several intermediaries and ensures confidentiality during transmission through its network by repeated application of encryption. It does not provide confidentiality for data leaving the network. [45–47]

By using the Tor network for communicating with mints and merchants (assuming the merchant or mint do and can not save additional personally identifying information), the origin of communications from the wallet are obscured, preventing their association with an IP address that is tied to the customer.

Chapter 6

Conclusion

6.1 Transformative effects of Taler and wallet

Much of the potential of GNU Taler hinges on its taxability and auditability. These properties make it more likely to be adopted as an officially accepted form of payment by states, possibly even becoming an accepted equivalent to physical currency. There is also a financial incentive - physical currency is expensive. This, of course, varies from country to country, depending on a variety of factors. Shifting entirely to electronic forms of payment may, however, according to Humphrey, et al. [48], save a country up to 1% of its GDP every year.

The lowered transaction costs caused by a shift to non-traditional forms of payment could also contribute to the acceptance of Taler by merchants. Provided that a Taler mint can be operated with acceptable costs for the withdrawal, deposit and refreshing procedures, it should be able to easily compete against traditional payment providers such as credit card companies or services like PayPal. Transaction costs of these providers are significantly higher than those of direct payments [49].

Having a mobile form of a Taler wallet can act as a catalyst for the adoption of Taler, because it enables payments not only on websites and services on the Internet, but also allows users to perform transactions in places where they would normally use cash or resort to privacy-infringing payment payment methods, such as shops or restaurants.

Lowered transaction costs also open avenues to new forms of services which would otherwise appear infeasible: payments for small "services" such as a fixed amount per tweet or blog article could become more common, since the small units of payment (*micropayments*) would not be dwarfed by much higher fixed costs for payment providers. An example for such a service would be Patreon [50], which provides regular payments to artists and other content producers in monthly instalments or per work unit, based on the donations given by users.

Taler's anonymity properties are also useful for donating funds to otherwise "risky" recipients, such as WikiLeaks. While it may not illegal to donate to such an organisation, it can be made difficult [51]. Using Taler replaces a centralised middle man (in this case, payment providers like MasterCard, Visa or PayPal) with a mint. Customers can, for instance, choose a mint in a different jurisdiction and thus avoid restrictions imposed by outside influences. Additionally, the anonymity preserving properties of Taler may encourage more donations of this kind, since privacy-concious users need not worry about being connected to such a donation.

6.2 Future work

Due to the unexpectedly high amount of work necessary to port the necessary components to the Android platform in a satisfactory manner, there are several areas in which the Wallet application could be improved.

First and foremost is the integration of the refreshing protocol for Taler coins. Without it, the coins in the wallet can be used only within the initial expiration time frame. Coins that have been partially spent can, without refreshing, be used to deanonimise their users.

With the refreshing protocol in place, the wallet should be extended with an optimisation routine to enable cost-efficient payment, refreshing and withdrawals, all of which can incur fees for the wallet owner when performed. This feature was planned for the initial release of the wallet but had to be postponed in favour of basic functionality.

Another important feature to be implemented is the use of a wholly encrypted database. This allows the wallet to protect its meta-data from outside access, such as the payment history, coin withdrawals and what mints and merchants are frequently used. To this end, the wallet will include the Guardian Project's SQLCipher library.

Making the database accessible from other applications in a controlled manner (e.g. via authentication codes) is an optional feature that may be useful in the future, depending on user demand.

Currently, the code for accessing the native libraries is supplied via a pre-compiled binary library. This is not desirable, since it is not possible to easily confirm whether the source code provided by the author and the sorce code used to create the compiled library are identical. Therefore, the source code for the Java Native Access framework (and the underlying *libffi* library) will be integrated into the Taler wallet. The build process for this library is outdated and incompatible with the rest of the current implementation, making further efforts necessary to enable compilation from scratch.

When the important features are rolled out, the user interface code will require a complete overhaul in order to conform with modern usability standards. While it is entirely functional in its current state, there is space for improvement.

The user interface overhaul will also require a reimplementation of the multi-threading and service infrastructure - this will allow the wallet to be more reliable when handling sudden changes in configuration and improve performance.

Bibliography

[1] Various, *Android Pay*, Google Inc., 2015, accessed: 2015-10-10. [Online]. Available: https://www.android.com/pay/

[2] Various, *Apple Pay*, Apple Inc., 2015, accessed: 2015-10-10. [Online]. Available: https://www.apple.com/apple-pay/

[3] Various, *Samsung Pay - Safe and Simple Mobile Payments*, Samsung Group, 2015, accessed: 2015-10-13. [Online]. Available: http://www.samsung.com/us/samsung-pay/

[4] Various. (2015) Paypal: Pay, send money & accept payments. PayPal Holdings, Inc. Accessed: 2015-10-10. [Online]. Available: https://www.paypal.com/uk/webapps/mpp/personal

