Money Transfers

Send money to cards and accounts. P2P, P2B, Card2Card, Payouts and more.

Introduction

Quicko Money Transfer Hub supports card to card, card to account and other types of transfers. The hub is integrated with various acquirers, banks and money transfer operators. It can initiate Moneysend, Mastercard Send and VISA Direct transactions. Customer can recommend the Acquirer through whom he wants to process transactions. Note that if the Customer requires the settlement of the transaction by a new Acquirer, Quicko must create a new integration, which increases the required development. The specification of the new Acquirer should be provided by the Customer.

How to connect with us?

Quicko provides access to the solution in various ways, depending on the Customer requirements. This means that the Customer can access the Money Transfer Hub solution in three ways: ready-made application, mobile SDK, REST API.

White Label Application

This is a fully ready to use solution developed in Android and iOS and prepared UX. To see the appearance and the processes of the application from the point of view of the end user, please check this link.

Mobile SDK

Verestro provides Software Development Kit (SDK) which can be used for mobile money transfer. As a company Quicko provides many products which can be used in single application. For that reason Wallet SDK is divided into separated modules which covers different functionalities. Verestro team actively supports Customer with integration. More information about each module in Mobile SDK can be found in iOS Transfers Documentation, iOS Receivers Documentation for iOS implementation and Android Transfer Documentation, Android Receivers Documentation for Android implementation. If the Customer wants to integrate through Verestro SDK, it is required to create a customer account at Verestro Artifactory. To create such account, please contact the Customer Service.

REST API

Quicko provides Money Transfer API which is implemented according to the REST model. This API offers methods that allow not only to transfer money, but also to calculate commissions, collect available currencies for a given balance or authenticate the user and his card. Verestro team actively supports Customer with integration. More informations about Money Transfer API can be found here.

Overview

This section provides general information about the solution, terminology description and a high-level description of the business and technical of the Money Transfer Hub.

Abbreviations

Abbreviations used in the document:

Abbreviations	Description
ACQ	Acquiring Institution/Acquirer
AP	Admin Panel
ACS	Access Control Server
C2C	Card to card
DC	Data Core API
P2P	Peer to peer API
MDC	Mobile Data Core API
SDK	Software Development Kit
THC	Transaction History Core
OS	Operative System
IBAN	Bank Account Number
MCS20	Mastercard Send 2.0
URI	Uniform Resource Identifier
Mid	Merchant Identifier

Terminology

This section explains a number of key terms and concepts used in this document:

Name	Description

Customer	Institution which is using Verestro products. This institution decides which SDK should be used and how transaction should be processed. Basicly Customer can be called Verestro client.
User	User which is using Money Transfer Hub Application. It is root of entity tree. User is identified in Wallet Server by some unique identifier which is provided after registration. User can have access to his data and operations based on session. User's session is created after device pairing is performed. When session expires then user authentication have to be performed. Session is valid 10 minutes, however it is configurable parameter.
Card	Card belongs to the user. User can have many cards. Card is identified via internal id given after storing card on Wallet Server. Whole PAN is stored on Wallet Server which has PCI DSS certificate.
Device	Device belongs to user. When user starts using application after installation then device pairing is performed. After pairing device with some unique id, unique device installation id is generated and this installation is assigned to user. It is possible to have one active installation on specific device for specific user.
Session Token	Token which defines User. It is an authorization way of the User. This entity is created after paring device and this is needed to perform any actions in the application. When session is expired then user authentication needs to be performed. Session is valid 10 minute s, however it is configurable parameter.
Sender	Verestro Wallet user which triggers transaction to the Receiver (check User description).
Receiver	Entity which gets funds sent by Sender. Receiver can be identified in Wallet Server (Internal) or may be an entity that does not exist in Wallet Server (External) ◦ Internal (Wallet) – this type of Receiver has his own unique identifier just like sender. It can also act as a Sender in the transaction process, ◦ External – this type of Receiver does not exist in Wallet Server. Transfers that are made to this type of Receiver require the entering of his card data by Sender
Mid	Merchant identifier. This entity is representing Merchant in Acquirer's system. Customer have to provide the mid information to enable mid configuration in the Verestro system. Required to process 3DS authentication via Verestro System.
Acquirer	External institution responsible for processing transaction and 3ds requests ordered by the Verestro Money Transfer Hub Application. Acquirer connects with banks / card issuers and returns information whether the ordered action on a given card is possible.

PAN	It is 7-15 digits of credit card number. These digits contain the Permanent Account Number (PAN) assigned by the bank to uniquely identify the account holder.
Wallet Server	Provides the backend services to support Mobile Payment Application via Verestro Wallet SDK and is responsible for managing users, devices, cards , device tokens, storing transactions history and communication with Acquirers.
PCI DSS	PCI DSS (Payment Card Industry Data Security Standard) is a security standard used in environments where the data of payment cardholders is processed. The standard covers meticulous data processing control and protection of users against violations.
IBAN	IBAN (International Bank Account Number) is an international standard for bank account numbering that allows you to transfer funds to foreign accounts and to receive transfers from foreign entities to domestic bank accounts. One of the assumptions of the IBAN standard is to simplify the system of cross-border transfers.
QR	QR code is a type of barcode or scannable pattern that contains various forms of data like website links, account information, phone numbers, or even entire object of the transaction.

Card to card transaction

Card to Card Payments (C2C) is a technology which enables peer to peer transfer via mobile device. This functionality allows banks and payment institutions to perform transactions from Sender card to the Receiver card. In the first setting, the settlement agent performs Funding, in the next, it pushes funds by making Payment. In this type of transaction, Funding is possible only for users strongly verified using the 3DS Authentication method, which is described later in the document and it is an individual requirement depending on the Settlement Agent.

Card to card transaction high level overview

This diagram shows high level components which are involved in whole solution.

image-1658903579159.png

Card to card transaction key components

Component	**Description**

Verestro Wallet Server	Backend services of Money Transfer solution. In the described product, they are responsible for adding / keeping the context of the user and the card in the database, user authentication, calculate commissions, forwarding a transaction and 3ds authentication requests to various Acquirers or keeping transaction history.
Verestro P2P Wallet SDK	A set of functionalities that allow to handle user requests and provide the required information to backend services which are required by Acquirers in C2C transfers. Simply put, this component allows to take advantage of the possibilities offered by Verestro Wallet Server.
Notification Service	Delivers all necessary information about transaction statuses and other actions which was performed between individual Verestro backend components and/or external.
Admin Panel	Frontend component that allows Customer to check transaction statuses and transaction history of his clients.

> Verestro Money Transfer Hub also supports transfers using a QR code. Transfers using the QR code are described in a separate document.

Mastercard Send 2.0

Mastercard Send 2.0 is a technology which enables peer to peer transfer via mobile device. This functionality allows banks and payment institutions to perform transactions using cards, IBANs, QRs or other URI. Each transaction pushed to Mastercard is settled with the payment institution / bank registered in this program on a settlement basis (collective invoice 1 per day).

> Mastercard Send 2.0 technology is available only in the test environment

MC Send 2.0 Payments high level overview

This diagram shows high level components which are involved in whole solution.

image-1658903536789.png

MC Send Payments Key Components

Component	**Description**

Verestro Wallet Server	Backend services of Money Transfer solution. In the described product, they are responsible for adding / keeping the context of the user and the card in the database, user authentication, calculate commissions, forwarding a transaction requests to Mastercard Send 2.0 or keeping transaction history.
Verestro P2P Wallet SDK	A set of functionalities that allow to handle user requests and provide the required information to backend services which are required in Mastercard Send 2.0. Simply put, this component allows to take advantage of the possibilities offered by Verestro Wallet Server.
Notification Service	Delivers all necessary information about transaction statuses and other actions offered by Money Transfer which was performed between individual Verestro backend components and/or external.
Admin Panel	Frontend component that allows Customer to check transaction statuses and transaction history of his clients.

Verestro Money Transfer Hub

Verestro Money Transfer Hub is a solution that was created to make it easier for customers to make quick transfers between two entities - Sender and Receiver. Money Transfer Hub provides functionalities for the management, identification and verification of Users and the possibility of making transfers based on specific methods of data transfer, such as internal user identifiers, data entered basicly "from the finger", QRs etc.

Solution consists of:

- Server components:
    - Wallet Server – backend component,
    - Wallet Admin Panel – frontend component,
- Mobile components:
    - Wallet SDK – Android / iOS libs.

Wallet Types

Money Transfer Hub Solution supports one type of wallet which can be used in the implementation:

- OPEN - user registers itself in the application and provides data like PAN etc.

Note that the Receiver is not required to be a client of the application. The Sender can perform transactions to external Receivers this way as well.

Implementation models

Verestro provides three different implementation models for products: REST API, mobile SDK and standalone version.

REST API

In this model Verestro provides Money Transfer Hub API methods. Customer is responsible for integration with provided API methods with his own application and manage User authentication (based on MDC SDK).

Mobile SDK

In this model Customer is owner of Money Transfer Hub Solution. Verestro provides Wallet SDK and Wallet Server. Customer is responsible for integrate provided SDK with his own application and manage User authentication (based on MDC SDK). More information about each module in Mobile SDK can be found in iOS Transfers Documentation, iOS Receivers Documentation for iOS implementation and Android Transfer Documentation, Android Receivers Documentation for Android implementation.

Standalone

In this model Verestro provides whole Money Transfer Hub Solution: “Ready to use” application with implemented SDK. Furthermore, Verestro manages direct User authentication.

Architecture

This diagram shows big picture of Verestro Money Transfer Hub architecture.

image-1658908839784.png

Server Components

Server components are backend services which are designed to process requests from the mobile part, provide the necessary information such as user ID and communicate with Acquirers.

Deployment Models

In Money Transfer Hub Solution Server components are deployed and configured on Verestro side. Verestro is responsible for maintaining infrastructure, deploying applications and monitoring.

Wallet Server

Wallet Server is the backend component which consists of few internal services which are responsible for managing users, cards, IBANs, security tokens, transactions and transactions history. This component is also responsible for connection with Acquirers. Services included in the Wallet Server component can be divided into two groups:

- Services that are part of the Money Transfer Solution.
- Services supporting the functionalities offered by Money Transfer Solution.

List of services which are the part of the Money Transfer Solution:

- Mobile API - available via Wallet SDK to performs operations directly from mobile device, performs client app authentication.
- Wallet Mobile SDK - The mobile part of the Money Transfer solution. Responsible for forwarding customer requests to the appropriate Verestro backend components. All key processes such as logging in, authentication or transaction take place through this component.
- Money Transfer API – One of the Verestro backend services. This service handles requests from Verestro Wallet SDK and communicates with other Verestro internal services which are supporting the Money Transfer solution. Money Transfer Hub API is also responsible for communication with the Acquirer.

List of services which are supporting the functionalities offered by Money Transfer Solution (each of the services listed below has dedicated documentation):

- LC API - server API dedicated for Issuer to manage users and cards data on Wallet Server. Connection is secured using VPN,
- MDC API - server API which is responsible for providing access token to Mobile API. It is also an intermediary between the mobile SDK and Data Core,
- DC API - server API which stores card and user data,
- Admin Panel – frontend application which allows Customer to check transaction status and/or history,
- Midas API - server API integrating Acquirers and their individual 3ds authentication strategies. Additionally Midas API stores mid configuration,
- THC API - server API responsible for keeping transaction history. The data stored in the THC API is used by the Admin Panel.

Wallet Server operates with domain objects like:

- User (Sender) - User which is using Money Transfer Hub Application,
- Session Token – Token which defines User. It is an authorization way of the User,
- Device – This entity is created after user registration and is required to login the User,
- Card - User Card which can be charged or recharged,
- Receiver - Verestro Wallet user or external entity which receives funds from the Sender,
- IBAN - belongs to user. User can have many IBANs. IBAN is identified via id which is sha256Hex value of IBAN. One IBAN can be assigned to multiple users.

