Cashless to E-Cash

Bachelor's Thesis

Course of study	Bachelor of Science in Computer Science
Author	Joel Roman Häberli
Advisor	Prof. Dr. Benjamin Fehrensen
Co-advisor	Prof. Dr. Christian Grothoff
Expert	Dr. Alain Hiltgen, UBS

Version 1.0 of April 21, 2024

- Technic and Computer Science
- Institute for Cybersecurity and Engineering ICE

Abstract

In order to buy Taler, the *Taler Exchange* needs guarantees to legally secure the payment. Buying Taler physically establishes direct trust, since cash can be used in order to buy Taler and the transaction is completed. If you want to buy Taler using cashless systems like credit cards, the Exchange has no proof that the payment has succeeded. In order to fill this cap, this thesis proposes a framework allowing cashless withdrawals using Taler. A reference implementation is created which establishes a trust relationship between the terminal manufacturer Wallee and the *Taler Exchange* through a newly created component called *C2EC*. This enables a trust relationship between the *Taler Exchange* and the terminal operator which allows withdrawing Taler without using cash. The liability for the digital cash is on the side of the terminal operator and therefore establishes the guarantees for the *Taler Exchange*.

Contents

Abstract iii

1. Introduction 1
   1.1. Motivation . . . 1
   1.2. Perspectives . . . 2
      1.2.1. Taler Exchange (C2EC) . . . 2
      1.2.2. Terminal Application . . . 2
      1.2.3. Taler Wallet . . . 2
   1.3. Goal . . . 2
      1.3.1. C2EC . . . 3
      1.3.2. Wallee POS Terminal . . . 3

2. Overview 5
   2.1. Components . . . 5
   2.2. Process . . . 5
      2.2.1. The Terminal . . . 7
      2.2.2. The C2EC . . . 8
      2.2.3. The Wallet . . . 9

3. Architecture 11
   3.1. C2EC . . . 11
      3.1.1. C2EC Perspective . . . 11
      3.1.2. Withdrawal-Operation state transitions . . . 11
      3.1.3. Authentication . . . 12
      3.1.4. The C2EC RESTful API . . . 13
      3.1.5. Taler Wirewatch Gateway API . . . 14
      3.1.6. The C2EC database . . . 16
   3.2. Payto wallee-transaction extension . . . 17
      3.2.1. Payto refund using Wallee . . . 18
      3.2.2. Extensibility . . . 18
   3.3. Taler Wallet . . . 18
      3.3.1. Taler Wallet Perspective . . . 19
   3.4. Wallee . . . 19
      3.4.1. Wallee Perspective . . . 19
      3.4.2. Wallee Terminal . . . 19
      3.4.3. Wallee Backend and API . . . 20

4. Implementation 23
4.1. Concepts . . . 23
4.1.1. Consumers and Producers . . . 23
4.1.2. Long-Polling . . . 23
4.1.3. Publish-Subscribe Pattern . . . 23
4.1.4. Go Language . . . 24
4.2. Database . . . 25
4.2.1. Schema . . . 25
4.2.2. Triggers . . . 25
4.3. C2EC . . . 25
4.3.1. Decoupling steps . . . 25
4.3.2. Bank-Integration API . . . 26
4.3.3. Wire-Gateway API . . . 27
4.3.4. Payment Attestation . . . 27
4.3.5. Wallee Client . . . 29
4.3.6. Security . . . 29
4.4. Wallee POS Terminal . . . 31
4.5. Wallet . . . 31

5. Results 33
5.1. Discussion . . . 33
5.2. Results . . . 33

Bibliography 37

List of Figures 39

List of Tables 41

Listings 43

Glossary 45

A. Appendix A 47
A.1. API . . . 47

B. Appendix B 57
B.1. Meeting notes . . . 57

1. Introduction

1.1. Motivation

Which payment systems do you use in your daily live and why? Probably one you know it is universally accepted, reliable, secure and the payment goes through more or less instantly.

The **universal acceptance** was identified as one of the most important aspects in a report which was published on behalf of the ECB (European Central Bank) in march 2022 as result of a focus group concerning the acceptance of a digital euro [1] as new payment system. The universal acceptance was even identified as *the* most important property amongst the general public and tech-savvy people in the report [2].

In a world, where everything is connected and everything is accessible from everywhere (one might think), it is therefore very important to make it as easy as possible to on-board people on a product. This is also the case for Taler. For a wide acceptance of the payment system Taler, it is important that various ways exist to withdraw digital cash in Taler.

This is where this thesis hooks in. Currenlty it is possible to withdraw digital cash using Taler at a Bank which runs a *Taler Exchange* and integrates the respective API. At time of this writing only one Bank is in the process of running a *Taler Exchange*. At the Berner Fachhochschule an *Exchange* is operated and digital cash can be withdrawn at the secretariat using cash.

To make the access to digital cash using Taler easier and allow faster spreading of the payment system Taler, a framework for cashless withdrawal of digital cash is proposed and implemented in order to open new doors for the integration and adoption of the Taler payment system within the society.

To make the withdrawals using a credit card possible, various loose ends must be put together within the Taler ecosystem and the terminal provider.

Therefore a new component C2EC shall help, establishing a trustworthy relationship, which makes it possible for the *Exchange* to issue digital cash to a customer. Therefore the *Exchange* is not putting his trust on cash received but rather on the promise of a trusted third party (a terminal provider) to put the received digital cash in a location, controlled by the *Exchange* eventually (e.g. a bank account owned by the *Exchange*).

This enables a broader group of people to leverage Taler for their payments. Which eventually leads to a wider adoption of the payment system Taler.

1.2. Perspectives

During the initial analysis of the task, three areas of work were discovered. One is the *Taler Exchange*, one the Application for the terminal and the (Taler) *Wallet*. This led to different views on the system by two different players within it. To allow a more concise view on the system and to support the readers and implementer, two perspectives shall be kept in mind. They have different views on the process but need to interact with each other seamlessly.

1.2.1. Taler Exchange (C2EC)

The perspective of the *Taler Exchange* includes all processes within C2EC component and the interaction with the terminal application, terminal backend and the wallet of the user. The *Taler Exchange* wants to allow withdrawal of digital digital cash only to users who pay the equivalent value to the *Exchange*. The *Exchange* wants to stay out of any legal implications at all costs.

1.2.2. Terminal Application

The perspective of the terminal application includes all processes within the application which interacts with the user, their *Wallet* and credit card allowing the withdrawal of digital cash. The terminal application wants to conveniently allow the withdrawal of digital cash and charge fees to cover its costs and risks.

1.2.3. Taler Wallet

The *Wallet* holds the digital cash owned by the customer. The *Wallet* wants to eventually gather the digital cash from the *Taler Exchange*. The owner of the *Wallet* must therefore present their credit card at a *Terminal* of the terminal provider and pay the *Exchange* as well accept the fees of the provider.

1.3. Goal

The goal of this thesis is to propose a framework for cashless withdrawals and implement the process which allows withdrawing Taler using a credit card at a terminal of the terminal provider *Wallee*.

1.3.1. C2EC

Therefore a new component, named C2EC, will be implemented as part of the Taler Exchange. C2EC will mediate between the Taler Exchange and the terminal provider. This includes checking that the transaction of the debitor reaches the account of the Exchange and therefore the digital currency can be withdrawn by the user, using its Wallet.

1.3.2. Wallee POS Terminal

The Wallee payment terminal, also called Point of Sales (POS) terminal, interfaces with payment cards (Credit Cards, Debit Cards) to make electronic fund transfers, i.e. a fund transfer to a given GNU Taler Exchange. For our purpose, we will extend the functionality of the terminal to initiate the corresponding counter payment from the exchange to the GNU Taler wallet of the payee.

2. Overview

2.1. Components

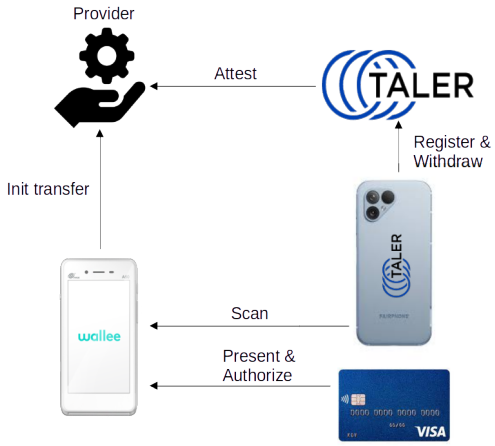

Figure 2.1.: Involved components and devices

The component diagram shows the components involved by the withdrawal using the terminal. Besides the credit card owned by the user, two systems are involved and within each system two components are required to fulfill the task. The Taler ecosystem which represents the Taler Wallet and the Taler Exchange (C2EC is a part of the Exchange) involved in the withdrawal process. In the Terminal system, the terminal and the backend system of the terminal manufacturer are leveraged in the process.