[5] Various, *Paydirekt*, paydirekt GmbH, 2015, accessed: 2015-10-10. [Online]. Available: https://www.paydirekt.de/mehr-informationen-zu-paydirekt.html

[6] S. Nakamoto. (2008, October) Bitcoin: A peer-to-peer electronic cash system. Accessed: 2015-10-10. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[7] Various, *Bitcoin HowTo*, Microsoft Inc., 2015, accessed: 2015-10-10. [Online]. Available: https://commerce.microsoft.com/PaymentHub/Help/Right?helppagename=CSV_BitcoinHowTo.htm

[8] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 397–411, accessed: 2015-10-14. [Online]. Available: https://isi.jhu.edu/~mgreen/ZerocoinOakland.pdf

[9] U. N. Y. Southern, *Manhattan U.S. Attorney Announces Forfeiture Of $28 Million Worth Of Bitcoins Belonging To Silk Road*, 2014, accessed: 2015-10-14. [Online]. Available: http://www.justice.gov/usao-sdny/pr/manhattan-us-attorney-announces-forfeiture-28-million-worth-bitcoins-belonging-silk

[10] F. Dold, S. H. Totakura, B. Müller, and C. Grothoff, *Taler: Taxable Anonymous Libre Electronic Reserves*, GNUnet e.V., 2015, accessed: 2015-10-11.

[Online]. Available: http://www.git.taler.net/?p=mint.git;a=tree;f=doc/paper;h=a273a642c281bc7eee21a26427ea32e81e0b630e;hb=HEAD

[11] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*. Springer, 1983, pp. 199–203.

[12] Various, *The Libgcrypt Reference Manual*, g10code GmbH and the Free Software Foundation, September 2015, accessed: 2015-10-11. [Online]. Available: https://www.gnupg.org/documentation/manuals/gcrypt/

[13] R. Tyley, *SpongyCastle*, 2015, accessed: 2015-10-11. [Online]. Available: https://rtyley.github.io/spongycastle/

[14] Various, *Google Android sources, external modules*, Google Inc., 2015, accessed: 2015-10-11. [Online]. Available: https://android.googlesource.com/platform/external/bouncycastle/

[15] S. H. Totakura and C. Grothoff, *GNUnet RSA cryptography source code*, GNUnet e.V., 2015, accessed: 2015-10-11. [Online]. Available: https://gnunet.org/svn/gnunet/src/util/crypto_rsa.c

[16] Various, *BoringSSL*, Google Inc., 2015, accessed: 2015-10-11. [Online]. Available: https://www.chromium.org/Home/chromium-security/boringssl

[17] K. Michaelis, C. Meyer, and J. Schwenk, "Randomly failed! the state of randomness in current java implementations," Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Tech. Rep., 2013, accessed: 2015-10-05. [Online]. Available: https://hgi.rub.de/media/nds/veroeffentlichungen/2013/03/25/paper_2.pdf

[18] A. Klyubin. (2013) Some securerandom thoughts. Google Inc. Accessed: 2015-10-05. [Online]. Available: http://android-developers.blogspot.de/2013/08/some-securerandom-thoughts.html

[19] Various, *GnuPG for Android - Makefile for external sources*, 2015. [Online]. Available: https://github.com/guardianproject/gnupg-for-android/blob/master/external/Makefile

[20] F. Dold, S. H. Totakura, B. Müller, and C. Grothoff, *Taler utility library source code*, GNUnet e.V., 2015, accessed: 2015-10-12. [Online]. Available: http://www.git.taler.net/?p=mint.git;a=tree;f=src/util;h=610b3f8ee3fac5b3abc9404e4b992280a0f8c75d;hb=99865ad6d46c61eea43b59c110730d6781aade6d

[21] T. Wall *et al.*, *Java Native Access source code*, 2015, accessed: 2015-10-12. [Online]. Available: https://github.com/java-native-access/jna

[22] Various, *Getting Started with the NDK*, Google Inc., 2015, accessed: 2015-10-14. [Online]. Available: https://developer.android.com/ndk/guides/concepts.html

[23] S. Liang, *The Java™ Native Inteface.* Addison-Wesley, 1999, accessed: 2015-10-14. [Online]. Available: https://www.fer.unizg.hr/_download/repository/jni.pdf

[24] Various, *SWIG-3.0 Documentation*, 2015, accessed: 2015-10-12. [Online]. Available: http://www.swig.org/Doc3.0/index.html

[25] O. Chalfik *et al.*, *BridJ documentation*, 2015, accessed: 2015-10-11. [Online]. Available: https://github.com/nativelibs4java/BridJ

[26] F. Dold, S. H. Totakura, B. Müller, and C. Grothoff, *The Mint RESTful JSON API*, GNUnet e.V., 2015, accessed: 2015-10-11. [Online]. Available: http://api.taler.net/api-mint.html

[27] F. Dold, S. H. Totakura, B. Müller, M. Stanisci, and C. Grothoff, *The Merchant API*, GNUnet e.V., 2015, accessed: 2015-10-11. [Online]. Available: http://api.taler.net/api-merchant.html