More detailed information about objects above are described in Terminology chapter.

Wallet Admin Panel

Wallet Admin Panel - web frontend application which is dedicated for Customer to follow user’s transactions. Using the admin panel, the customer can also add his users to the Verestro Wallet database.
This component is not a part of the Money Transfer Solution but it is supporting some features. For more information about Wallet Admin Panel see “Verestro Wallet Admin Panel Documentation”.

Mobile Components

Mobile components are dedicated for using on Android and iOS mobile devices.

Wallet SDK

Verestro provides Software Development Kit (SDK) called Wallet SDK which can be used for mobile money transfer. As a company Verestro provides many products which can be used in single application. For that reason Wallet SDK is divided into separated modules which covers different functionalities. There are two main modules dedicated for Verestro Money Transfer Hub Solution: P2P SDK and QR SDK. P2P SDK provides user data management such as authentication and payment cards management. It is also responsible for initiating peer to peer transactions and for adding individual recipients to “favorites”.

QR SDK is responsible for creating the appropriate QR code, parse it and for transferring the data contained in this code. Based on such data, a transaction will be initiated.
Note that both SDKs are separated. This means that the for example P2P SDK will not have components that have a QR SDK.

Below is a detailed list of SDKs included in Mobile Components:

- P2P Transfers SDK - supports the process of generating and reporting transactions. The share of this module in the application takes its payment functions to a higher level, enabling the initiation of transfers to a card, telephone number or QR code *(for more technical information please check “P2P Transfer SDK documentation”)*.
- P2P Receivers SDK - supporting module that improves the service of senders. The function allows you to store a list of recipients for a given user and to obtain data for transaction initiation to a telephone number *(for more technical information please check “P2P Receivers SDK documentation”)*.
- QR SDK - The QR module was designed to work with the applicable MPQR (Merchant Presented QR) standard developed by EMV. Thanks to the integration of this module with P2P and meeting the requirements of Mastercard, the user will be able to pay with sellers using QR codes. An additional functionality is that the user can use the code generated for his card and thus receive funds from other people within one implementation *(for more technical information please check “QR SDK documentation”)*.

Access

The account at Verestro Artifactory is required to get access to Verestro repository.

Versioning

SDK version contains three numbers. For example: 1.0.0.:
(For more information check what is "Semantic Versioning" standrand)

- First version digit tracks compatibility-breaking changes in SDK public APIs. It is mandatory to update application code, to use SDK, when this is incremented.
- Second version digit tracks new, not compatibility-breaking changes in public API of SDK. It is optional to update application code, when this digit is incremented.
- Third version digit tracks internal changes in SDK. No updates in application code are necessary to update to version, which has this number incremented.

Changes not breaking compatibility:

- Adding new optional interface to SDK setup,
- Adding new method to any domain,
- Adding new enum value to input or output,
- Adding new field in input or output model.

Communication with Wallet Server

Wallet SDK at the very beginning performs authentication of application and device to Wallet Server.

Security

MDC SDK is responsible for most of the security issues. However, in the Money Transfer Hub solution, sensitive data such as PAN or CVC are processed. They are taken as an array of characters. This data is not held, but immediately wiped from RAM.

Security Checks and Data Clearing

There are performed security checks on Wallet SDK side. Security checks consists of:

- root access detection,
- hooking protection,
- debugging protection,
- custom ROM protection,
- data tampering protection.

Requirements

Wallet SDK has some mandatory requirements to make it work:

- device cannot be rooted,
- Android OS should be in version 6.0 or above,
- iOS OS should be in version 13.0 or above,

- devices cannot have enabled debugging,
- MDC SDK integration.

Configuration

The entire solution requires configuration data necessary for the product to operate in line with the Customer's expectations. The most important information is:

- Product’s name – this name will be representing a given Issuer in Verestro system,
- How transactions are reported to THC – this point tells about what type of transactions and transactions with what status will be reported to the transaction history database. For example:
  - whether the transaction should be stored in THC while at the funding stage or only when the recipient's account top-up has been performed,
  - whether transactions with FAILED status should be reported or only successful transactions,
- To which ACQ we should send the transaction request,
- Whether the product is supposed to support 3DS or not,
  - If the Product supports 3DS, Verestro have to integrate into this process with a given ACQ (unless it has already been done),
  - If the Product supports 3DS, the Customer have to provide the data of his account created in the ACQ’s system such as login, password, merchant Id (mid) and terminal Id if exists (unless the whole mid configuration already has been done in Verestro system).

Use cases

This section describes a detailed description of the processes provided in the solution and the appearance of the application from the end user point of view.

Wallet Server MDC API

This section describes use cases which can be initiated using Wallet Server MDC API. This API should be used by Customers through integrated Money Transfer Wallet SDK to manage Users and cards data on Wallet Server.

User with Card registration

User with Card Registration is process when user and cards are transferred to Wallet Server to make possible use them in different processes (e.g. money transfer) later in the application. Registration can be done when user starts using Verestro Money Transfer Application. Basically User have to provide all necessary personal and card data to start using application. This data goes through the Mobile SDK to MDC API which will return session token. MDC API is also responsible for the subsequent user authentication.
It is required to perform card strong verification. The 3ds card verification will be started by application after adding card.

This diagram shows high level User registration process.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
participant "User" as user
participant "Mobile App" as mob
participant "Verestro Mobile SDK" as sdk
participant "Verestro Mobile Data Core" as mdc
participant "Verestro Data Core" as dc
note right of user: User initiates registration
user->mob: 1. Registration
mob->sdk: 2. Provides User's data
sdk->mdc: 3. Provide User's data
mdc->mdc: 4. Validates request
user<-mdc: 5. Sends SMS OTP / Activation link
user->mob: 6. Activates account
mob->sdk
sdk->mdc: 7. Provides User's data and pair device with User
mdc->dc: 8. Store user
mdc<-dc: 9. User added to Wallet collection
mob<-mdc: 10. Registration success
user<-mob: 11. Your account has been successfully registered
user->mob: 12. Login
mob->sdk: 13. Provides login data
sdk->mdc: 14. Check if provided login data are valid
sdk<-mdc: 15. Success - returns session token
mob<-sdk: 16. Login success
user<-mob: 17. Login success
@enduml
```

This diagram shows high level card registration process.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
participant "User" as user
participant "Mobile App" as mob
participant "Verestro Mobile SDK" as sdk
participant "Verestro Data Core" as dc
participant "Bank" as bank
note right of user: User initiates card registration and provides necessary data (first and last name, card no, cvc, exp date, password)
user->mob: 1. Provides card data
mob->sdk: 2. Provides obtained card data
sdk->dc: 3. Provide obtained card data
dc->dc: 4. Add card to Verestro Wallet
sdk<-dc: 5. Card has been added to Verestro collection
mob<-sdk: 6. Success
user<-mob: 7. Card added - perform strong verification
note right of user: User starts card verification process
mob->sdk: 8. Verify card
sdk->dc: 9. Verify card
dc->bank: 10. Check if provided card data are valid and perform 3ds
dc<-bank: 11. Success
dc->dc: 12. Changes card verification level
sdk<-dc: 13. Card strong verified
mob<-sdk: 14. Card strong verified
user<-mob: 15. Your card has been successfully verified
@enduml
```

Wallet Server Money Transfer API

This section describes use cases which can be initiated using Wallet Server P2P API. This API should be used by Customers through integrated Money Transfer Wallet SDK to manage Transactions and Transaction’s Senders/Receivers, Commission calculation and determine Currencies on Wallet Server. Every method below is secured by session token.
or more information about session token, please see "User with Card registration".

Receiver management

In order to make any transaction, it is necessary to determine who the funds will be sent to. This section presents the method of collecting the Receiver’s data so that the Sender can order a transfer of funds. To illustrate this process, two sequence diagrams were made as there are two Receiver types in the Money Transfer Hub solution – Internal Receiver and External Receiver. *For more information about Receiver types, please see "Terminology".*

This diagram shows high level External Receiver management.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
participant "User" as user
participant "Mobile App" as mob
participant "Verestro Mobile SDK" as sdk
participant "Verestro Money Transfer API" as p2p
participant "Verestro Data Core" as dc
note right of user: User initiates money transfer
user->mob: 1. User opens contacts list
mob-->sdk
sdk->p2p: 2. Which contacts exist in Verestro Wallet?
p2p->dc: 3. Which contacts exist in Verestro Wallet?
note left of dc: Check by phone number provided by Wallet SDK
p2p<-dc: 4. Provides existing Receivers IDs
sdk<-p2p: 5. nProvides existing Receivers IDs
mob<-sdk: 6. Returns information which contact exists in Verestro Wallet
```

```
note right of user: Receiver existing in Verestro Wallet will be highlighted
user->mob: 7. Chooses Receiver existing in Verestro Wallet from contacts
note right of user: In this case all Receiver's necessary data are provided by Verestro Server
user->mob: 8. Confirms chosen Receiver
@enduml
```

In this process all Receiver’s data are stored in Verestro Wallet Server as Receiver is also Verestro Wallet user.

This diagram shows high level External Receiver management.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
participant "User" as user
participant "Mobile App" as mob
participant "Verestro Mobile SDK" as sdk
participant "Verestro Money Transfer API" as p2p
participant "Verestro Data Core" as dc
note right of user: User initiates money transfer
user->mob: 1. User opens contacts list
mob-->sdk
sdk->p2p: 2. Which contacts exist in Verestro Wallet?
p2p->dc: 3. Which contacts exist in Verestro Wallet?
note left of dc: Check by phone number
p2p<-dc: 4. Provides existing Receivers IDs
sdk<-p2p: 5. Provides existing Receivers IDs
mob<-sdk: 6. Returns information which contact exists in Verestro Wallet
note right of user: Receiver existing in Verestro Wallet will be highlighted
```

```
user->mob: 7. Chooses Receiver which doesn't exist in Verestro Wallet from contacts
note right of user: In this case Sender is responsible for provide all data which application requires to perform transfer
user->mob: 8. Confirms provided necessary Receiver's data
@enduml
```

In this process all Receiver’s necessary data should be provided by Sender. Provided data will be validated by Verestro to confirm they are correct – for example is provided card number is compatible with Luhn alghorithm.

Determine currency and calculate commission

Before making a transfer, Sender needs to know what currencies the card supports. This is the first step in the Money Transfer card to card transaction process. Information about the currencies supported by the card is provided by Acquirer by contacting the card Issuer.

The next and equally important step is to calculate the commission for the transfer and possibly currency conversion. The currency conversion fee and information about the current exchange rate are provided by Acquirer.
Acquirer receives a file with current rates from Mastercard or VISA every day and keeps it on his side.