2.2. Process

The figure 2.2 shows a high level overview of the components involved and how they interact. In an initial step (before the process is effectively started as depicted), the customer or owner of the terminal selects the *Exchange*, which shall be used for the withdrawal. The process is then started. The numbers in the diagrams are picked up by the description of the steps what is done between the different components:

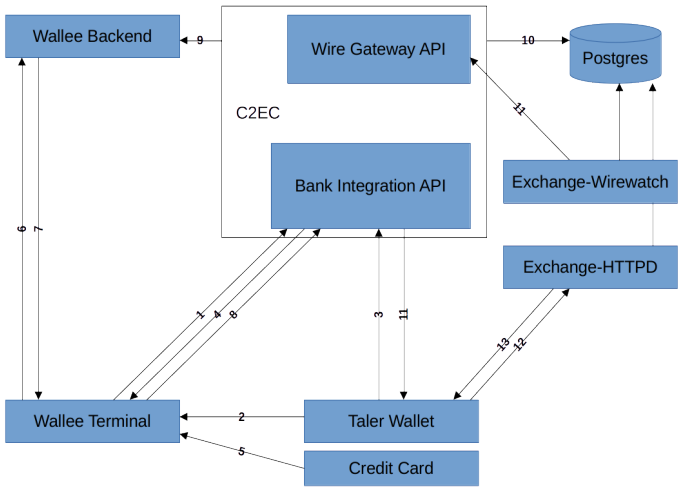

Figure 2.2.: Diagram of included components and their interactions

1. Wallee Terminal requests to be notified when parameters are *selected* by C2EC.
2. The Wallet scans the QR code at the Terminal.
3. The Wallet registers a reserve public key and the *wopid*.
4. The Bank-Integration API of C2EC notifies the Terminal, that the parameters were selected.
5. The POS initiates a payment to the account of the GNU Taler Exchange. For the payment the POS terminal requests a payment card and a PIN for authorizing the payment.
6. The Terminal triggers the payment at the Wallee Backend.
7. The Terminal receives the result of the payment.
   a) successful
   b) unsuccessful
8. The Terminal sends the payment notification to the Bank Integration API of C2EC.
9. The C2EC component approves the payment by requesting the transaction of the Wallee Backend.
10. C2EC updates the database by either setting the status of the withdrawal operation to *confirmed* or *abort* (depending on the response of the Wallee Backend).

11. Now decoupled from each other the Exchange-Wirewatch asks the Wire Gateway API of C2EC for a list of transactions and the Bank-Integration API sends a *confirmed* or *abort* message to the wallet.

12. The Wallet asks the Exchange to be notified, when a reserve with the reserve public key becomes available.

13. The Exchange can send the digital cash back to the Wallet.

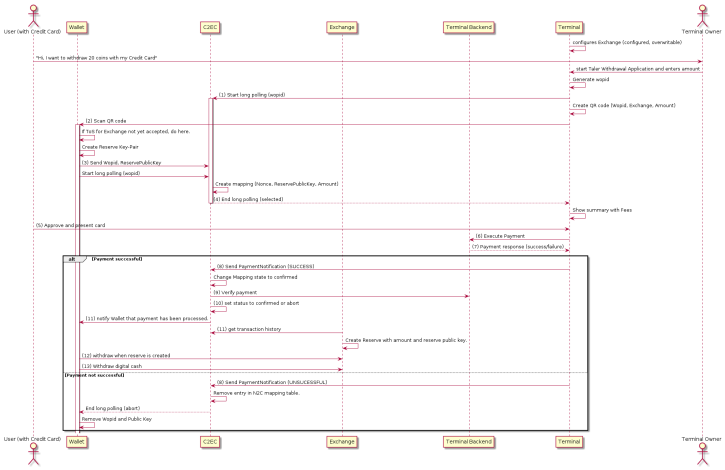

Figure 2.3.: Process of a withdrawal using a credit card

The diagram in figure 2.3 shows the high level flow to withdraw digital cash using the credit card terminal and Taler. It shows when the components of figure 2.2 interact with each other. It shows the implementation of the flow. Terminal, Wallet and Exchange are linked leveraging a *wopid* initially generated by the terminal and presented to the Exchange by the withdrawing Wallet accompanied by a reserve public key.

The process requires the Terminal, the Wallet, the C2EC component and the Exchange which interact with each other. In this section the highlevel process as showed in figure 2.3 is explained.

2.2.1. The Terminal

The Terminal initiates the withdrawal leveraging an application which works as follows:

1. At startup of the application, the terminal loads the C2EC configuration

2. When a user wishes to do a withdrawal, the owner of the terminal opens the application and initiates a new withdrawal. A withdrawal is basically a funds transfer to the IBAN account of the *Exchange*.

a) Application creates a *wopid*

b) The application starts long polling at the C2EC and awaits the selection of the reserve parameters mapped to the *wopid*. The parameters are sent by the Wallet to C2EC.

c) *Wopid* is packed into a QR code (with Exchange and amount entered by the terminal owner)

d) Terminal calculates fees and shows summary and the Terms of Service (ToS) of Taler.

e) The user accepts the offer, agrees with the ToS

f) QR code is displayed

3. The user now scans the QR Code using his Wallet.
4. The application receives the notification of the C2EC, that the parameters for the withdrawal were selected.
5. The Terminal executes the payment (after user presented their credit card, using the Terminal Backend).
6. The terminal initiate the fund transfer to the *Exchange*. The customer has to authorize the payment by presenting his payment card and authorizing the transaction with his PIN. The terminal processes the payment over the an available connector configured on the *Wallee Backend*. Possible connectors are Master Card, VISA, TWINT, Maestro, Post Finance, and others [3].

   a) It presents the result to the user.

   b) It tells the C2EC, that the payment was successful.

2.2.2. The C2EC

The C2EC component manages the withdrawal using a third party provider (e.g. Wallee) and seeks guarantees in order to create a reserve containing digital cash which can be withdrawn by the Wallet.

1. C2EC retrieves a long polling request for a *wopid* (from the Terminal).
2. C2EC creates a mapping entry with the *wopid* and an empty reserve public key field
3. C2EC retrieves a request including a *wopid* and a reserve public key.
4. C2EC validates the request and adds the key to the mapping. This establishes the *wopid* to reserve public key mapping.

5. C2EC ends the long polling from the terminal (by sending back the reserve public key).
6. C2EC receives payment notification of the terminal.
7. C2EC verifies the notification by asking the terminal backend for confirmation.
8. C2EC, upon successfully checking the notification, checks that the transaction went through and therefore a reserve is created by the wirewatch gateway (using the public key in the payment purpose field).

2.2.3. The Wallet

The Wallet must attest its presence to the terminal by registering a *wopid* and belonging reserve public key which will holds the digital currency that can eventually be withdrawn by the Wallet.

1. The Wallet scans the QR Code (*wopid*, Exchange information and amount) on the Terminal
2. It creates a reserve key pair
3. The Wallet sends the reserve public key and the scanned *wopid* to the C2EC
4. The Wallet can withdraw digital cash from the created reserve.

3. Architecture

3.1. C2EC

The C2EC (**c**ashless**2ec**ash) component is the central coordination component in the cashless withdrawal of digital cash using Taler. It initializes the parameters and mediates between the different stakeholders of a withdrawal, which finally allows the customer to withdraw digital cash from a reserve owned by the *Exchange*. Therefore C2EC provides API which can be integrated and used by the *Terminal*, *Wallet* and the *Exchange*.

The API of the C2EC (cashless2ecash) component handles the flow from the creation of a C2EC mapping to the creation of the reserve. For the integration into the Taler ecosystem, C2EC must implement the Taler Wirewatch Gateway API [4] and the Taler Bank Integration API [5].

The exact specification can be found in the official Taler docs repository as part of the core specifications of the bank integration [5] and wire gateway [4]

3.1.1. C2EC Perspective

From the perspective of C2EC, the system looks as follows:

- Is requested by the *Taler Wallet* to register a new *wopid* to reserve public key mapping.
- Is notified by the *Wallee Terminal* about a payment.
- Attests a payment by requesting the payment proof at the *Wallee Backend*
- Supplies the Taler Wire Gateway API that the respective *Exchange* can retrieve new transactions and create reserves which are then created and can be withdrawn by the *Taler Wallet*.

3.1.2. Withdrawal-Operation state transitions

Basically C2EC mediates between the stakeholders of a withdrawal in order to maintain the correct state of the withdrawal. Therefore it decides when a withdrawal's status can be transitioned. The diagram in figure 3.1 shows the transitions of states in which a withdrawal operation can be and which events will trig-

ger a transition. The term attestation in this context means, that the backend of the provider was asked and the transaction was successfully processed (or not). So if a transaction was successfully processed by the provider, the final state is the success case *confirmed*, where the *Exchange* will create a reserve and allow the withdrawal. If the attestation fails, thus the provider could not process the transaction successfully, the failure case *aborted*, is reached as final state.