[28] Various, *AsyncTask documentation*, Google Inc., 2015, accessed: 2015-10-14. [Online]. Available: https://developer.android.com/reference/android/os/AsyncTask.html

[29] Various, *Services*, Google Inc., 2015, accessed: 2015-10-14. [Online]. Available: https://developer.android.com/guide/components/services.html

[30] Various, *Saving Data in SQL Databases*, Google Inc., 2015, accessed: 2015-10-11. [Online]. Available: https://developer.android.com/training/basics/data-storage/databases.html

[31] N. Fet *et al.*, *GnuCash Android*, 2015, accessed: 2015-10-11. [Online]. Available: https://github.com/codinguser/gnucash-android

[32] Various, *The SQLite Encryption Extension (SEE)*, Hipp, Wyrick & Company, Inc., 2015, accessed: 2015-10-11. [Online]. Available: http://www.hwaci.com/sw/sqlite/see.html

[33] Various, *SQLCipher: Encrypted Database*, The Guardian Project, 2015, accessed: 2015-10-11. [Online]. Available: https://guardianproject.info/code/sqlcipher/

[34] Various, *HttpURLConnection documentation*, Google Inc., 2015, accessed: 2015-10-11. [Online]. Available: https://developer.android.com/reference/java/net/HttpURLConnection.html

[35] I. Singh and J. Leitch, *Gson Design Document*, Google Inc., 2015, accessed: 2015-10-11. [Online]. Available: https://sites.google.com/site/gson/gson-design-document

[36] S. Liang, T. Fast, and T. Wall, *Memory (JNA API)*, 2015, accessed: 2015-10-10. [Online]. Available: https://java-native-access.github.io/jna/4.2.0/

[37] Various, *Building Local Unit Tests*, Google Inc., 2015, accessed: 2015-10-13. [Online]. Available: https://developer.android.com/training/testing/unit-testing/local-unit-tests.html

[38] S. Faber *et al.*, *Mockito*, 2015, accessed: 2015-10-11. [Online]. Available: http://mockito.org/

[39] M. Hara *et al.*, "Method and apparatus for reading an optically two-dimensional code." Patent EP0 672 994, September, 1995, accessed: 2015-10-11. [Online]. Available: http://worldwide.espacenet.com/publicationDetails/originalDocument?CC=EP&NR=0672994A1&KC=A1&FT=D&ND=3&date=19950920&DB=EPODOC&locale=en_EP

[40] Various, *System Permissions*, Google Inc., 2015, accessed: 2015-10-05. [Online]. Available: https://developer.android.com/guide/topics/security/permissions.html

[41] Various, *ClipboardManager documentation*, Google Inc., 2015, accessed: 2015-10-05. [Online]. Available: https://developer.android.com/reference/android/content/ClipboardManager.html

[42] C. Percival and S. Josefsson, "The scrypt password-based key derivation function," 2015, accessed: 2015-10-14. [Online]. Available: https://tools.ietf.org/html/draft-josefsson-scrypt-kdf-00

[43] Various. (2012, June) Android take screenshot on rooted device. Post by "cgolden". [Online]. Available: https://stackoverflow.com/questions/10965409/android-take-screenshot-on-rooted-device

[44] P. Teoh, *How to dump memory of any running processes in Android (rooted)*, December 2011, accessed: 2015-10-14. [Online]. Available: https://tthtlc.wordpress.com/2011/12/10/how-to-dump-memory-of-any-running-processes-in-android-2/

[45] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004, accessed: 2015-10-14. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA465464

[46] Various, *Top changes in Tor since the 2004 design paper (Part 1)*, 2012, accessed: 2015-10-12. [Online]. Available: https://blog.torproject.org/blog/top-changes-tor-2004-design-paper-part-1

[47] Various, *Top changes in Tor since the 2004 design paper (Part 2)*, 2012, accessed: 2015-10-12. [Online]. Available: https://blog.torproject.org/blog/top-changes-tor-2004-design-paper-part-2

[48] D. Humphrey, M. Willesson, T. Lindblom, and G. Bergendahl, "What does it cost to make a payment?" *Review of network economics*, vol. 2, no. 2, 2003, accessed:

2015/10/14. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.496.5865&rep=rep1&type=pdf

[49] R. M. Grüschow, J. Kemper, and M. Brettel, "Do transaction costs of payment systems differ across customers in e-commerce?" 2015, accessed: 2015/10/14. [Online]. Available: http://aisel.aisnet.org/cgi/viewcontent.cgi?article=1063&context=ecis2015_cr

[50] Various, *What is Patreon?*, Patreon, Inc., 2015, accessed: 2015-10-14. [Online]. Available: https://patreon.zendesk.com/hc/en-us/articles/204606315-What-is-Patreon-

[51] Unknown, *Banking Blockade*, WikiLeaks, June 2011, accessed: 2015-10-15. [Online]. Available: https://wikileaks.org/Banking-Blockade.html