This diagram shows high level determine currency and calculate commission processes.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
participant "User" as user
participant "Mobile App" as mob
participant "Verestro Mobile SDK" as sdk
participant "Verestro Money Transfer API" as p2p
participant "Acquirer" as acq
note left of mob: User has chosen Receiver
user->mob: 1. Chooses Receiver
mob->sdk: 2. Provides chosen Receiver data
sdk->p2p: 3. Determines available currencies
note left of sdk: All data was provided during the Receiver selection process
p2p->acq: 4. Gets available currencies from Acquirer
p2p<-acq: 5. Provides available currencies
sdk<-p2p: 6. Provides available currencies
mob<-sdk: 7. Returns obtained currencies
user<-mob: 8. Shows available currencies
note right of user: User sees chosen Receiver and available currencies
user->mob: 9. Provides amount to be transferred and chooses currency
mob->sdk: 10. Provides chosen amount and currency
sdk->p2p: 11. Calculate commission
p2p->acq: 12. Get actual currency rates if chosen currencies are different
note left of acq: Acquirer has actual currency rates which is provided to him by MC or VISA
p2p<-acq: 13. Returns requested actual currency rates
sdk<-p2p: 14. Provides actual currency rates
mob<-sdk: 15. Provides actual currency rates
user<-mob: 16. Shows actual currency rates
user->mob: 17. Confirms Money Transfer
@enduml
```

In this process, the Sender sees what additional charge will be made after the transfer is processed and then decides whether the transfer of funds is to be made.

3DS Authentication

The Money Transfer Hub Solution supports the 3DS 2.0 process and it is required when user is initiating a transaction. This is an authentication method based on the alleged cardholder data check, biometric authentication and improved customer experience.

As mentioned in the "Configuration" paragraph, a specific 3DS integration is required depending on which ACQ Verestro will connect to when making a transaction.

- If the integration with a given ACQ has already been made, Verestro only need to configure the appropriate merchant identifier, which should be provided by the Customer.

If the integration with the required billing agent has not yet been completed, it will be one of the activities for which Verestro will be responsible. Note that integration with 3DS increases the scope of required development.
In card to card transfer, the authentication process is required. It is to confirm that the user is definitely the owner of the card.

This diagram shows high level 3ds authentication processes.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
participant "User" as user
participant "Mobile App" as mob
participant "Verestro Mobile SDK" as sdk
participant "Verestro Money Transfer API" as p2p
participant "Acquirer" as acq
participant "Mastercard/VISA" as mcvisa
participant "ACS/Issuer" as acs
note left of mob: User has confirmed currencies and accepted shown currency rate
user->mob: 1. Enters CVC code
note left of mob: In this step the application begins 3DS authentication process.
note left of mob: If the bank decides that 3ds is not required, not any action will be performed.
note left of mob: Diagram shows the option requiring the user to authenticate.
mob->sdk: 2. Initialize 3DS process
sdk->p2p: 3. Initialize 3DS process
p2p->acq: 4. Initialize 3DS process
note left of acs: Bank returns decision whether it is necessary for the user to authenticate
acq->mcvisa: 5. Is 3DS authentication required?
acq<-mcvisa: 6. ThreeDS Method
```

```
p2p<-acq: 7. ThreeDS Method
p2p->acq: 8. Continue 3DS process
acq->acs: 9. Continue 3DS process
acq<-acs: 10. Challenge required
p2p<-acq: 11. Challenge required
sdk<-p2p: 12. Challenge required
mob<-sdk: 13. Challenge required
user<-mob: 14. Informs user that challenge is required
note left of mob: Bank's page content is provided
user->user: 15. Performs challenge
note right of user: After a successful challenge, the Bank sends PaRes/cRes, which is intercepted by the Wallet SDK and provided to the Money Transfer API
user-->mob
mob-->sdk
sdk->p2p: 16. Provides intercepted PaRes/cRes
p2p->acq: 17. Provides obtained PaRes/cRes
acq->acs: 18. Check authentication for provided PaRes/cRes
acq<-acs: 19. Authentication success
p2p<-acq: 20. Authentication success
sdk<-p2p: 21. Authentication success
mob<-sdk: 22. Authentication success
user<-mob: 23. Informs about the successful completion of the authentication process
@enduml
```

To facilitate understanding of the process flow, each of the steps is described below in the correct order *(points highlighted in blue are performed outside the Verestro Server)*:

1. Mobile application contacts the Verestro Server via the Verestro Mobile SDK to start 3ds process.
2. Mobile SDK provides all necessary data to the Money Transfer API (including user/card id and 3ds authentication request id) while calling the 3ds initialization method.
3. Having all the necessary information, Money Transfer API orders Acquirer to start the 3ds authentication process.
4. Acquirer transfers the card and user details to ACS.
5. If the user is the owner of the card, the ACS returns a positive answer and a decision whether the continuation of the authentication process is required (according to the diagram above, the scenario with the necessity to continue the process is described).
6. ACS informs the Acquirer that the 3ds process continue is required.
7. Acquirer informs Money Transfer API about ACS decision.
8. Money Transfer API requests Acquirer to continue the authentication process (the authentication id is provided.
9. Acquirer provide the request to continue the process to the ACS.
10. ACS informs about the necessity to perform the Challenge process and returns necessary parameters such as:

    1. challengeHtmlFormBase64 - this field is a BASE64 encrypted html source file containing the ACS’ challenge 3DSecure frame
    2. cReq - data for building a form such as challengeHtmlFormBase64
11. Acquirer informs Verestro Money Transfer API that ACS requires Challengeand provides above parameters.
12. Verestro Money Transfer API forwards the obtained information to Verestro Mobile SDK.
13. Verestro Mobile SDK decodes the received challengeHtmlFormBase64 parameter and transmits the received frame of the mobile application.
14. The user is redirected to the bank's website where he performs the Challenge process.
15. After a successful Challenge process, the bank sends the cRes / PaRes parameter. This response is intercepted by the Verestro Mobile SDK and forwarded to the Verestro Money Transfer API.
16. Verestro Money Transfer API provides the received cRes / PaRes to Acquirer.
17. Acquirer provides the above parameters to ACS for verification.
18. If everything was done correctly, the ACS informs Acquirer about the successful completion of the 3ds authentication. Among other things, the following parameters are included in the response
    1. cavv - property determined by the ACS. The value may be used to provide proof of authentication
    2. eci - property is determined by the ACS. This property contains the two digit Electronic Commerce Indicator (ECI) value, which is to be submitted in a credit card authorization message
19. Acquirer provides information about successful completion of the 3ds authentication among with the above parameters obtained from the ACS to Verestro Money Transfer API.
20. Verestro Money Transfer API provides information about successful completion of the 3ds authentication among with the above parameters obtained from the ACS to Verestro Mobile SDK.
21. Verestro Mobile SDK provides positive response to mobile application. The information is shown to the user.

Transaction process

If the user and his card are present in the Verestro system, such a user can perform transactions using the Money Transfer Hub solution.

Card to card money transfer

This type of transaction allows Sender to make money send transfers from card to card. The transaction is secured by the 3DSecure process.

This diagram shows high level money transfer card to card processes.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
participant "User" as user
participant "Mobile App" as mob
participant "Verestro Mobile SDK" as sdk
participant "Verestro Money Transfer API" as p2p
participant "Verestro Data Core" as mdc
participant "Verestro THC API" as thc
participant "Acquirer" as acq
participant "Issuer" as acs
note left of mob: User has passed 3DS authentication process
mob->sdk: 1. Perform money transfer
sdk->p2p: 2. Perform money transfer
p2p->mdc: 3. Check if provided card belong to the user
p2p->mdc: 4. Check if provided card belong to the receiver
p2p<-mdc: 5. OK
p2p<-mdc: 6. OK
p2p->acq: 7. Money send
acq->acs: 8. Peform Funding from provided card if possible
acq<-acs: 9. Success
acq->acs: 10. Perform Credit for Receiver account
acq<-acs: 11. Success
p2p<-acq: 12. Success + transaction id
note left of p2p: Receiver will be added to "favourite" in Senders contacts if checkbox has been checked - optional
p2p->thc: 13. Store transaction with status Funding
p2p<-thc: 14. Transaction has been stored successfully
sdk<-p2p: 15. Transaction success
```