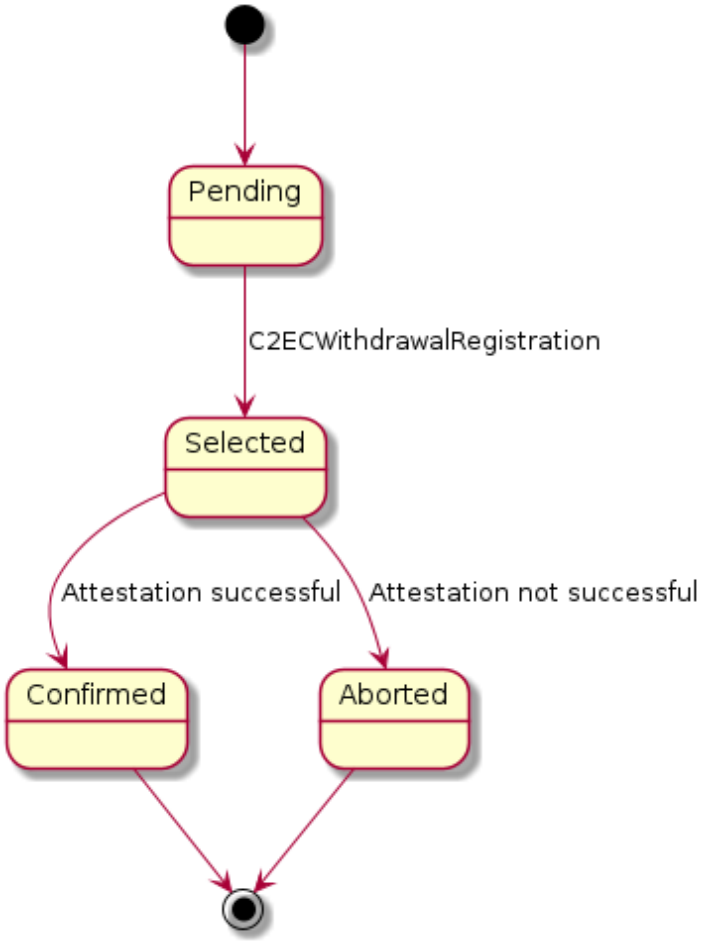

Figure 3.1.: Withdrawal Operation state transition diagram

3.1.3. Authentication

Terminals and the Exchange which authenticate against the C2EC API must provide their respective access token. Therefore, they provide a `Authorization: Bearer $ACCESS_TOKEN` header, where `$ACCESS_TOKEN` is a secret authentication token configured by the exchange and must begin with the prefix specified in RFC 8959 [6]: *secret-token*.

3.1.4. The C2EC RESTful API

This section describes the various API implemented in the C2EC component. The description contains a short list of the consumers of the respective API. Consumer in this context does not necessarily mean that data is consumed but rather that the consumer uses the API to either gather data or send data to C2EC.

Taler Bank Integration API

Withdrawals with a C2EC are based on withdrawal operations which register a withdrawal identifier (nonce) at the C2EC component. The provider must first create a unique identifier for the withdrawal operation (the `WOPID`) to interact with the withdrawal operation and eventually withdraw using the wallet. The withdrawal operation API is an implementation of the *Bank Integration API* [5].

GET - config

- **Method:** GET
- **Endpoint:** /config
- **Description:** Return the protocol version and configuration information about the C2EC API.
- **Response:** HTTP status code 200 OK. The exchange responds with a `C2ECConfig` object.
- **Consumers:** Components who want to use the API and therefore want to load the config of the instance.

POST - withdrawal-operation

This API is not specified within the standard Bank Integration API and therefore an extension to the official specification. The Wallet must implement the initialization through this flow.

- **Method:** POST
- **Endpoint:** /withdrawal-operation
- **Description:** Initiate the withdrawal operation, identified by the `WOPID`.
- **Request:** The request body contains a `C2ECWithdrawRegistration` object.
- **Response:** The response is HTTP status code 204 No Content on success and a 400 or 500 status code on failure (with respective `ErrorDetail`)
- **Consumers:** The *Taler Wallet* registers and initializes the withdrawal operation through this API.

GET - withdrawal-operation by wopid

- **Method:** GET
- **Endpoint:** /withdrawal-operation/$WOPID
- **Description:** Query information about a withdrawal operation, identified by the `WOPID`.
- **Response:** HTTP status code 200 OK and body containing a `C2ECWithdrawalStatus` object or 404 Not found.
- **Consumers:** The API is used by the *Terminal* and *Taler Wallet* to retrieve information about the current state of the withdrawal operation. The API allows long-polling and can therefore be used by the consumer to be updated if the status of the withdrawal operation changes.

POST - withdrawal-operation by wopid

- **Method:** POST
- **Endpoint:** /withdrawal-operation/$WOPID
- **Description:** Notifies C2EC about an executed payment for a specific withdrawal.
- **Request:** The request body contains a `C2ECPaymentNotification` object.
- **Response:** HTTP status code 204 No content, 400 Bad request, 404 Not found, or 500 Internal Server error.
- **Consumers:** The API is used by the *Terminal* to notify the C2EC component that a payment was made and to give the C2EC component information about the payment itself (e.g. the provider specific transaction identifier).

3.1.5. Taler Wirewatch Gateway API

The Taler Wirewatch Gateway [4] must be implemented in order to capture incoming transactions and allow the withdrawal of digital cash. The specification of the Taler Wirewatch Gateway can be found in the official Taler documentation [4].

The wirewatch gateway helps the Exchange communicate with the C2EC component using a the API. It helps the Exchange to fetch guarantees, that a certain transaction went through and that the reserve can be created and withdrawn. This will help C2EC to capture the transaction of the Terminal Backend to the Exchange's account and therefore allow the withdrawal by the customer. Therefore the wirewatch gateway API is implemented as part of C2EC. When the wirewatch gateway can get the proof, that a transaction was successfully processed, the exchange will create a reserve with the corresponding reserve public key.

For C2EC not all endpoints of the Wire Gateway API are needed. Therefore the endoints which are not needed will be implemented but always return http status code 400 with explanatory error details as specified by the specification.

GET - config

- **Method:** GET
- **Endpoint:** /config
- **Description:** Returns a `WireConfig` object with configuration information about the Wirewatch Gateway API of the C2EC component.
- **Response:** HTTP status code 200 OK. The exchange responds with a `C2ECConfig` object.
- **Consumers:** Components who want to use the API and therefore want to load the config of the instance.

POST - transfer

- **Method:** GET
- **Endpoint:** /transfer
- **Description:** Allows the *Exchange* to make a transaction. This API is used in case of a refund. The transfer will therefore pointed towards a `payto://wallee-transaction` address.
- **Request:** The request contains a `TransferRequest` object.
- **Response:** HTTP status code 200 OK. The exchange responds with a `C2ECConfig` object.
- **Consumers:** The *Exchange* who wants to transfer digital cash to a account which can be handled by the C2EC component.

GET - history of incoming transactions

- **Method:** GET
- **Endpoint:** /history/incoming
- **Description:** Returns a list of transactions which were recently created in the C2EC component. In case of C2EC, this are withdrawal operations which are confirmed and a reserve can therefore be created by the exchange.
- **Response:** HTTP status code 200 OK. The exchange responds with a `C2ECConfig` object.
- **Consumers:** The *Exchange* who will create the reserve which then can be withdrawn by the *Taler Wallet*.

3.1.6. The C2EC database

The database of the C2EC component must track two different aspects. The first is the mapping of a nonce (the `WOPID`) to a reserve public key to enable withdrawals and the second aspect is the authentication of terminals allowing withdrawals owned by terminal providers like *Wallee*.

Terminal Provider

Table in figure 3.2 describing providers of C2EC compliant terminals. The name of the provider is important, because it decides which flow shall be taken in order to attest the payment. For example will the name *Wallee* signal the terminal provider to trigger the attestation flow of *Wallee* once the payment notification for the withdrawal reaches C2EC.

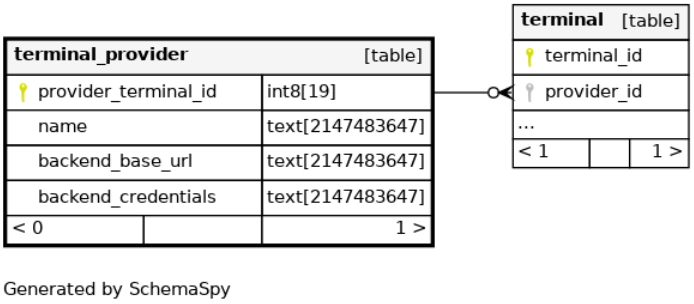

Figure 3.2.: Terminal Provider Table

Terminal

Table in figure 3.3 contains information about terminals of providers. This includes the provider they belong to and an authentication token, which is generated by the Exchange and allows authenticating the terminal. A terminal belongs to one terminal provider.