```
mob<-sdk: 16. Transaction success
user<-mob: 17. Your transaction has been sent
@enduml
```

After ordering the transfer, Wallet Mobile SDK forwards request to the Wallet API. Wallet API in turn forwards the request to the Acquirer. Based on the data received, the Acquirer contacts the bank to complete the transaction. If the bank agrees, the funding process is carried out - collecting funds from the Sender's account. After the funding is completed, the funds are transferred to the appropriate Receiver.

To facilitate understanding of the process flow, each of the steps is described below in the correct order *(points highlighted in blue are performed outside the Verestro Server)*:

1. After going through the 3ds process, the funds transfer process begins,
2. The application orders the transfer of funds by communicating with the Verestro backend via the Verestro Mobile SDK,
3. Money Transfer API communicates with Data Core to check whether the card details belong to the user and the receiver,
4. After receiving a positive response from Data Core, Money Transfer API performs money send transaction,
5. Acquirer receives a request to make a transfer of funds,
6. The Acquirer communicates with the Issuer regarding the execution of the transfer,
7. The Sender's account is debited,
8. The Receiver’s account is credited,
9. The Acquirer receives information from the Issuer that the operation has been performed,
10. Acquirer informs Money Transfer API about the positive status of the ordered operation,
11. Money Transfer API saves transaction details in the Transaction History Core API,
12. Money Transfer API informs Mobile SDK about the positive status of the ordered operation,
13. Mobile SDK informs Mobile Application about the positive status of the ordered operation,
14. The Mobile Application displays to the user a successful transfer of funds.

> Verestro Money Transfer Hub also supports transfers using a QR code.
> Transfers using the QR code are described in a separate document.

Application flow

This chapter introduces the main actions and processes in the application from the end user point of view.

User registration flow

Invoking registration is usually the primary activity right after the first launch of the application. User registration, depending on the configuration, requires providing data, including authentication, which are at a later stage his login to the application. Required data are e-mail address, telephone number, accepting tos, assigning a password to the account and assigning a pin to the application.

The pictures below show the first step of the registration process:

image-1650446717200.png	image-1650446725423.png	image-1650446735943.png
On the screen above user registers in to application by clicking “Sign up” button.	After clicking "Sign up" button, the user is redirected to the screen with the fields that must be filled in to register in the application. Fields which are required: • E-mail, • Phone number, • Password, • Repeat password, • Accept Terms & Conditions.	After providing all the necessary data, the user can go to the next step of registration by clicking the "Next" button. The “Next” button remains inactive until all required fields are correctly filled in.

image-1650446991254.png
In the screen beside the pop-up with regulations has been shown. This pop-up is displayed to the user after checking the "Accept Terms & Conditions" checkbox. The user can accept the regulations by clicking the "Accept" button or reject it by clicking the "Discard" button. Acceptance of the regulations is required to complete the registration process, and thus to use the application.

The pictures below show the “Set up PIN” view:

image-1650447077291.png	image-1650447083737.png

On the screen above user sets PIN of his account. To do such the user need to provide four digits and repeat them - this prevents from making a mistake. To save new PIN the user must confirm changes by clicking Confirm button. Button remains inactive unless
user provides new PIN and repeat it correctly. User is able to exit this section by clicking “<” button in the upper left corner.

User login flow

The first login of the user takes place using the login and password. The login, depending on the configuration, may be a telephone number or an e-mail address. When logging in, the user may agree to log in to the application using biometrics, as long as his device supports such a solution. Additionally, it is possible to reset the password in case the user forgets it.

The pictures below show the login process view with “use fingerprint” checkbox:

image-1650448497416.png	image-1650448506875.png	image-1650448580854.png
On the screen above user logs in to application. Sign in button remains inactive unless user provides his phone number and password. Phone number and password must be valid. Password can be reseted. User may also login by fingerprint by checking “Use fingerprint” checkbox.	On the screen above user tries to reset his password by providing his email. Email must be valid.	Until the user enters the email, the "SEND" button remains inactive. The authorization code will be sent to provided email.

The next login takes place in accordance with the settings selected by the user.

The pictures below show other possibilities of the login:

image-1650453306410.png	image-1650453312442.png	image-1650453318217.png
On the screen above user logs in to application using his fingerprint. To do this, the user has to tap his finger on the screen. It is also possible to use a PIN by clicking “Use PIN” button in the down left corner of the screen.	On the screen above user logs in to application by PIN. PIN must be valid. User may select more login options by clicking “MORE OPTIONS” button.	On the screen above user chooses how he wants to log in.

Selecting the "Reset PIN" option will redirect the user to the following subpage:

image-1650453419929.png
In the screen beside user is able to reset PIN. To perform this action the user must authorize himself with his standard password. The option of logging in with a standard password and resetting the password was presented earlier in the chapter "User login flow". Possible errors: • Invalid fingerprint, • Invalid PIN, • Too many failed attempts, user PIN, • Too many failed attempts, user login and password.

Add card flow

The application, in cooperation with Verestro backend compliant with PCI DSS, provides the possibility to perform operations on cards belonging to its owner. Adding a card from the outside can be done by scanning its data with a camera built into the device or by manually entering its data into the form. Adding such a card requires its verification by calling 3ds and is successfully completed after the card issuer is authorized to use the card.

The pictures below show the possibility of the add card by scanning its data:

image-1650453901514.png	image-1650453909876.png

On the screen above user adds his card by scanning its data using device camera. The user can also add the card manually by selecting the option "enter card data manually".

The pictures below show the possibility of the add card by providing its details manually:

image-1650454118693.png	image-1650454126225.png

On the screen above user provides card data and chooses whether this card will be the default one or not. After providing all the necessary data, the user can confirm adding a card with the "Confirm" button. The “Confirm” button remains inactive until all required fields are correctly filled in.

After adding the card, the application will require its verification with the 3ds standard. To authenticate, an SMS will be sent with a code to the phone number assigned to the user’s account. This code should be entered in the designated place.

The pictures below show the 3ds process that the user must go through in order to be able to use the added card in the application:

image-1650454217524.png	image-1650454225732.png

Add address

The address module in the application can be used as a private, highly secured section for the user's needs. In addition, it acts as a storage of the user's address, which, due to the legal requirements regarding AML, is used during card transactions from the application level. The data to be provided in the add address section are listed below:

- First name,
- Last name,
- Company name,
- Street,
- House number,
- Local number,
- Postal code,
- City,
- Phone number,
- NIP,
- Country,
- Checkbox - default address.

The pictures below show “adding address” view:

image-1650456748251.png	image-1650456754093.png

On the screen above user provides his address details. After providing all the necessary data, the user can confirm adding an address with the "Confirm" button. The “Confirm” button remains inactive until all required fields are correctly filled in.

image-1650456795698.png

On the screen beside user is able to whether update his address or delete it. In case of update each field can be changed. User can confirm his changes by clicking “Confirm” button. This button remains inactive until all required fields are correctly filled in. By checking “My address” checkbox, the user confirms that this is his default address. User can delete his address by clicking “waste-basket” button.

Receiver management flow

Receivers is a module that streamlines and automates getting the recipient object to generate a transaction via Money Transfer Hub. The usefulness of this module takes place in 2 key functions:

- Active users – feature which allow to provide the user with information which contacts from his personal contact list also use this application. Thanks to this solution, users maintained within one infrastructure can transfer funds between themselves "to the telephone number". Receivers which also use the application are marked with the “V” logo,
- Last used / favorite receivers – feature which allow to store a personal, secure contact book with cards. Communication between the application and the server after a single transmission of the encrypted card number takes place later on the basis of unique identifiers, where the card data itself on the servers are stored and used in accordance with the PCI DSS standard, ensuring the security of all data entrusted to it.

The pictures below show the user’s contact list:

image-1650456966776.png

In the screen beside user is able to choose transfer receiver from his contact list. Receiver which uses Money Transfer Application is marked with “V” logo. The user has the option of filtering potential receivers by entering the receiver’s data (name, surname) or by marking the "V" filter which means that only active users will be displayed in the contact list. There is also an option to add new
receiver but clicking “+” button.

The pictures below show the user’s contact list with active filters:

image-1650457156844.png	image-1650457164176.png
On the screen above the user’s contact list with active “Money Transfer Application users” filtering has been shown.	On the screen above the user’s contact list with active “Your recipients” filtering has been shown.

Adding a new receiver can be divided into two cases. New and existing receiver. If a receiver already exists in user's list of recipients, an appropriate message is displayed and there is an option to add another card which will be assigned to him. If receiver does not exist in the user's list of recipients, the user must assign him a card. In both cases the "Confirm" button remains inactive until all required fields are completed.

The pictures below show case of adding receiver:

image-1650457330593.png	image-1650457336627.png	image-1650457341313.png
On the screen above user adds new receiver by providing necessary data – first name, last name and phone number with prefix. User can confirm adding new receiver by clicking the "Confirm" button. The "Confirm" button remains inactive until all required fields are correctly filled in.		On the screen above user is informed that his new receiver has a card assigned to him. The user can choose whether he use this card or add new one.

The pictures below show case of next step of adding receiver (this step occurs only if the user wants to assign new card to his receiver):

image-1650457428569.png	image-1650457435242.png

On the screen above user adds new card to the receiver by providing necessary data – display name and card number. There is also "Save recipient" checkbox which is responsible for save the receiver in "favourite ones" list. User can confirm adding new receivers card by clicking the "Confirm" button. The "Confirm" button remains inactive until all required fields are correctly filled in.

Transaction flow

The function of this module is to generate a payment transaction from data entered manually by the user or supplied from other modules such as QR or Friends. Transaction are secured by 3ds protocol.

Card to card transaction

This module allows you to make a transfer from card to card. The user selects the card which will be debited and the recipient's card which will be topped up.

The pictures below show "money transfer" view:

image-1650540852315.png	image-1650540859358.png	image-1650540866062.png

On the screen above user chooses which of his cards will be debited and then confirm his choice. Chosen receiver is shown on the screen.	On the screen above user chooses currency and the amount of the transfer. The transfer button remains inactive until all required fields are correctly filled in.	On the screen above user filled in all required fields and is able to perform the transfer. In addition the information about debit amount is shown.

image-1650541058468.png	image-1650541065810.png	image-1650541072109.png
On the screen above user is able confirm the transfer using his fingerprint or PIN.	On the screens above the 3ds authentication is required to perform money transfer. Bank sends the authentication code via SMS.	

image-1650541140975.png	image-1650541195027.png
On the screen above user is informed about the successfully money transfer. Return button will redirect the user to the main page.	On the screen above user is informed about the unsuccessfully money transfer. In this case user is able to try again the transfer or return to the main page. Example reasons of the transaction fail: • Luck of funds, • 3ds failed, • Invalid card data.

> *To see the appearance of the QR transaction component from the end user level please visit QR Payments Application flow*

Transaction history flow

In the Payment History section, the user can check the history of his transactions in the application as well as make statements of charges and credits from given time periods.

The pictures below show some of “payment history” view:

image-1651756901486.png The picture above shows the contents of the "Transactions" tab. After entering this tab, the history of all transactions sorted by the newest and divided per day is shown to the user.	image-1651756895522.png The picture above shows the contents of the "Spends" tab. After entering this tab, a list of all expenses for a given month is shown to the user, broken down by type of expense.

image-1651756930461.png	image-1651756932836.png
On the screen above there is transaction history filtering option. The user can narrow down the number of results by selecting different filtering criteria. In the section, there is also possibility to remove all selected filters by clicking the "Remove all filters" option.	On this screen, the user sees information about which card the transaction was related to with its thumbnail, who is the addressee, transaction category, transaction ID, as well as its status, transaction amount, date and time of the transfer.

Technical Documentation

Technical Documentation API

Money Transfer Hub provides possibility to process Person-2-Person and Person-2-Merchant transactions in various forms. Please check details in the below documentation.

> This documentation contains the methods for **mobile-server** integration. The methods included in the documentation are intended for Customers creating their own mobile SDK.

> The Customer creating the SDK must also remember about the integration with the MobileDC component

> Documentation for the **server-to-server** integration is available here but is **deprecated.**

Receiver types which can be used to set Receiver.Type

Based on ReceiverType user can fill different field in Receiver object in requests.

ReceiverType	**Description**
BARE_CARD_NUMBER	Bare card number in **Receiver.card** field
FRIEND_ID	Should pass FriendId in **Receiver.Card** field
WALLET_CARD_ID	Should pass DataCoreCardId to **Receiver.Card** field and UserDataCoreCardId to **Receiver.userId** field
EMPTY	Means that the receiver have the same card data like sender. This type may be useful on Determine Currency

JWE

Peer To Peer Transaction Service supports encryption of requests and responses as standard JSON Web Encryption (JWE) per RFC 7516.

Recommended to read the JWE standard: RFC 7516.

Methods that support request encryption in the JWE standard are tagged in the documentation with the header: *Content-Type:application/x-jwe-encryption-body+json*. If the response is to be encrypted with the JWE standard then the header must be added: *X-Encryption-Public-Key* with the public key.

Processing requests and responses can be divided into 4 options listed below:

1. Base request → Base response - the following headers should be provided to pass this case:
   - *Content-Type: application/json*
2. Base request → Encrypted response - the following headers should be provided to pass this case:
   - *Content-Type: application/json*
3. Encrypted request → Base response - the following headers should be provided to pass this case:
   - *Content-Type: application/x-jwe-encryption-body+json*
4. Encrypted request → Encrypted response - the following headers should be provided to pass this case:
   - *Content-Type: application/x-jwe-encryption-body+json*

Overview

JWE represents encrypted content using JSON data structures and Base64 encoding. The representation consists of three parts: a JWE Header, a encrypted payload, and a signature. The three parts are serialized to UTF-8 bytes, then encoded using base64url encoding. The JWE's header, payload, and signature are concatenated with periods (.).

JWE typically takes the following form:

```
{Base64 encoded header}.{Base64 encoded payload}.{Base64 encoded signature}
```

JWE header contains:

Type	Value	Constraints	Description

alg	RSA-OAEP-256	Required	Identifies the cryptographic algorithm used to secure the JWE Encrypted Key. Supported algorithms: **RSA-OAEP-256**, **RSA-OAEP-384**, **RSA-OAEP-512**. Recommend value: **RSA-OAEP-256**.
enc	A256GCM	Required	Identifies the cryptographic algorithm used to secure the payload. Supported algorithms: **A128GCM**, **A192GCM**, **A256GCM**, **A128CBC-HS256**, **A192CBC-HS384**, **A256CBC-HS512**. Recommend value: **A256GCM**.
typ	JOSE	Optional	Identifies the type of encrypted payload. Recommend value: **JOSE**.
iat	1637929226	Optional	Identifies the time of generation of the JWT token. Supported date format: unix time in UTC. In the case of *iat*send, the validity of JWE is validated. Recommend send the header due to the increase in the security level.
kid	5638742a5094327fcd7a5945d06a45a9d83e9006	Optional	Identifies the public key of use to encrypt payload. Supported format: SHA-1 value of the public key. In the case of *kid* send, the validity of public key is validated, so we can inform the client that the public key has changed.

Payload Encryption

Every encrypted request should include JWE token. The jwe token should be passed in the field: *value*.

In case of problems with the implementation of JWE, please contact the administrator.

To prepare the encrypted payload:

The steps may differ depending on the libraries used.

1. Get the public key using the method: [???](#Get publicKey). The public key is encoded with Base64.
2. Decode the public key.
3. Then create a correct object to be encrypted.
4. Encrypt the created object with the public key.
5. Create JWE header compatible with: JWE Header
6. Make a request on the method that supports JWE. Set the JWE token in the field: *value*. Methods supporting JWE use the following header: *Content-Type:application/x-jwe-encryption-body+json*.

Payload Decryption

To prepare the decrypted payload:

The steps may differ depending on the libraries used.

The cryptographic algorithm used to secure the payload is: **A256GCM**, while to secure the encrypted JWE key: **RSA-OAEP-256**.

1. For the response to be encrypted you need to send *public key* in the header: *X-Encryption-Public-Key*. The header value must be encoded *Base64*.
2. After receiving the response, you should get the JWE token from the field: *value*.
3. Decrypt the JWE token from the field: *value* with the private key.

Public key format to be encoded in Base64.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0IDAQAB...
-----END PUBLIC KEY-----
```

P2P

Every single method should contains Authorization and Mobile-Product headers.