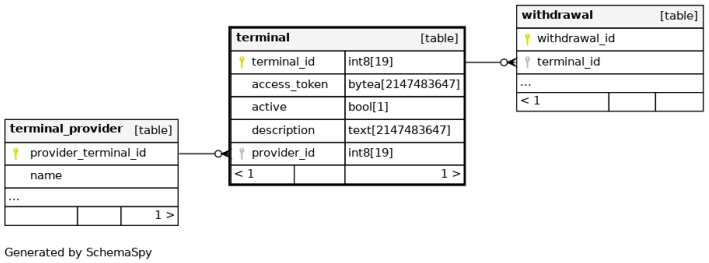

Figure 3.3.: Terminal Table

Withdrawal

Table in figure 3.4 represents the withdrawal processes initiated by terminals. This table therefore contains information about the withdrawal like the amount, which terminal the withdrawal was initiated from and which reserve public key is used to create a reserve in the Exchange.

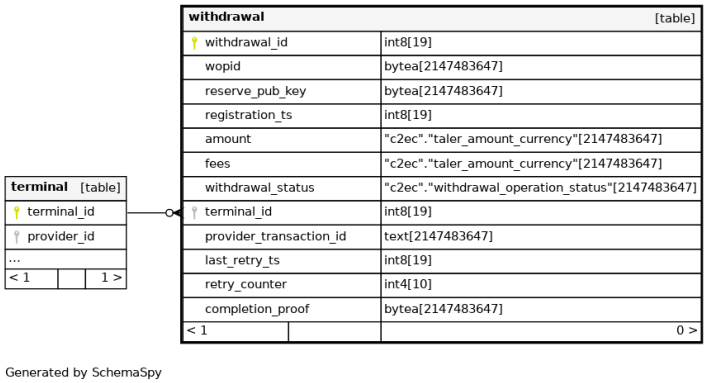

Figure 3.4.: Withdrawal Table

Relationships

The structure of the three tables forms a tree which is rooted at the terminal provider. Each provider can have many terminals and each terminal can have many withdrawals. The reverse does not apply. A withdrawal does always belong to exactly one terminal and a terminal is always linked to exactly one provider. These relations are installed by using foreign keys, which link the sub-entities (Terminal and Withdrawal) to their corresponding owners (Provider and Terminal). A provider owns its terminals and a terminal owns its Withdrawals.

3.2. Payto wallee-transaction extension

RFC 8905 [7] specifies a URI scheme (complying with RFC 3986 [8]), which allows to address a creditor with theoretically any protocol that can be used to pay someone (such as IBAN, BIC etc.) in a standardized way. Therefore it introduces a registry which holds the specific official values of the standard. The registry is supervised by the GANA (GNUnet Assigned Numbers Authority) [9].

In case a refund becomes necessary, which might occur if a credit card transaction does not succeed, a new *target type* called *wallee-transaction* is registered. It takes a transaction identifier as *target identifier* which identifies the transaction for which a refund process shall be triggered. The idea is that the handler of the

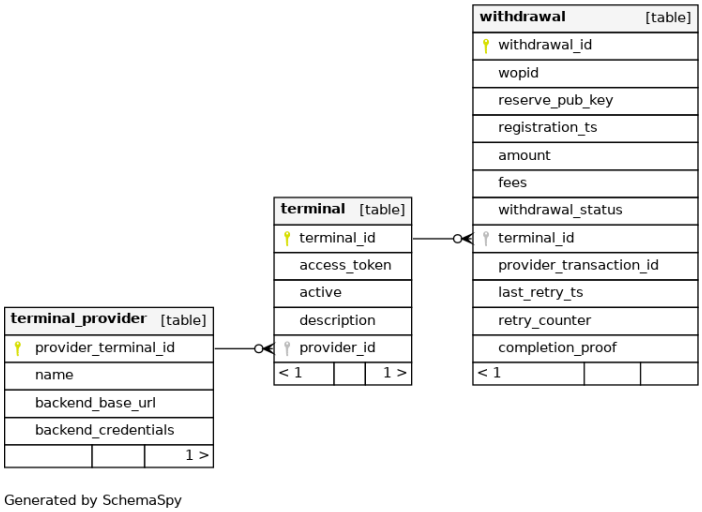

Figure 3.5.: Relationships of the entities.

payto URI is able to deduct the transaction from the payto-uri and trigger the refund process.

3.2.1. Payto refund using Wallee

Wallee allows to trigger refunds using the Refund Service of the Wallee backend. The service allows to trigger a refund given a transaction identifier. Therefore the C2EC component can trigger the refund using the refund service if needed, and the payto-uri retrieved as debit account by the wirewatch gateway API, is leveraged to delegate the refund process to the Wallee Backend using the Refund Service and parsing the transaction identifier of the payto-uri.

3.2.2. Extensibility

The flow is extensible and other providers like Wallee might be added. They must therefore register their own refund payto-uri with the GANA and then the refund process can be implemented likewise.

3.3. Taler Wallet

The *Taler Wallet* is responsible to create a reserve key pair which will allow him the withdrawal using the *Exchange* using the reserve public key of the key pair.

The reserve public key is created by the *Taler Wallet* and sent to C2EC to establish the mapping between the *wopid* and the reserve public key. The reserve public

key is used to eventually create a reserve at the exchange which contains the digital cash. The *Taler Wallet* can then withdraw the digital cash from this reserve using the withdrawal process of the wallet [10]. The process for the case of C2EC is slightly different from the present processes because the requests to the Bank-Integration API contain different properties than the currently supported. This means the *Taler Wallet* must be extended in order to allow the withdrawal using C2EC.

3.3.1. Taler Wallet Perspective

From the perspective of the Wallet, the system looks as follows:

- Uses the QR Code displayed on the *Wallee Terminal* to identify nonce and read exchange information.
- Uses the Bank-Integration API of *C2EC* to register the reserve public key and retrieve information about the confirmation of the withdrawal.
- Uses the *Exchange* to withdraw the digital cash.

3.4. Wallee

Wallee offers level 1 PCI-DSS [11] compliant payment processes to its customers [12] and allows an easy integration of its process into various kinds of merchant systems (e.g. websites, terminals, etc).

3.4.1. Wallee Perspective

From the perspective of Wallee, the system looks as follows:

- Uses the Bank-Integration API of *C2EC* to get notified about parameter selection and inform *C2EC* about the payment.
- Needs the credit card of the customer in order to execute the payment.
- Uses the *Wallee Backend* to execute the payment using the supplied Android Till SDK sous-sous-section 3.4.2

3.4.2. Wallee Terminal

Wallee Terminals are based on android and run a modified, certified android version as operating system. Thus they can be used for payments and establish strong authentication in a trusted way.

Withdrawal Operation Identifier

The **W**ithdrawal-**OP**eration-**ID**entifier (*wopid*) is leveraged by all components to establish the connection to an entry in the withdrawal table (figure 3.4) of C2EC. The *wopid* is therefore crucial and every participant of the withdrawal must eventually gain knowledge about the value of the *wopid* in order to process the withdrawal. The *wopid* is created by the Terminal and advertised to the Exchange by requesting notification, when the reserve public key belonging to the *wopid* was received and the mapping could be created. The Wallet gains the *wopid* value when scanning the QR code at the Terminal and then sends the *wopid* (and the other parameters) to the Exchange.

Creation of the WOPID

Besides the entropy needed to establish a correct *wopid*, the hash function leveraged must be specified. (TODO - e.g. FIPS 180-4 [13] (SHA-1 and SHA-2 families) or FIPS-202 [14] (SHA-3 family, which is still beeing reviewed))

Wallee Till API

Wallee supplies the Wallee Android Till SDK [15] which allows the implementation of custom application for their android based terminals. The API facilitates the integration with the Wallee backend and using it to create payments.

3.4.3. Wallee Backend and API

Terminals of Wallee are used to communicate with the customer at the shop of the merchant. The payment and processing of the transaction is run on the *Wallee Backend*. The *Wallee Backend* is used by C2EC to attest a payment, when a `C2ECPaymentNotification` message reaches C2EC. The *Wallee Backend* is also used in order to do refunds, in case something goes wrong during the payment. Therefore the API of *Wallee Backend* is used to collect this information or process a refund. Wallee structures its API using *Services*. For C2EC this means that the *Transaction Service* [16] and *Refund Service* [17] must be implemented.

Transaction Service

The *Transaction Service* is used by C2EC to attest a transaction was successfully processed and the reserve can be created by the *Exchange*. Therefore the `GET /api/transaction/read` API of the *Transaction Service* is used. If the returned transaction is in state *fulfill*, the transaction can be stored as *completion_proof* for the withdrawal as specified in the withdrawal table figure 3.4 and the withdrawal status can be transitioned to *confirmed*. This will tell the *Exchange* to create the reserve which can eventually be withdrawn by the wallet.

Refund Service

The *Refund Service* is used by C2EC in case of a refund. Therefore the C2EC gets notified by the *Exchange* that the transaction shall be refunded. To trigger the refund process at the Wallee backend, the `POST /api/refund/refund` is used.

Wallee Transaction State

In order to decide if a transaction was successful, the states of a transaction within Wallee must be mapped to the world of Taler. This means that a reserve shall only be created, if the transaction is in a state which allows Taler not having any liabilities regarding the transaction and that Wallee could process the payment successfully. The documentation states that *only* in the transaction state *fulfill*, the delivery of the goods (in case of withdrawal this means, that the reserve can be created) shall be started [18]. For the withdrawal this means, that the only interesting state for fulfillment is the *fulfill* state. Every other state means, that the transaction was not successful and the reserve shall not be created.

4. Implementation

4.1. Concepts

This chapter describes high level concepts which are used in the implementation of the components. The short explanations aim to support the understanding of the reader to faster and better understand the implementation of the components.

4.1.1. Consumers and Producers

The two terms consumer and producer are used through the entire documentation. They describe the role of a component in an interaction with another component. The consumer is the component askig or requesting a producer to gather information or trigger some action. The Producer on the other hand is the component who receives information or call for action of a consumer.

4.1.2. Long-Polling

Long-Polling is the concept of not closing a connection until a a response can be delivered or a given duration exceeds. This allows a consumer to ask for information which it assumes will arrive in the future at the producer. The producer therefore will not close the request of the consumer but instead keep the connection open and respond with the response, once it is available. The consumer and the producer can both close the connection after a certain amount of time, which is called the timeout. This can also happen if the wanted result of the producer cannot be returned to the consumer.

4.1.3. Publish-Subscribe Pattern

The concept of publishers and subscribers is used heavily in the implementation. It allows decoupling different steps of the process and allows different steps to be handled and executed in their own processes. Publishers can also be called notifiers or similar, while the subscriber can also be called listener or similar.

The communication of publishers and subscribers happends through channels. A publisher will publish to a certain channel when a defined state is reached. The

subscriber who is subscribed or listens to this channel will capture the message sent through the channel by the publisher and start processing it.

The publish-subscribe scheme enables loose coupling and therefore helps to improve the performance of individual processes, because they cannot be hindered by others.

4.1.4. Go Language

Go comes with handy features to implement the concepts like pub/sub sous-section 4.1.3 or long polling sous-section 4.1.2.

Contexts

Go standard library contians a package called *context*. You will stumble over this package all the time, even when using third party libraries or when writing your own code. The *context* package allows to control the lifetime and cancellation activities for function and allows concurrent running threads to be executed within the same context. For example if you have a function which can sort a list concurrently and will fail if any thread has a failure, supplying each concurrent execution of the function the same context allows to leave the function early if the context is left by propagating the done signal through the *Done* channel which is part of each context. A context also defines the *cancellation function*. This function shall be called, for each context when it becomes obsolete. The cancellation function will execute cleanup activities related to the specific context. It is a best practice to defer the cancellation function right after the creation of the context [19].

Go Routines

In concurrent programs it is a challenge to keep up with the complexity which they add to the code. Also one has to take care of interprocess communication and if memory is accessed in shared manner by the program, the access to the data stored should be mutual exclusive. Go therefore comes with the concept of Goroutines. They are designed to be very cheap, lightweight threads. They share the same address space and are just executed besides each other as simple functions. Also Go encourages the use of channels to communicate between different goroutines. The use of channels makes locking memory for concurrent access obsolete and therefore removes possible concurrency problems by making them impossible by design [20].

Coroutines

Coroutines are the coutnerpart to the *Go routines* in Kotlin and are leveraged in the development of the Terminal Application.

Memory safety

Even when Go is a low level language which compiles to native bytecode directly, it implements a garbage collector and memory management which takes care of allocating and releasing memory as needed. This reduces the risk of memory leaks to bugs in the memory management and the garbage collector.

4.2. Database

The Database is implemented using Postgresql. This database is also used by other Taler components and therefore is a good fit.

Besides the standard SQL features to insert and select data, Postgres also comes with handy features like LISTEN and NOTIFY.

This allows the implementation of neat pub/sub models allowing better performance and separation of concerns.

4.2.1. Schema

For the C2EC component the schema c2ec is created. It holds three tables, custom types and triggers.

4.2.2. Triggers

Triggers are used to decouple the different sub processes in the withdrawal flow from one another.

The trigger runs a Postgres function which will execute a NOTIFY statement using Postgres built-in function *pg_notify*, which wraps the statement in a Postgres function allowing to be used more easy.

4.3. C2EC

4.3.1. Decoupling steps

To decouple different steps in the withdrawal process an event based architecture is implemented. This means that every write action to the database will represent

an operation which will trigger an event. The applications processes are listening to those events. The consumer of the API can wait to be notified by the API, by registering to those events via a long polling request at the API. This long-polling will then wait until the listener receives the event and return the received event to the consumer.

Following a short list of events and from whom they are triggered and who listens to them:

- Registration of the Withdrawal Operation.
  - Registered by: Wallet
  - Listened by: Terminal
- Payment Confirmation sent to the Bank-Integration API of C2EC.
  - Registered by: Terminal
  - Listened by: Attestor
- Payment attestation success will send a withdrawal operation status update event.
  - Registered by: Attestor
  - Listened by: Consumers (via Bank-Integration-API)
- Payment attestation failure will trigger a retry event.
  - Registered by: Attestor
  - Listened by: Retrier
- Transfers which represent refunds in C2EC.
  - Registered by: Exchange (through the wire gateway API)
  - Listened by: Transfer

4.3.2. Bank-Integration API

The Bank Integration API was implemented according to the specification [5]. It only implements messages and API specific to the indirect withdrawal operation.

Namely this are the following endpoints:

- GET /config
- GET /withdrawal-operation/[WOPID]
- POST /withdrawal-operation/[WOPID]
- POST /withdrawal-operation/[WOPID]/payment

- POST /withdrawal-operation/[WOPID]/abort

4.3.3. Wire-Gateway API

The Wire-Gateway API delivers the transaction history to the exchange which will create reserves for the specific public keys and therefore allow the customers to finally withdraw the digital cash using their wallet.

Following endpoints are implemented by the wire gateway API implementation:

- GET /config
- POST /transfer
- GET /history/incoming
- GET /history/outgoing

Keeping track of transfers

The Wire-Gateway specification requires the implementor of the API to keep track of incoming transfer requests in order to guarantee the idempotence of the API. Therefore the implementation keeps track of all transfers in the database table *transfers*. It stores the transfer data in the database. If a request with the same UID is sent to the transfer-API, first it is checked that the incoming request is exactly the same as the previous one by comparing the request to the values stored in the database. Only if the hashes are the same, the transfer request is processed further. Otherwise the API responds with a conflict response.

4.3.4. Payment Attestation

The attestation of a transaction is crucial, since this is the action which allows the exchange to create a reserve and can proof to the provider and customer, that the transaction was successful and therefore can put the liability for the money on the provider. The attestation process is implemented using a provider client interface and a provider transaction interface. This allows the process to be the same for each individual provider and new providers can be added easily by providing a specific implementation of the interfaces.

Provider Client

The provider client interface is called by the attestation process depending on the notification received by the database upon receiving a payment notification of the provider's terminal. The specific provider clients are registered at the startup of

the component and the attestation process will delegate the information gathering to the specific client, based on the notification received by the database.

The provider client interface defines three functions:

1. SetupClient: The setup function is called by the startup of the application and used to initialize the client. Here it makes sense to check that everything needed for the specific client is in place and that properties like access credentials are available.

2. GetTransaction: This function is used by the attestation process to retrieve the transaction of the provider system. It takes the transaction identifier supplied with the payment notification message and loads the information about the transaction. Based on this information the decision to confirm or abort the transaction is done.

3. Refund: Since the transaction of the money itself is done by the provider, also refunds will be unwind by the provider. This functions mean is to trigger this refund transaction at the provider.

Provider Transaction

Since the attestation process is implemented to support any provider, also the transaction received by the provider clients *GetTransaction* function is abstracted using an interface. This interface must be implemented by any provider transaction which belongs to a specific provider client.

The provider client interface defines following functions:

1. AllowWithdrawal: This function shall return true, when the transaction received by the provider enters a positive final state. This means that the provider accepted the transaction and could process it.