Active Accounts

Method used to find users with valid mc card type (not expired, strong verified). Response will contain phone numbers with user and card identifiers. Users without accepted TOS or without valid MC card will not be returned in response. If user has multiple cards that match criteria response will contain only user’s default card id.

Request

Request headers

Request body with header: *X-Encryption-Public-Key*

Type	Value	Constraints	Description
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.
X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Request fields

Response

Error response - ERROR_VALIDATION.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
   "traceId": "{{traceId}}",
   "errorStatus": "ERROR_VALIDATION",
   "message": "Some fields are invalid",
   "data": [
       {
           "field": "{{field_name_from_request}}",
           "message": "{{message}}"
       }
   ]
}
```

Error response - ERROR_BAD_TOKEN.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "traceId": "{{traceId}}",
    "errorStatus": "ERROR_BAD_TOKEN"
}
```

Error response - PRODUCT_NOT_FOUND.

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "traceId": "{{traceId}}",
    "errorStatus": "PRODUCT_NOT_FOUND",
    "message": "Product by name {{product_name}} not found."
}
```

Error response - INTERNAL_SERVER_ERROR.

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "traceId": "{{traceId}}",
    "errorStatus": "INTERNAL_SERVER_ERROR"
}
```

Response fields

Errors

Encrypted response fields when sent header: *X-Encryption-Public-Key*

Http Status	Error Status	Description
400 - Bad Request	ERROR_VALIDATION	Some fields are invalid
400 - Bad Request	ERROR_BAD_TOKEN	Invalid authorization token
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error decoding public key has sent in header: *X-Encryption-Public-Key*
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error on decrypting request
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error on encrypting response
400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE encryption Key is invalid
400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE payload is expired
400 - Bad Request	INVALID_PHONE_NUMBERS	Phone numbers has incorrect format
404 - Not Found	PRODUCT_NOT_FOUND	Product not found based on sent header: *Product-Name*

500 - Internal Server Error	INTERNAL_SERVER_ERROR	Internal application error

Examples

Determine currency

Request body with header: *X-Encryption-Public-Key*.

Method is used to determine currencies applied for given sender and receiver cards.

Request

Receiver.receiverType = WALLET_CARD_ID.

```
POST /mobile-api/determine-currency HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name:
Content-Type: application/json
Content-Length: 56

{
    "sender": {
        "cardId": "219754"
    },
    "receiver": {
        "card": ["2","1","4","4","9","2"],
        "userId": "1223",
        "receiverType": "WALLET_CARD_ID"
    }
}
```

Receiver.receiverType = FRIEND_ID.

```
POST /mobile-api/determine-currency HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56

{
    "sender": {
        "cardId": "219754"
    },
    "receiver": {
        "userId": "21",
        "receiverType": "FRIEND_ID"
    }
}
```

Receiver.receiverType = EMPTY.

```
POST /mobile-api/determine-currency HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56

{
    "sender": {
        "cardId": "219754"
    },
    "receiver": {
        "receiverType": "EMPTY"
    }
}
```

Receiver.receiverType = BARE_CARD_NUMBER.

```
POST /mobile-api/determine-currency HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56

{
    "sender": {
        "cardId": "219754"
    },
    "receiver": {
        "card": ["2","2","2","1","0","0","4","0","7","2","1","9","0","1","8","5"],
        "receiverType": "BARE_CARD_NUMBER"
    }
}
```

Request headers

Request body with header: X-Encryption-Public-Key

Type	Value	Constraints	Description
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.
X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Request fields

Response

Error response - ERROR_VALIDATION.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
```

```
{
   "traceId": "{{traceId}}",
   "errorStatus": "ERROR_VALIDATION",
   "message": "Some fields are invalid",
   "data": [
      {
         "field": "{{field_name_from_request}}",
         "message": "{{message}}"
      }
   ]
}
```

Error response - ERROR_BAD_TOKEN.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
   "traceId": "{{traceId}}",
   "errorStatus": "ERROR_BAD_TOKEN"
}
```

Error response - PRODUCT_NOT_FOUND.

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
```

```
{
   "traceId": "{{traceId}}",
   "errorStatus": "PRODUCT_NOT_FOUND",
   "message": "Product by name {{product_name}} not found."
}
```

Error response - INTERNAL_SERVER_ERROR.

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
   "traceId": "{{traceId}}",
   "errorStatus": "INTERNAL_SERVER_ERROR"
}
```

Response fields

Errors

Encrypted response fields when sent header: *X-Encryption-Public-Key*

Http Status	**Error Status**	**Description**
400 - Bad Request	ERROR_VALIDATION	Some fields are invalid
400 - Bad Request	ERROR_BAD_TOKEN	Invalid authorization token
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error decoding public key has sent in header: *X-Encryption-Public-Key*
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error on decrypting request
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error on encrypting response
400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE encryption Key is invalid
400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE payload is expired

400 - Bad Request	ERROR_SENDER_CARD_NOT_ACTIVE	Sender card is not active
400 - Bad Request	ERROR_RECEIVER_CARD_NOT_ACTIVE	Receiver card is not active
400 - Bad Request	UNKNOWN_ERROR	Unknown error
404 - Not Found	PRODUCT_NOT_FOUND	Product not found based on sent header: *Product-Name*
404 - Not Found	CANT_FIND_CARD	Not found card
404 - Not Found	FRIEND_NOT_EXISTS	Not found friend
500 - Internal Server Error	INTERNAL_SERVER_ERROR	Internal application error
500 - Internal Server Error	ERROR_ON_GETTING_DEFAULT_CARD	Error on getting card for friend
500 - Internal Server Error	FENIGE_ERROR	Fenige error

Examples

Currency Rate

Request body with header: *X-Encryption-Public-Key*.

Method is used for determine currency rate for revaluation from funding to payment (lowerRate) and payment to funding (higherRate).
Notice that `lowerRate` is used to transaction processing.

Api Send-money allows users to select the direction of revaluation by providing specify type value in send-money request.
1 - User by selecting type = SENDER defines amount of funding in given currency. This amount is collected from sender card in selected currency.
2 - User by selecting type = RECEIVER defines amount of payment in given currency.
This amount is transferred to receiver card in selected currency.In case there’s need revaluation from one currency to another, system uses lowerRate for situation 1 and higherRate for situation 2

Request

Request headers

Request body with header: *X-Encryption-Public-Key*

Type	Value	Constraints	Description

Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Response

Error response - ERROR_BAD_TOKEN.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "traceId": "{{traceId}}",
  "errorStatus": "ERROR_BAD_TOKEN"
}
```

Error response - PRODUCT_NOT_FOUND.

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "traceId": "{{traceId}}",
  "errorStatus": "PRODUCT_NOT_FOUND",
  "message": "Product by name {{product_name}} not found."
}
```

Error response - INTERNAL_SERVER_ERROR.

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "traceId": "{{traceId}}",
    "errorStatus": "INTERNAL_SERVER_ERROR"
}
```

Response fields

Errors

Encrypted response fields when sent header: *X-Encryption-Public-Key*

Http Status	**Error Status**	**Description**
400 - Bad Request	ERROR_BAD_TOKEN	Invalid authorization token
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error decoding public key has sent in header: *X-Encryption-Public-Key*
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error on encrypting response
400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE encryption Key is invalid
400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE payload is expired
404 - Not Found	PRODUCT_NOT_FOUND	Product not found based on sent header: *Product-Name*
500 - Internal Server Error	INTERNAL_SERVER_ERROR	Internal application error
500 - Internal Server Error	FENIGE_ERROR	Fenige error

Examples

Calculate commission

Request body with header: *X-Encryption-Public-Key*.

This method is used to receive information about the commission that will be charged for the transaction. Additional description:

- If value the field: "reconciliationType" is "PLUS", the commission during the transaction will be added to the amount sent (the amount charged from the sender will be increased by a commission).
- If value the field: "reconciliationType" is "MINUS", then the commission during the transaction will be deducted from the amount received (the amount that will be received by the receiver will be reduced by the commission).
- If value the field: "reconciliationType" is "DEPOSITED", the commission during the transaction will neither be subtracted nor added (the amount to be received by the receiver is the same as the amount sent).

In addition, the user may specify in the field: type two values **SENDER** or **RECEIVER**.
After selecting the value: **SENDER**, the transaction will be sent in the amount indicated in the field: amount. Whereas after choosing the value: **RECEIVER**, the transaction will be received in the amount indicated in the field: amount. The method allows user to calculate commissions for the currencies that have been entered.

Request

Receiver.receiverType = WALLET_CARD_ID.

```
POST /mobile-api/calculate-commission HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 101

{
    "amount": 200078,
    "type": "RECEIVER",
    "sender":{
        "cardId":"219834",
        "currency":"PLN"
    },
    "receiver":{
        "userId": 2345,
        "card": ["2","2","1","2","4","5"],
        "currency": "PLN",
        "receiverType": "WALLET_CARD_ID"
    }
}
```

Receiver.receiverType = FRIEND_ID.

```
POST /mobile-api/calculate-commission HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 101

{
    "amount": 200078,
    "type": "RECEIVER",
    "sender":{
        "cardId":"219834",
        "currency":"PLN"
    },
    "receiver":{
        "userId": 2345,
        "currency": "PLN",
        "receiverType": "FRIEND_ID"
    }
}
```

Receiver.receiverType = BARE_CARD_NUMBER.

```
POST /mobile-api/calculate-commission HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 101

```

```
{
   "amount": 200078,
   "type": "RECEIVER",
   "sender":{
      "cardId":"219834",
      "currency":"PLN"
   },
   "receiver":{
      "card": ["5","4","9","5","9","8","4","1","7","9","0","8","2","6","4","5"],
      "currency": "PLN",
      "receiverType": "BARE_CARD_NUMBER"
   }
}
```

Request headers

Request body with header: *X-Encryption-Public-Key*

Type	**Value**	**Constraints**	**Description**
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.
X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Request fields

Response

Error response - ERROR_VALIDATION.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
```

```
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "traceId": "{{traceId}}",
  "errorStatus": "ERROR_VALIDATION",
  "message": "Some fields are invalid",
  "data": [
    {
      "field": "{{field_name_from_request}}",
      "message": "{{message}}"
    }
  ]
}
```

Error response - ERROR_BAD_TOKEN.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "traceId": "{{traceId}}",
  "errorStatus": "ERROR_BAD_TOKEN"
}
```

Error response - PRODUCT_NOT_FOUND.

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
```

```
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "traceId": "{{traceId}}",
    "errorStatus": "PRODUCT_NOT_FOUND",
    "message": "Product by name {{product_name}} not found."
}
```

Error response - INTERNAL_SERVER_ERROR.

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "traceId": "{{traceId}}",
    "errorStatus": "INTERNAL_SERVER_ERROR"
}
```

Response fields

Errors

Encrypted response fields when sent header: *X-Encryption-Public-Key*

Http Status	**Error Status**	**Description**
400 - Bad Request	ERROR_VALIDATION	Some fields are invalid
400 - Bad Request	ERROR_BAD_TOKEN	Invalid authorization token

400 - Bad Request	CRYPTOGRAPHY_ERROR	Error decoding public key has sent in header: *X-Encryption-Public-Key*
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error on decrypting request
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error on encrypting response
400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE encryption Key is invalid
400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE payload is expired
400 - Bad Request	ERROR_WHILE_GETTING_COUNTRY_CODE	Could not get card country code
400 - Bad Request	ERROR_WHILE_GETTING_SENDER_COUNTRY_CODE	Could not get card country code for sender
400 - Bad Request	ERROR_WHILE_GETTING_RECEIVER_COUNTRY_CODE	Could not get card country code for receiver
400 - Bad Request	ERROR_SENDER_CARD_NOT_ACTIVE	Sender card is not active
400 - Bad Request	ERROR_RECEIVER_CARD_NOT_ACTIVE	Receiver card is not active
400 - Bad Request	UNKNOWN_ERROR	Unknown error
404 - Not Found	PRODUCT_NOT_FOUND	Product not found based on sent header: *Product-Name*
404 - Not Found	CANT_FIND_CARD	Not found card
404 - Not Found	FRIEND_NOT_EXISTS	Not found friend
500 - Internal Server Error	INTERNAL_SERVER_ERROR	Internal application error
500 - Internal Server Error	ERROR_ON_GETTING_DEFAULT_CARD	Error on getting card for friend
500 - Internal Server Error	FENIGE_ERROR	Fenige error

Examples

Send Money

Request body with header: *X-Encryption-Public-Key*.

This method is used to full MoneySend transaction (funding and payment).
Transfers can be make in any currency.
1 - User by selecting type = **SENDER** defines amount of funding in given currency.
This amount is collected from sender card in selected currency. 2 - User by selecting type = **RECEIVER** defines amount of payment in given currency.
This amount is transferred to receiver card in selected currency.

In case there’s need revaluation from one currency to another, system uses lowerRate for situation 1 and higherRate for situation 2. For more details about specific rates please refer to Currency Rate method.
This method adds friend to sender after successful transaction.

Additionally, you can perform full MoneySend transaction with externalAuthentication (see: ??? and Authentication)

Request

Receiver.receiverType = WALLET_CARD_ID.

```
POST /mobile-api/send-money HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56

{
    "amount": 1000,
    "cvc2": ["1","2","3"],
    "type": "RECEIVER",
    "addressIp": "192.168.0.1",
    "sender": {
        "firstName": "Mark",
        "lastName": "Wards",
        "street": "Olszewskiego",
        "houseNumber": "17A",
        "city": "Lublin",
        "postalCode": "20-400",
        "flatNumber": "2",
        "email": "senderEmail@fenige.pl",
        "currency": "PLN",
        "expirationDate": "03/20",
        "personalId": "AGC688910",
        "cardId": "219708"
    },
    "receiver": {
        "firstName": "Rob",
        "lastName": "Wring",
        "currency": "PLN",
        "card": ["2","1","9","7","0","8"],
        "displayName": "Rob W.",
        "phoneNumber": "48718222333",
        "receiverType": "WALLET_CARD_ID",
        "userId": "13001"
    }
}
```

Receiver.receiverType = FRIEND_ID.

```
POST /mobile-api/send-money HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56

{
    "amount": 1000,
    "cvc2": ["1","2","3"],
    "type": "RECEIVER",
    "addressIp": "192.168.0.1",
    "sender": {
        "firstName": "Mark",
        "lastName": "Wards",
        "street": "Olszewskiego",
        "houseNumber": "17A",
        "city": "Lublin",
        "postalCode": "20-400",
        "flatNumber": "2",
        "email": "senderEmail@fenige.pl",
        "currency": "PLN",
        "expirationDate": "03/20",
        "personalId": "AGC688910",
        "cardId": "219708"
    },
    "receiver": {
        "firstName": "Rob",
        "lastName": "Wring",
        "currency": "PLN",
        "displayName": "Rob W.",
        "receiverType": "FRIEND_ID",
        "userId": "123"
    }
}
```

Receiver.receiverType = BARE_CARD_NUMBER.

```
POST /mobile-api/send-money HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56

{
    "amount": 1000,
    "cvc2": ["1","2","3"],
    "type": "RECEIVER",
    "addressIp": "192.168.0.1",
    "sender": {
        "firstName": "Mark",
        "lastName": "Wards",
        "street": "Olszewskiego",
        "houseNumber": "17A",
        "city": "Lublin",
        "postalCode": "20-400",
        "flatNumber": "2",
        "email": "senderEmail@fenige.pl",
        "currency": "PLN",
        "expirationDate": "03/20",
        "personalId": "AGC688910",
        "cardId": "219708"
    },
    "receiver": {
        "firstName": "Rob",
        "lastName": "Wring",
        "currency": "PLN",
        "card": ["5","1","4","2","3","3","3","6","2","9","5","2","3","7","3","2"],
        "displayName": "displayName",
        "phoneNumber": "48299000111",
      "receiverType": "BARE_CARD_NUMBER"
    }
  }
```

ExternalAuthentication.authenticationId.

```
POST /mobile-api/send-money HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56

{
  "amount" : 1000,
  "cvc2" : [ "1", "2", "3" ],
  "type" : "RECEIVER",
  "addressIp" : "192.168.0.1",
  "sender" : {
    "firstName" : "Mark",
    "lastName" : "Asdasd",
    "street" : "Olszewskiego",
    "houseNumber" : "17A",
    "city" : "Lublin",
    "postalCode" : "20-400",
    "flatNumber" : "2",
    "email" : "senderEmail@fenige.pl",
    "currency" : "PLN",
    "expirationDate" : "03/20",
    "personalId" : "AGC688910",
    "cardId" : "219708"
  },
  "receiver" : {
    "firstName" : "Rob",
    "lastName" : "Wring",
    "currency" : "PLN",
    "card" : [ "2", "1", "9", "7", "0", "8" ],
    "displayName" : "displayName",
    "phoneNumber" : "phoneNumber",
    "receiverType" : "WALLET_CARD_ID",
    "userId" : "123"
  },
  "externalAuthentication" : {
    "authenticationId" : "authenticationId"
  }
}
```

ExternalAuthentication.cavv, eci, transactionXId, authenticationStatus.

```
POST /mobile-api/send-money HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56

{
  "amount" : 1000,
  "cvc2" : [ "1", "2", "3" ],
  "type" : "RECEIVER",
  "addressIp" : "192.168.0.1",
  "sender" : {
    "firstName" : "Mark",
    "lastName" : "Asdasd",
    "street" : "Olszewskiego",
    "houseNumber" : "17A",
    "city" : "Lublin",
    "postalCode" : "20-400",
    "flatNumber" : "2",
    "email" : "senderEmail@fenige.pl",
    "currency" : "PLN",
    "expirationDate" : "03/20",
    "personalId" : "AGC688910",
    "cardId" : "219708"
  },
  "receiver" : {
    "firstName" : "Rob",
    "lastName" : "Wring",
    "currency" : "PLN",
    "card" : [ "2", "1", "9", "7", "0", "8" ],
    "displayName" : "displayName",
    "phoneNumber" : "phoneNumber",
    "receiverType" : "WALLET_CARD_ID",
    "userId" : "123"
  },
  "externalAuthentication" : {
    "cavv" : "jEu04WZns7pbARAApU4qgNdJTag",
    "eci" : "PLN",
    "authenticationStatus" : "Y",
    "transactionXId" : "9742432a-dfdc-41ca-9ae9-b6595de65f1d"
  }
}
```

Request headers

Type	Value	Constraints	Description
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.
X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Request fields

Response

Error response - ERROR_VALIDATION.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
```

```
Expires: 0
X-Frame-Options: DENY

{
    "traceId": "{{traceId}}",
    "errorStatus": "ERROR_VALIDATION",
    "message": "Some fields are invalid",
    "data": [
        {
            "field": "{{field_name_from_request}}",
            "message": "{{message}}"
        }
    ]
}
```

Error response - ERROR_BAD_TOKEN.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "traceId": "{{traceId}}",
    "errorStatus": "ERROR_BAD_TOKEN"
}
```

Error response - PRODUCT_NOT_FOUND.

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
```

```
X-Frame-Options: DENY

{
  "traceId": "{{traceId}}",
  "errorStatus": "PRODUCT_NOT_FOUND",
  "message": "Product by name {{product_name}} not found."
}
```

Error response - INTERNAL_SERVER_ERROR.

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "traceId": "{{traceId}}",
  "errorStatus": "INTERNAL_SERVER_ERROR"
}
```

Response fields

Errors

Encrypted response fields when sent header: *X-Encryption-Public-Key*

Http Status	**Error Status**	**Description**
400 - Bad Request	ERROR_VALIDATION	Some fields are invalid
400 - Bad Request	ERROR_BAD_TOKEN	Invalid authorization token
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error decoding public key has sent in header: *X-Encryption-Public-Key*
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error on decrypting request
400 - Bad Request	CRYPTOGRAPHY_ERROR	Error on encrypting response
400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE encryption Key is invalid

400 - Bad Request	CRYPTOGRAPHY_ERROR	JWE payload is expired
400 - Bad Request	ERROR_WHILE_GETTING_COUNTRY_CODE	Could not get card country code
400 - Bad Request	ERROR_MERCHANT_NOT_SUPPORT_CARD_PROVIDER	Merchant not support card provider
400 - Bad Request	ERROR_SENDER_CARD_NOT_ACTIVE	Sender card is not active
400 - Bad Request	ERROR_RECEIVER_CARD_NOT_ACTIVE	Receiver card is not active
400 - Bad Request	ERROR_SENDER_CARD_IS_BLOCKED	Sender card is blocked
400 - Bad Request	ERROR_RECEIVER_CARD_IS_BLOCKED	Receiver card is blocked
400 - Bad Request	UNKNOWN_ERROR	Unknown error
404 - Not Found	PRODUCT_NOT_FOUND	Product not found based on sent header: *Product-Name*
404 - Not Found	CANT_FIND_CARD	Not found card
404 - Not Found	FRIEND_NOT_EXISTS	Not found friend
500 - Internal Server Error	INTERNAL_SERVER_ERROR	Internal application error
500 - Internal Server Error	FENIGE_ERROR	Fenige error
500 - Internal Server Error	ERROR_ON_GETTING_DEFAULT_CARD	Error on getting card for friend

Examples

Add Friend

Request body with header: *X-Encryption-Public-Key*.

This method allow user to add Friend.

Request

friendType = WALLET.

```
POST /mobile-api/wallet-users/friends HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56
```

```

{
  "friendWalletDataCoreId": 1,
  "displayName": "Display name",
  "phoneNumber": "48999111222",
  "friendType": "WALLET",
  "firstName": "First",
  "lastName": "Last",
}
```

friendType = EXTERNAL.

```
POST /mobile-api/wallet-users/friends HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9nzW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 56

{
  "displayName": "Display name",
  "phoneNumber": "48999111222",
  "friendType": "EXTERNAL",
  "firstName": "First",
  "lastName": "Last",
  "cardNumber": ["5","5","2","7","4","7","9","6","6","8","3","9","0","9","5","7"]
}
```

Request headers

Request body with header: *X-Encryption-Public-Key*

Type	**Value**	**Constraints**	**Description**
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-response-body+json	Optional	Header must be present if the response body must have body.

Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.
X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Request fields

Response

Response fields

Examples

Get User friends

Request body with header: *X-Encryption-Public-Key*.

This method allow user to get all his friends

Request

Request headers

Encrypted request body with header: Content-Type: application/x-jwe-encryption-body+json

Type	Value	Constraints	Description
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name

X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Response

Response fields

Examples

Update Friend

Request body with header: *X-Encryption-Public-Key*.

This method allow user to update friend. For a friend of the type: WALLET, can update only the field: displayName. For a friend of the type: EXTERNAL, can update the fields: phoneNumber, displayName, firstName, lastName, cardNumber.

Request

friendType = WALLET.

```
PUT /mobile-api/wallet-users/friends/24 HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9naW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 101

{
  "displayName":"Display name"
}
```

friendType = EXTERNAL.