2. AbortWithdrawal: It doesn't mean that if a transaction does not allow to do the withdrawal, that the transaction shall be cancelled immediately. It could also be that the transaction was not yet processed by the provider. In this case we need means to check if the provider transaction is in an abort state if it is not ready for withdrawal, before aborting it. AbortWithdrawal shall therefore answer the question if the provider transaction is in a negative final state, which means the transaction is to be aborted.

3. Bytes: This function shall return a byte level representation of the transaction which will be used as proof of the transaction and stored in the exchanges database.

Retries

If the attestation fails, but the transaction is not in the refund state as specified by the provider's transaction, the problem could simply be that the service was not available or the transaction was not yet processed by the provider's backend. In order to not need to abort the transaction directly and give the system some robustness, a retry mechanism was implemented which allows retrying the attestation step.

The retry will only be executed, when the transaction attestation failed because the transaction was not in the abort state or if for some reason the transaction information could not have been retrieved.

4.3.5. Wallee Client

The Wallee client is the first implementation of the provider client interface and allows the attestation of transactions using the Wallee backend system. The backend of Wallee provides a ReST-API to their customers, which allows them to request information about payments, refunds and so on. To access the API, the consumer must authenticate themself to Wallee by using their own authentication token as explained in sous-sous-section 4.3.6.

As indicated by the provider client interface, we will use two API of the Wallee backend:

- Transaction service: The transaction service aims to provide information about a transaction registered using a Wallee terminal.
- Refund service: The refund service allows to trigger a refund for a given transaction using the transaction identifier. The refund will then be executed by the Wallee backend, back to the Customer.

4.3.6. Security

Authenticating at the Wallee ReST API

The Wallee API specifies four Wallee specific headers which are used to authenticate against the API. It defines its own authentication standard and flow. The flow builds on a MAC (message authentication code) which is built on a version, user identifier, and a timestamp. For the creation of the MAC the HMAC (hash based message authentication code) SHA-512 is leveraged which takes the so called *application-user-key* (which is basically just an access-token, which the user receives when creating a new API user) as key and the above mentioned properties plus information about the requested http method and the exactly requested path (including request parameters) as message [21]. The format of the message is specified like:

```
Version|User-Id|Unix-Timestamp|Http-Method|Path
```

- Version: The version of the algorithm
- User-Id: The user-id of the requesting user
- Unix-Timestamp: A unix timestamp (seconds since 01.01.1970)
- Http-Method: one of `HEAD`, `GET`, `POST`, `PUT`, `DELETE`, `TRACE`, `CONNECT`
- Path: The path of the requested URL including the query string (if any)

The resulting string must then be UTF-8 encoded according to RFC-3629 [22].

API access

Bank-Integration API

The Bank-Integration API is accessed by Wallets and Terminals. This results in two different device types for the autentication procedure. The Wallet should be able to authenticate against the exchange by using an access token according to the specified authentication flow of the core bank API [23] which leverages a bearer token as specified by DD-49 [24]. For terminals the authentication mechanism is based on a basic auth scheme as specified by RFC-7617 [25]. Therefore a generated access-token used as password and a username which is generated registering the terminal using the cli explained in sous-sous-section 4.3.6 are leveraged.

Wire-Gateway API

The wire gateway specifies a basic authentication scheme [26] as described in RFC-7617 [25]. Therefore the C2EC component allows the configuration of a username and password for the exchange. During the request of the exchange at the wire gateway API, the credentials are checked.

Registering Providers and Terminals

A provider may want to register a new Terminal or maybe even a new provider shall be registered for the exchange. To make this step easier for the exchange operators, a simple cli program (command line interface) was implemented. The cli will either ask for a password or generate an access token in case of the terminal registration. The credentials are stored has hashes using a PBKDF (password based key derivation function) so that even if the database leaks, the credentials cannot be easily read by an attacker.

Deactivating Terminals

A Terminal can be stolen, hijacked or hacked by malicious actors. Therefore it must be possible to disable a terminal immediately and no longer allow withdrawals using this terminal. Therefore the *active* flag can be set to *false* for a registered terminal. The Bank-Integration API which processes withdrawals and authenticates terminals, must check that the requesting terminal is active and is allowed to initiate withdrawals. Since the check for the *active* flag must be done for each request of a terminal, the check can be centralized and is implemented as part of the authentication flow. A Wallee terminal can be deactivated using the cli mentioned in sous-sous-section 4.3.6.

4.4. Wallee POS Terminal

4.5. Wallet

5. Results

5.1. Discussion

What is the significance of your results? – the final major section of text in the paper. The Discussion commonly features a summary of the results that were obtained in the study, describes how those results address the topic under investigation and/or the issues that the research was designed to address, and may expand upon the implications of those findings. Limitations and directions for future research are also commonly addressed.

5.2. Results

What did you find? – a section which describes the data that was collected and the results of any statistical tests that were performed. It may also be prefaced by a description of the analysis procedure that was used. If there were multiple experiments, then each experiment may require a separate Results section.

Declaration of Authorship

I hereby declare that I have written this thesis independently and have not used any sources or aids other than those acknowledged.

All statements taken from other writings, either literally or in essence, have been marked as such.

I hereby agree that the present work may be reviewed in electronic form using appropriate software.

April 21, 2024

J. Häberli

Bibliography

[1] Fabio Panetta. A digital euro that serves the needs of the public: striking the right balance. https://www.ecb.europa.eu/press/key/date/2022/html/ecb.sp220330_1~f9fa9a6137.en.html, March 2022.

[2] on behalf of ECB Kantar Public (Verian since November 2023). Study on new digital payment methods. https://www.ecb.europa.eu/euro/digital_euro/investigation/profuse/shared/files/dedocs/ecb.dedocs220330_report.en.pdf, March 2022.

[3] Wallee. Payment connectors. https://app-wallee.com/connectors.

[4] Taler. Taler wire gateway http api. https://docs.taler.net/core/api-bank-wire.html.

[5] Taler. Taler bank integration api. https://docs.taler.net/core/api-bank-integration.html.

[6] Mark Nottingham. The "secret-token" URI Scheme. RFC 8959, January 2021.

[7] Florian Dold and Christian Grothoff. The 'payto' URI Scheme for Payments. RFC 8905, October 2020.

[8] Tim Berners-Lee, Roy T. Fielding, and Larry M Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, January 2005.

[9] GNUnet Project. The gnunet assigned numbers authority (gana). https://gana.gnunet.org/.

[10] Taler. Withdrawal. https://docs.taler.net/taler-wallet.html#withdrawal.

[11] PCI Security Standards Council. Pci data security standard. https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0.pdf.

[12] Wallee. Transaction states. https://app-wallee.com/de-de/doc/payment.

[13] Quynh Dang. Secure hash standard, 2015-08-04 2015.

[14] National Institute of Standards and Technology. Sha-3 standard: Permutation-based hash and extendable-output functions. https://doi.org/10.6028/NIST.FIPS.202.

[15] Wallee. Wallee android till sdk. https://github.com/wallee-payment/android-till-sdk.

[16] Wallee. Transaction service. https://app-wallee.com/de-de/doc/api/web-service#transaction-service.

[17] Wallee. Refund service. https://app-wallee.com/de-de/doc/api/web-service#refund-service.

[18] Wallee. Transaction states. https://app-wallee.com/de-de/doc/payment/transaction-process.

[19] Matt T. Proud Jean de Klerk. Contexts and structs. https://go.dev/blog/context-and-structs, February 2021.

[20] Go. Share by communicating. https://go.dev/doc/effective_go#sharing.

[21] Wallee. Authentication. https://app-wallee.com/en-us/doc/api/web-service#_authentication.

[22] François Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629, November 2003.

[23] Taler. Authentication. https://docs.taler.net/core/api-corebank.html#authentication.

[24] Taler. Authentication. https://docs.taler.net/design-documents/049-auth.html.

[25] Julian Reschke. The 'Basic' HTTP Authentication Scheme. RFC 7617, September 2015.

[26] Taler. Taler wire gateway http api. https://docs.taler.net/core/api-bank-wire.html#authentication.

List of Figures

2.1. Involved components and devices 5
2.2. Diagram of included components and their interactions 6
2.3. Process of a withdrawal using a credit card 7

3.1. Withdrawal Operation state transition diagram 12
3.2. Terminal Provider Table 16
3.3. Terminal Table 16
3.4. Withdrawal Table 17
3.5. Relationships of the entities. 18

List of Tables

Listings

A.1. C2EC API specification 47

Glossary

This document is incomplete. The external file associated with the glossary 'main' (which should be called `thesis.gls`) hasn't been created.

Check the contents of the file `thesis.glo`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

If you don't want this glossary, add `nomain` to your package option list when you load `glossaries-extra.sty`. For example:

```
\usepackage[nomain]{glossaries-extra}
```

Try one of the following:

- Add `automake` to your package option list when you load `glossaries-extra.sty`. For example:

  ```
  \usepackage[automake]{glossaries-extra}
  ```

- Run the external (Lua) application:

  ```
  makeglossaries-lite.lua "thesis"
  ```

- Run the external (Perl) application:

  ```
  makeglossaries "thesis"
  ```

Then rerun LaTeX on this document.

This message will be removed once the problem has been fixed.

A. Appendix A

A.1. API

```
..
  This file is part of GNU TALER.

  Copyright (C) 2014-2024 Taler Systems SA

  TALER is free software; you can redistribute it and/or modify it under the
  terms of the GNU Affero General Public License as published by the Free Software
  Foundation; either version 2.1, or (at your option) any later version.

  TALER is distributed in the hope that it will be useful, but WITHOUT ANY
  WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR
  A PARTICULAR PURPOSE.  See the GNU Affero General Public License for more details.

  You should have received a copy of the GNU Affero General Public License along with
  TALER; see the file COPYING.  If not, see <http://www.gnu.org/licenses/>

  @author Joel Haeberli

============================
The C2EC RESTful API
============================

.. note::

  **This API is experimental and not yet implemented**

```

```
27 This chapter describe the APIs that third party providers need to
       integrate to allow
28 withdrawals through indirect payment channels like credit cards or
       ATM.
29
30 .. contents:: Table of Contents
31
32 --------------
33 Authentication
34 --------------
35
36 Terminals which authenticate against the C2EC API must provide
       their respective
37 access token. Therefore they provide a ‘‘Authorization: Bearer
       $ACCESS_TOKEN‘‘ header,
38 where ‘$ACCESS_TOKEN‘‘ is a secret authentication token configured
       by the exchange and
39 must begin with the RFC 8959 prefix.
40
41 ----------------------------
42 Configuration of C2EC
43 ----------------------------
44
45 .. http:get:: /config
46
47   Return the protocol version and configuration information about
         the C2EC API.
48
49   **Response:**
50
51   :http:statuscode:‘200 OK‘:
52     The exchange responds with a ‘C2ECConfig‘ object. This request
         should
53     virtually always be successful.
54
55   **Details:**
56
57   .. ts:def:: C2ECConfig
58
59     interface C2ECConfig {
60       // Name of the API.
61       name: "taler-c2ec";
62
```

```
63        // libtool-style representation of the C2EC protocol
              version, see
64        //
              https://www.gnu.org/software/libtool/manual/html_node/Versioning.html#Versioni
65        // The format is "current:revision:age".
66        version: string;
67      }
68
69  ----------------------------
70  Withdrawing using C2EC
71  ----------------------------
72
73  Withdrawals with a C2EC are based on withdrawal operations which
      register a withdrawal identifier
74  (nonce) at the C2EC component. The provider must first create a
      unique identifier for the withdrawal
75  operation (the ``WOPID``) to interact with the withdrawal
      operation and eventually withdraw using the wallet.
76
77  .. http:post:: /withdrawal-operation
78
79    Register a `WOPID` belonging to a reserve public key.
80
81    **Request:**
82
83    .. ts:def:: C2ECWithdrawRegistration
84
85      interface C2ECWithdrawRegistration {
86        // Maps a nonce generated by the provider to a reserve
              public key generated by the wallet.
87        wopid: ShortHashCode;
88
89        // Reserve public key generated by the wallet.
90        // According to TALER_ReservePublicKeyP
              (https://docs.taler.net/core/api-common.html#cryptographic-primitives)
91        reserve_pub_key: EddsaPublicKey;
92
93        // Optional amount for the withdrawal.
94        amount?: Amount;
95
96        // Id of the terminal of the provider requesting a
              withdrawal by nonce.
97        // Assigned by the exchange.
98        terminal_id: SafeUint64;
```

```
99      }
100
101   **Response:**
102
103   :http:statuscode:`204 No content`:
104     The withdrawal was successfully registered.
105   :http:statuscode:`400 Bad request`:
106     The ``WithdrawRegistration`` request was malformed or
          contained invalid parameters.
107   :http:statuscode:`500 Internal Server error`:
108     The registration of the withdrawal failed due to server side
          issues.
109
110 .. http:get:: /withdrawal-operation/$WOPID
111
112   Query information about a withdrawal operation, identified by
        the ``WOPID``.
113
114   **Request:**
115
116   :query long_poll_ms:
117     *Optional.*  If specified, the bank will wait up to
          ``long_poll_ms``
118     milliseconds for operationt state to be different from
          ``old_state`` before sending the HTTP
119     response.  A client must never rely on this behavior, as the
          bank may
120     return a response immediately.
121   :query old_state:
122     *Optional.* Default to "pending".
123
124   **Response:**
125
126   :http:statuscode:`200 Ok`:
127     The withdrawal was found and is returned in the response body
          as ``C2ECWithdrawalStatus``.
128   :http:statuscode:`404 Not found`:
129     C2EC does not have a withdrawal registered with the specified
          ``WOPID``.
130
131   **Details**
132
133     .. ts:def:: C2ECWithdrawalStatus
134
```

```
135       interface C2ECWithdrawalStatus {
136         // Current status of the operation
137         // pending: the operation is pending parameters selection
              (exchange and reserve public key)
138         // selected: the operations has been selected and is
              pending confirmation
139         // aborted: the operation has been aborted
140         // confirmed: the transfer has been confirmed and
              registered by the bank
141         // Since protocol v1.
142         status: "pending" | "selected" | "aborted" | "confirmed";
143
144         // Amount that will be withdrawn with this operation
145         // (raw amount without fee considerations).
146         amount: Amount;
147
148         // A refund address as ``payto`` URI. This address shall
              be used
149         // in case a refund must be done. Only not-null if the
              status
150         // is "confirmed" or "aborted"
151         sender_wire?: string;
152
153         // Reserve public key selected by the exchange,
154         // only non-null if ``status`` is ``selected`` or
              ``confirmed``.
155         // Since protocol v1.
156         selected_reserve_pub?: string;
157       }
158
159
160 .. http:post:: /withdrawal-operation/$WOPID
161
162     Notifies C2EC about an executed payment for a specific
          withdrawal.
163
164   **Request:**
165
166   .. ts:def:: C2ECPaymentNotification
167
168     interface C2ECPaymentNotification {
169
170       // Unique identifier of the provider transaction.
171       provider_transaction_id: string;
```

```
172
173       // Specifies the amount which was payed to the provider
              (without fees).
174       // This amount shall be put into the reserve linked to by
              the withdrawal id.
175       amount: Amount;
176
177       // Fees associated with the payment.
178       fees: Amount;
179     }
180
181   **Response:**
182
183   :http:statuscode:‘204 No content‘:
184     C2EC received the ‘‘C2ECPaymentNotification‘‘ successfully and
          will further process
185     the withdrawal.
186   :http:statuscode:‘400 Bad request‘:
187     The ‘‘C2ECPaymentNotification‘‘ request was malformed or
          contained invalid parameters.
188   :http:statuscode:‘404 Not found‘:
189     C2EC does not have a withdrawal registered with the specified
          ‘‘WOPID‘‘.
190   :http:statuscode:‘500 Internal Server error‘:
191     The ‘‘C2ECPaymentNotification‘‘ could not be processed due to
          server side issues.
192
193
194 --------------
195 Taler Wire Gateway
196 --------------
197
198 C2EC implements the wire gateway API in order to check for
      incoming transactions and
199 let the exchange get proofs of payments. This will allow the C2EC
      componente to add reserves
200 and therefore allow the withdrawal of the digital cash. C2EC does
      not entirely implement all endpoints,
201 because the it is not needed for the case of C2EC. The endpoints
      not implemented are not described
202 further. They will be available but respond with 400 http error
      code.
203
204 .. http:get:: /config
```

```
205
206   Return the protocol version and configuration information about
          the bank.
207   This specification corresponds to ``current`` protocol being
          version **0**.
208
209   **Response:**
210
211   :http:statuscode:`200 OK`:
212     The exchange responds with a `WireConfig` object. This request
            should
213     virtually always be successful.
214
215   **Details:**
216
217   .. ts:def:: WireConfig
218
219     interface WireConfig {
220       // Name of the API.
221       name: "taler-wire-gateway";
222
223       // libtool-style representation of the Bank protocol
              version, see
224       //
              https://www.gnu.org/software/libtool/manual/html_node/Versioning.html#Versioni
225       // The format is "current:revision:age".
226       version: string;
227
228       // Currency used by this gateway.
229       currency: string;
230
231       // URN of the implementation (needed to interpret 'revision'
              in version).
232       // @since v0, may become mandatory in the future.
233       implementation?: string;
234     }
235
236 .. http:post:: /transfer
237
238   This API allows the exchange to make a transaction, typically to
          a merchant.  The bank account
239   of the exchange is not included in the request, but instead
          derived from the user name in the
240   authentication header and/or the request base URL.
```

```
241
242 To make the API idempotent, the client must include a nonce.
        Requests with the same nonce
243 are rejected unless the request is the same.
244
245 **Request:**
246
247 .. ts:def:: TransferRequest
248
249   interface TransferRequest {
250     // Nonce to make the request idempotent.  Requests with the
          same
251     // ``request_uid`` that differ in any of the other fields
252     // are rejected.
253     request_uid: HashCode;
254
255     // Amount to transfer.
256     amount: Amount;
257
258     // Base URL of the exchange.  Shall be included by the bank
          gateway
259     // in the appropriate section of the wire transfer details.
260     exchange_base_url: string;
261
262     // Wire transfer identifier chosen by the exchange,
263     // used by the merchant to identify the Taler order(s)
264     // associated with this wire transfer.
265     wtid: ShortHashCode;
266
267     // The recipient's account identifier as a payto URI.
268     credit_account: string;
269   }
270
271 **Response:**
272
273 :http:statuscode:`200 OK`:
274   The request has been correctly handled, so the funds have been
        transferred to
275   the recipient's account.  The body is a `TransferResponse`.
276 :http:statuscode:`400 Bad request`:
277   Request malformed. The bank replies with an `ErrorDetail`
        object.
278 :http:statuscode:`401 Unauthorized`:
279   Authentication failed, likely the credentials are wrong.
```

```
280   :http:statuscode:`404 Not found`:
281     The endpoint is wrong or the user name is unknown. The bank
          replies with an `ErrorDetail` object.
282   :http:statuscode:`409 Conflict`:
283     A transaction with the same ``request_uid`` but different
          transaction details
284     has been submitted before.
285
286   **Details:**
287
288   .. ts:def:: TransferResponse
289
290     interface TransferResponse {
291       // Timestamp that indicates when the wire transfer will be
            executed.
292       // In cases where the wire transfer gateway is unable to
            know when
293       // the wire transfer will be executed, the time at which the
            request
294       // has been received and stored will be returned.
295       // The purpose of this field is for debugging (humans trying
            to find
296       // the transaction) as well as for taxation (determining
            which
297       // time period a transaction belongs to).
298       timestamp: Timestamp;
299
300       // Opaque ID of the transaction that the bank has made.
301       row_id: SafeUint64;
302     }
303
304 .. http:get:: /history/incoming
305
306   **Request:**
307
308   :query start: *Optional.*
309     Row identifier to explicitly set the *starting point* of the
          query.
310   :query delta:
311     The *delta* value that determines the range of the query.
312   :query long_poll_ms: *Optional.*  If this parameter is specified
        and the
313     result of the query would be empty, the bank will wait up to
          ``long_poll_ms``
```

```
314     milliseconds for new transactions that match the query to
            arrive and only
315     then send the HTTP response.  A client must never rely on this
            behavior, as
316     the bank may return a response immediately or after waiting
            only a fraction
317     of ``long_poll_ms``.
318
319   **Response:**
320
321   .. ts:def:: IncomingReserveTransaction
322
323     interface IncomingReserveTransaction {
324       type: "RESERVE";
325
326       // Opaque identifier of the returned record.
327       row_id: SafeUint64;
328
329       // Date of the transaction.
330       date: Timestamp;
331
332       // Amount transferred.
333       amount: Amount;
334
335       // Payto URI to identify the sender of funds.
336       debit_account: string;
337
338       // The reserve public key extracted from the transaction
              details.
339       reserve_pub: EddsaPublicKey;
340
341     }
```