```
PUT /mobile-api/wallet-users/friends/24 HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9naW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 101

{
  "phoneNumber":"48999000111",
  "displayName":"Display name",
  "firstName":"First",
  "lastName":"Last",
  "cardNumber":["4","4","4","0","0","0","0","4","4","4","0","4","0"]
}
```

Request headers

Encrypted request body with header: *Content-Type: application/x-jwe-encryption-body+json*

Type	**Value**	**Constraints**	**Description**
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.

Request fields

Response

Examples

Delete friend

Encrypted request body with header: *Content-Type: application/x-jwe-encryption-body+json*.

This method allow user to delete friend

Request

Request headers

Type	Value	Constraints	Description
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name

Response

Examples

Get publicKey

This method allow user to get publicKey

Request

Request headers

Type	Value	Constraints	Description
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name

Response

Response fields

Examples

MC Send

Every single method should contains Authorization and Mobile-Product headers.

Master Card Send

Methods allow sending money in MasterCard Send 2.0

Request

Sender.paymentAccountType = WALLET_CARD_ID.

```
POST /mobile-api/mc-send HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9naW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 885

{
  "transactionId" : "bbb8597d-582c-4a12-a1c8-be9377aed6f9",
  "amount" : 40,
  "currency" : "INR",
  "sender" : {
    "account" : "walletCardId",
    "cvc2": ["3","2","1"],
    "addressId" : "123",
    "paymentAccountType" : "WALLET_CARD_ID"
  },
  "recipient" : {
    "name" : "Juniper Jane",
    "accountUri" : "402414000000006",
    "nationality" : "USA",
    "dateOfBirth" : "2011-05-13",
    "address" : {
      "city" : "Cape Girardeau",
      "country" : "USA",
      "state" : "MO",
      "postalCode" : "23232",
      "street" : "Mastercard Blvd"
    },
    "phone" : "1234567890",
    "email" : "jane.doe@mastercard.com",
    "governmentIds" : [ "123456789", "123456789" ],
    "receiverType" : "BARE_CARD_NUMBER"
  },
  "qrData" : "12",
  "transactionPurpose" : "07",
  "additionalMessage" : "message",
  "merchantCategoryCode" : "6536"
}
```

Sender.paymentAccountType = IBAN_ID.

```
POST /mobile-api/mc-send HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9naW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 885

{
  "transactionId" : "bbb8597d-582c-4a12-a1c8-be9377aed6f9",
  "amount" : 40,
  "currency" : "INR",
  "sender" : {
    "account" : "ibanId",
    "addressId" : "123",
    "paymentAccountType" : "IBAN_ID"
  },
  "recipient" : {
    "name" : "Juniper Jane",
    "accountUri" : "402414000000006",
    "nationality" : "USA",
    "dateOfBirth" : "2011-05-13",
    "address" : {
      "city" : "Cape Girardeau",
      "country" : "USA",
      "state" : "MO",
      "postalCode" : "23232",
      "street" : "Mastercard Blvd"
    },
    "phone" : "1234567890",
    "email" : "jane.doe@mastercard.com",
    "governmentIds" : [ "123456789", "123456789" ],
    "receiverType" : "BARE_CARD_NUMBER"
  },
  "qrData" : "12",
  "transactionPurpose" : "07",
  "additionalMessage" : "message",
  "merchantCategoryCode" : "6536"
}
```

Recipient.receiverType = WALLET_CARD_ID.

```
POST /mobile-api/mc-send HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9naW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 885

{
  "transactionId" : "bbb8597d-582c-4a12-a1c8-be9377aed6f9",
  "amount" : 40,
  "currency" : "INR",
  "sender" : {
    "account" : "ibanId",
    "addressId" : "123",
    "paymentAccountType" : "IBAN_ID"
  },
  "recipient" : {
    "name" : "Juniper Jane",
    "accountUri" : "4024",
    "nationality" : "USA",
    "dateOfBirth" : "2011-05-13",
    "address" : {
      "city" : "Cape Girardeau",
      "country" : "USA",
      "state" : "MO",
      "postalCode" : "23232",
      "street" : "Mastercard Blvd"
    },
    "phone" : "1234567890",
    "email" : "jane.doe@mastercard.com",
    "governmentIds" : [ "123456789", "123456789" ],
    "userId" : 13001,
    "receiverType" : "WALLET_CARD_ID"
  },
  "qrData" : "12",
  "transactionPurpose" : "07",
  "additionalMessage" : "message",
  "merchantCategoryCode" : "6536"
}
```

Recipient.receiverType = FRIEND_ID.

```
POST /mobile-api/mc-send HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9naW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 885

{
  "transactionId" : "bbb8597d-582c-4a12-a1c8-be9377aed6f9",
  "amount" : 40,
  "currency" : "INR",
  "sender" : {
    "account" : "ibanId",
    "addressId" : "123",
    "paymentAccountType" : "IBAN_ID"
  },
  "recipient" : {
    "name" : "Juniper Jane",
    "nationality" : "USA",
    "dateOfBirth" : "2011-05-13",
    "address" : {
      "city" : "Cape Girardeau",
      "country" : "USA",
      "state" : "MO",
      "postalCode" : "23232",
      "street" : "Mastercard Blvd"
    },
    "phone" : "1234567890",
    "email" : "jane.doe@mastercard.com",
    "governmentIds" : [ "123456789", "123456789" ],
    "userId" : 13001,
    "receiverType" : "FRIEND_ID"
  },
  "qrData" : "12",
  "transactionPurpose" : "07",
  "additionalMessage" : "message",
  "merchantCategoryCode" : "6536"
}
```

Recipient.receiverType = BARE_CARD_NUMBER.

```
POST /mobile-api/mc-send HTTP/1.1
Content-Type: application/json
Authorization: Mobile bG9naW46YWNrbWU=
Product-Name: TestProduct
Content-Length: 885

{
  "transactionId" : "bbb8597d-582c-4a12-a1c8-be9377aed6f9",
  "amount" : 40,
  "currency" : "INR",
  "sender" : {
    "account" : "ibanId",
    "addressId" : "123",
    "paymentAccountType" : "IBAN_ID"
  },
  "recipient" : {
    "name" : "Juniper Jane",
    "accountUri" : "402414000000006",
    "nationality" : "USA",
    "dateOfBirth" : "2011-05-13",
    "address" : {
      "city" : "Cape Girardeau",
      "country" : "USA",
      "state" : "MO",
      "postalCode" : "23232",
      "street" : "Mastercard Blvd"
    },
    "phone" : "1234567890",
    "email" : "jane.doe@mastercard.com",
    "governmentIds" : [ "123456789", "123456789" ],
    "receiverType" : "BARE_CARD_NUMBER"
  },
  "qrData" : "12",
  "transactionPurpose" : "07",
  "additionalMessage" : "message",
  "merchantCategoryCode" : "6536"
}
```

Request headers

Request body with header: *X-Encryption-Public-Key*

Type	**Value**	**Constraints**	**Description**
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.

X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Request fields

Response

errorStatus = INVALID_INPUT_FORMAT.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "traceId": "b4ce7ad5-758d-444f-90b3-ffbadb757e3f",
  "errorStatus": "INVALID_INPUT_FORMAT",
  "message": "Invalid Format",
  "data": {
    "error": [
      {
        "source": "recipient.accountURI.Expiration date",
        "reasonCode": "INVALID_INPUT_FORMAT",
        "errorDetailCode": "062000",
        "description": "Invalid Format"
      }
    ]
  }
}
```

A formal table with Reason Code

Error Detail Code	**Reason Code**	**Description**

062000	INVALID_INPUT_FORMAT	Value contains invalid character
072000	INVALID_INPUT_LENGTH	Invalid length
082000	INVALID_INPUT_VALUE	Invalid value
092000	MISSING_REQUIRED_INPUT	Value is required
110501	RESOURCE_ERROR	Duplicate value
110503	RESOURCE_ERROR	Account not eligible
110505	RESOURCE_ERROR	Invalid currency
110507	RESOURCE_UNKNOWN	Record not found
110510	RESOURCE_ERROR	Invalid Request
110537	RESOURCE_ERROR	Value is not supported for the merchant
130004	DECLINE	Per transaction maximum amount limit reached
130006	DECLINE	Transaction Limit is less than the minimum configured for the partner
130010	DECLINE	Partner not onboarded for the network to reach the account

errorStatus = ERROR_BAD_TOKEN.

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "traceId": "{{traceId}}",
    "errorStatus": "ERROR_BAD_TOKEN"
}
```

errorStatus = CANT_FIND_PAYMENT_TOKEN.

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
```

```
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "traceId": "89cdfc2b-346e-42d0-b20d-f3afa01cec68",
  "errorStatus": "CANT_FIND_PAYMENT_TOKEN",
  "message": "Payment token with given id was not found"
}
```

errorStatus = SYSTEM_ERROR.

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "traceId": "1c8d4f1f-16db-4c43-bdce-0fe43ae39195",
  "errorStatus": "SYSTEM_ERROR",
  "message": "Internal exception occurred.",
  "data": {
    "error": [
      {
        "source": "SYSTEM",
        "reasonCode": "SYSTEM_ERROR",
        "errorDetailCode": null,
        "description": "Internal exception occurred."
      }
    ]
  }
}
```

Response fields

Examples

Authentication

Every single method should contains Authorization and Mobile-Product headers.

Init Authentication

The authentication stage flow is indicated by the following field: threeDsMode

Method allows us to do initialize authentication using ThreeDs 2.0 protocol.

After this method you have 3 options:

- **FRICTIONLESS** - In response: authenticationStatus, transactionXId, cavv, eci and **threeDsMode = FRICTIONLESS** are present. This response denotes that authentication was finished.
- **ThreeDsMethod flow** - In response: threeDsMethodData and **threeDsMode = THREE_DS_METHOD** are present. This response denotes that you should perform ThreeDs method flow. After executing ThreeDs method flow, make a request for the method: Continue Authentication
- **CHALLENGE** - In response: acsUrl, creq, challengeHtmlFormBase64 and **threeDsMode = CHALLENGE** are present. This response denotes that you should perform challenge. After executing challenge, make a request for the method: Finalize Authentication

Request

Request headers

Request body with header: *X-Encryption-Public-Key*

Type	Value	Constraints	Description

Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.
X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Request fields

Response

threeDsMode = FRICTIONLESS.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "authenticationId": "authenticationId",
  "authenticationStatus": "Y",
  "transactionXId": "9742432a-dfdc-41ca-9ae9-b6595de65f1d",
  "cavv": "jEu04WZns7pbARAApU4qgNdJTag",
  "eci": "02",
  "threeDsMode": "FRICTIONLESS"
}
```