Listing A.1: C2EC API specification

B. Appendix B

B.1. Meeting notes

17.01.2024

Participants

- Fehrensen Benjamin
- Grothoff Christian
- Häberli Joel

Topics

- Kickoff
- Understanding the Task
- Device
- Taler

Questions

- What am I going to do?
- Which components are roughly involved?

Action points

- Setup Thesis Document
- GNU Taler Copyright Assignment
- SSH-Public Key for git
- Inspect taler-exchange-wirewatch

Decisions

- Implement process 'cashless2ecash' as part of Taler-Exchange
- Wallet initializes process by scanning QR code like in the 'cash2ecash' showcase

- cash2ecash was implented by the guy named "windfisch" on mattermost

20.02.2024

Participants

- Jung Florian
- Häberli Joel

Topics

- Introduce each other and explain ideas
- Discuss nonce2ecash draft
- Discuss who wants to do what

Action points

- I send Flo a plan of what I'm going to do until when (approximately)
- I update the sequence diagram as discussed and send the openapi spec to Flo for review.

Decisions

- We can establish a generic approach for both our cases. Therefore the abstraction of *Providers* will be implemented. The *Providers* abstract and generalize some endpoint which can accept digital cash in any form (Credit Card, Cash, and so on) and give the Exchange the guarantee, that the digital cash will eventually be transferred to the Exchange.
- The verification at the provider from the perspective of the exchange must be optional (withdrawing at an ATM will not get any better than the amount the ATM sends to the Excahnge in the payment notification). Therefore an additional request to the provider will not bring any benefit.

Notes

- Flo wants to create a Reserve containing digital cash from the ATM. He then wants to trigger a peer to peer transaction. And therefore this reserve deals as guarantee to the Exchange. This flow is possible if the provider is controlled, which in my case is not given (Wallee is a company and I cannot easily alter their source code to open a reserve)

22.02.2024

Participants

- Hiltgen Alain
- Fehrensen Benjamin
- Grothoff Christian
- Häberli Joel

Topics

- Task description
- Deeper understanding of the topic established?
- I contacted Florian Jung (alias Windfisch) and we bespoke his work on cash2ecash.

Questions

- Repository of Wallee Application will be different than 'cashless2ecash'? No
- Wallee: Master Password? Password received by Ben
- Wallee: Which SDK to use? Till-SDK (API to Wallee-Backend)
- How do we want to handle different currencies? How about currencies like Bitcoin? Currency is determined by the currency of the exchange.

06.03.2024

Participants

- Fehrensen Benjamin
- Grothoff Christian
- Häberli Joel

Topics

- API Spec nonce2ecash
- Database Spec nonce2ecash

Questions

- How can I create a reserve from the mapping table?
- Taler / Wallee : Which nonce to use? How to generate the nonce? Is there a preferred kind to generate nonces within taler?
- Do we add a maximal limit amount for a withdrawal on the side of the Taler Exchange?

Action points

- write API specification in .rst format (see /docs/core/api-*.rst in taler docs git)
- use Bank integration API
- write SQL schema and generate UML using schema-spy instead of writing UML.

13.03.2024

Participants

- Fehrensen Benjamin
- Grothoff Christian
- Häberli Joel

Topics

- SQL Schema of nonce2ecash.

Action points

- Add rst file to official docs Repository
- Add proper versioning to the SQL script.

20.03.2024

Participants

- Fehrensen Benjamin
- Grothoff Christian
- Häberli Joel

Topics

- Payto Specification.

Action points

- Specify payto-uri scheme in GANA repo

20.03.2024 - 2

Participants

- Grothoff Christian

- Häberli Joel

Topics

- Architecture
- Payto

Action points

- Look at Wire Gateway and Bank Integration API as specification of an API and not as individual components of Taler. C2EC must implement those specification in order to integrate into the Taler ecosystem.

27.03.2024

Participants

- Fehrensen Benjamin
- Grothoff Christian
- Häberli Joel

Topics

- Discussion of the Architecture documentation
- Feedback of Ben and Christian

Action points

- Integrate Feedback into documentation
- Use official docs repo to specify the API (e.g. Bank-Integration API and Wire Gateway API specification)
- No meeting next week.

10.04.2024

Participants

- Fehrensen Benjamin
- Grothoff Christian
- Häberli Joel

Topics

- Discussion of the C2EC code.

Action points

- Use ini-format to parse config
- Add support for PGHOST environment variable
- Rename config properties to be compliant with other Taler repositories.
  - serve
  - bind
  - unix-path-mode
  - etc.
- For the attestation there is the additional case that neither confirm nor abort is an option and instead retries are required.
- Remove doubled abstractions (Abstracting attestation is not necessary)

17.04.2024

Participants

- Hiltgen Alain
- Fehrensen Benjamin
- Grothoff Christian
- Häberli Joel

Topics

- Midterm Meeting with Expert Alain Hitlgen.
- Sequence diagram

Action points

- Fix Bank-Integration API
- Fees must be shown during the payment on the terminal
- The Wire Gateway API must implement "/history/outgoing" and return entries of the transfer table.

TEMPLATE

Participants

- Fehrensen Benjamin

- Grothoff Christian
- Häberli Joel

Topics

-

Questions

-

Action points

-

Decisions

-