threeDsMode = THREE_DS_METHOD.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "authenticationId": "authenticationId",
  "threeDsMethodData":
"eyJ0aHJlZURWZpY2F0aW9uVVJMIjoiaHR0cHM6Ly93ZWJob29rLnNpdGUvc3MiLCJ0aHJlZURTU2VydmVyVHJhbnNJR
CI6IjNmYWYwZjFZi1iYjQyLThkN2RhM2M0NjY5OSJ9",
  "threeDsMethodUrl": "https://threeDsMethodUrl-test.verestro.com/acs-mock",
  "threeDsMode": "THREE_DS_METHOD"
}
```

threeDsMode = CHALLENGE.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
  "authenticationId": "authenticationId",
  "acsUrl": "https://acs-url.verestro.com/mock-acs",
  "creq":
"eyJjYXJkQXV0aGVudGljYNmODlhLTk2MjQtNGQ1OS04NzZmLTNkMWViYTcyNzM3NiIsIm5vdGlmaWNhdGlvblVybvd
2ViaG9vay5zaXRlLzE5ODI3MWMyLTljYWYtNGEyMy05ZGJiLWRlZTc3ODExMDdlOSIsInRocmVlRFNTZXJ2ZXJUcmFuc
0lEIjoiM2ZhZjBmMWQtM2YxNy00MTJmLWJiNDItOGQ3ZGEzYzQ2Njk5IiwibWVzc2FnZVZlcnNpb24iOiIyLjEuMCJ9",
  "challengeHtmlFormBase64":
"PGh0bWw+PFNDUklQVCBMQU5mF2YXNjcmlwdCI+ZnVuY3Rpb24gT25Mb2FkRXZlbW1lbnQuZG93bmxvYWRGb3J
tLnN1Ym1pdCgpOyB9PC9TQ1JJUFQ+PGJvZHkgT25Mb2FkmVudCgpOyI+PGZvcm0gbmFtZT0iZG93bmxvYWRGb3Jt
IiBhY3Rpb249Imh0dHBzOi8vbXBpLXN0YWdpbmcuZmVuaWdlLnBsL21vY2stYWNzIiBtZXRob2Q9IlBPU1QiPjxJTlBVV
CB0eXBlPSJoaWRkZW4iXEiIHZhbHVlPSJleUpqWVhKa1FYVjBhR1Z1ZEdsallYUnBiMjVKWkNJNkltRmpZbU5tT0RsaEx
UazJNalF0TkdRMU9TMDROelptTFROa01XVmlZVGN5TnpNM05pSXNJbTV2ZEdsbWFXTmhkR2x2YmxWeWJDSTZJbW
gwZEhCek9pOHZkMlZpYUc5dmF5NXphWFJsTHpFNU9ESTNNV015TFRsallXWXROR0V5TXkwNVpHSmlMV1JsWlRjM
09ERXhNRGRsT1NJc0luUm9jbVZsUkZOVFpYSjJaWEpVY21GdWMwbEVJam9pTTJaaFpqQm1NV1F0TTJZeE55MDBNV
EptTFdKaU5ESXRPR1EzWkdFell6UTJOams1SWl3aWJXVnpjMkZuWlZabGNuTnBiMjRpT2lJeUxqRXVNQ0o5Ij48SU5Q
VVQgdHlwZT0iaGlkZGVuIiBuYW1lPSJ0aHJlZURTU2Vzc2lvbkRhdGEiIHZhbHVlPSJZV05pWTJZNE9XRXRPVFl5TkMwM
FpEVTVMVGczTm1ZdE0yUXhaV0poTnpJM016YzJOVGsyIiB2YWx1ZT0iMSIgLz48L2Zvcm0+PC9ib2R5PjwvaHRtbD4=",
    "threeDsSessionData": "YWNiY2Y4OWEtONC00ZDU5LTg3NmYtM2QxZWJhNzI3Mzc2",
    "threeDsMode": "CHALLENGE"
}
```

Response fields

Base response fields

Path	Type	Description
authenticationId	String	Unique authentication identifier
threeDsMethodData	String	Encoded data used for request to ACS
threeDsMethodUrl	String	ACS endpoint for hidden request. If endpoint is not present then request is not required.

authenticationStatus	String	Indicates whether a transaction qualifies as an authenticated transaction or account verification. Possible values are: **Y** - Authentication/account verification successful **N** - Not authenticated/account not verified; transaction denied **U** - Authentication/account verification could not be performed; technical or other problem as indicated in ARes or RReq **A** - Attempts processing performed; not authenticated/verified, but a proof of attempted authentication/verification is provided **C** - Challenge required; additional authentication is required using the CReq/CRes **R** - Authentication/account verification rejected; issuer is rejecting authentication/verification and request that authorization not be attempted **D** - Challenge required; decoupled authentication confirmed **I** - Informational only; ThreeDs Requestor challenge preference acknowledged The CRes message can contain only a value of Y or N. Values of D and I are only applicable for ThreeDs version 2.2.0.
transactionXId	String	This field indicates the transactionXid from recurring initial authentication.
cavv	String	This property is determined by the Access Control Server. This property will be valid if the TransactionStatus is "Y" or "A". The value may be used to provide proof of authentication.
eci	String	This property is determined by the Access Control Server. This property contains the two digit Electronic Commerce Indicator (ECI) value, which is to be submitted in a credit card authorization message. This value indicates to the processor that the customer data in the authorization message has been authenticated. The data contained within this property is only valid if the TransactionStatus is "Y" or "A".

acsUrl	String	If challenge is required, data for building a form such as challengeHtmlFormBase64
creq	String	If challenge is required, data for building a form such as challengeHtmlFormBase64
challengeHtmlFormBase64	String	This field is a BASE64 encrypted html source file containing the challenge 3-D Secure frame
threeDsSessionData	String	ThreeDsSessionData value
threeDsMode	String	ThreeDs process mode which informs about. One of: [FRICTIONLESS, THREE_DS_METHOD, CHALLENGE] **FRICTIONLESS** - this is where the authentication process was finished. **THREE_DS_METHOD** - next step is to execute the ThreeDs method process. After it is done, we need to make a request to the method: Continue Authentication **CHALLENGE** - next step is to execute the challenge process. After it is done, we need to make a request to the method: Finalize Authentication

Examples

Errors

Request body with header: *X-Encryption-Public-Key*

Http Status	**Error Status**	**Description**
400 - Bad Request	PROCESS_NOT_ALLOWED	Method not allowed - invoke calculate commission method is necessary first.
400 - Bad Request	ERROR_SENDER_CARD_NOT_ACTIVE	Sender card is not active

Continue Authentication

The authentication stage flow is indicated by the following field: threeDsMode

Method allows us to do continue authentication using ThreeDs 2.0 protocol. Use this method after perform process ThreeDsMethod. This step is optional in the authentication process. Required only if ThreeDsMethod case is present.

After this method you have 2 options:

- **FRICTIONLESS** - In response: authenticationStatus, transactionXId, cavv, eci and **threeDsMode = FRICTIONLESS** are present. This response denotes that authentication was finished.
- **CHALLENGE** - In response: acsUrl, creq, challengeHtmlFormBase64 and **threeDsMode = CHALLENGE** are present. This response denotes that you should perform challenge. After executing challenge, make a request for the method: Finalize Authentication

Request

Request headers

Request body with header: *X-Encryption-Public-Key*

Type	Value	Constraints	Description
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.
X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Request fields

Response

threeDsMode = FRICTIONLESS.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
```

```
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "authenticationId": "authenticationId",
    "authenticationStatus": "Y",
    "transactionXId": "9742432a-dfdc-41ca-9ae9-b6595de65f1d",
    "cavv": "jEu04WZns7pbARAApU4qgNdJTag",
    "eci": "02",
    "threeDsMode": "FRICTIONLESS"
}
```

threeDsMode = CHALLENGE.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY

{
    "authenticationId": "authenticationId",
    "acsUrl": "https://acs-url.verestro.com/mock-acs",
    "creq":
"eyJjYXJkQXV0aGVudGljYNmODlhLTk2MjQtNGQ1OS04NzZmLTNkMWViYTcyNzM3NiIsIm5vdGlmaWNhdGlvblVybvd
2ViaG9vay5zaXRlLzE5ODI3MWMyLTljYWYtNGEyMy05ZGJiLWRlZTc3ODExMDdlOSIsInRocmVlRFNTZXJ2ZXJUcmFuc
0lEIjoiM2ZhZjBmMWQtM2YxNy00MTJmLWJiNDItOGQ3ZGEzYzQ2Njk5IiwibWVzc2FnZVZlcnNpb24iOiIyLjEuMCJ9",
    "challengeHtmlFormBase64":
"PGh0bWw+PFNDUklQVCBMQU5mF2YXNjcmlwdCI+ZnVuY3Rpb24gT25Mb2FkRXZlbW1lbnQuZG93bmxvYWRGb3J
tLnN1Ym1pdCgpOyB9PC9TQ1JJUFQ+PGJvZHkgT25Mb2FkmVudCgpOyI+PGZvcm0gbmFtZT0iZG93bmxvYWRGb3Jt
IiBhY3Rpb249Imh0dHBzOi8vbXBpLXN0YWdpbmcuZmVuaWdlLnBsL21vY2stYWNzIiBtZXRob2Q9IlBPU1QiPjxJTlBVV
CB0eXBlPSJoaWRkZW4iXEiIHZhbHVlPSJleUpqWVhKa1FYVjBhR1Z1ZEdsallYUnBiMjVKWkNJNkltRmpZbU5tT0RsaEx
UazJNalF0TkdRMU9TMDROelptTFROa01XVmlZVGN5TnpNM05pSXNJbTV2ZEdsbWFXTmhkR2x2YmxWeWJDSTZJbW
gwZEhCek9pOHZkMlZpYUc5dmF5NXphWFJsTHpFNU9ESTNNV015TFRsallXWXROR0V5TXkwNVpHSmlMV1JsWlRjM
09ERXhNRGRsT1NJc0luUm9jbVZsUkZOVFpYSjJaWEpVY21GdWMwbEVJam9pTTJaaFpqQm1NV1F0TTJZeE55MDBNV
EptTFdKaU5ESXRPR1EzWkdFell6UTJOams1SWl3aWJXVnpjMkZuWlZabGNuTnBiMjRpT2lJeUxqRXVNQ0o5Ij48SU5Q
VVQgdHlwZT0iaGlkZGVuIiBuYW1lPSJ0aHJlZURTU2Vzc2lvbkRhdGEiIHZhbHVlPSJZV05pWTJZNE9XRXRPVFl5TkMwM
FpEVTVMVGczTm1ZdE0yUXhaV0poTnpJM016YzIiPjwvZm9ybT48L2JvZHk+PC9odG1sPg==",
    "threeDsSessionData": "YWNiY2Y4OWEtONC00ZDU5LTg3NmYtM2QxZWJhNzI3Mzc2",
    "threeDsMode": "CHALLENGE"
}
```

Response fields

Base response fields

Path	Type	Description
authenticationId	String	Unique authentication identifier
authenticationStatus	String	Indicates whether a transaction qualifies as an authenticated transaction or account verification. Possible values are: **Y** - Authentication/account verification successful **N** - Not authenticated/account not verified; transaction denied **U** - Authentication/account verification could not be performed; technical or other problem as indicated in ARes or RReq **A** - Attempts processing performed; not authenticated/verified, but a proof of attempted authentication/verification is provided **C** - Challenge required; additional authentication is required using the CReq/CRes **R** - Authentication/account verification rejected; issuer is rejecting authentication/verification and request that authorization not be attempted **D** - Challenge required; decoupled authentication confirmed **I** - Informational only; ThreeDs Requestor challenge preference acknowledged The CRes message can contain only a value of Y or N. Values of D and I are only applicable for ThreeDs version 2.2.0.
transactionXId	String	This field indicates the transactionXid from recurring initial authentication.

cavv	String	This property is determined by the Access Control Server. This property will be valid if the TransactionStatus is "Y" or "A". The value may be used to provide proof of authentication.
eci	String	This property is determined by the Access Control Server. This property contains the two digit Electronic Commerce Indicator (ECI) value, which is to be submitted in a credit card authorization message. This value indicates to the processor that the customer data in the authorization message has been authenticated. The data contained within this property is only valid if the TransactionStatus is "Y" or "A".
acsUrl	String	If challenge is required, data for building a form such as challengeHtmlFormBase64
creq	String	If challenge is required, data for building a form such as challengeHtmlFormBase64
challengeHtmlFormBase64	String	This field is a BASE64 encrypted html source file containing the challenge 3-D Secure frame
threeDsSessionData	String	ThreeDsSessionData value
threeDsMode	String	ThreeDs process mode which informs about. One of: [FRICTIONLESS, CHALLENGE] **FRICTIONLESS** - this is where the authentication process was finished. **CHALLENGE** - next step is to execute the challenge process. After it is done, we need to make a request to the method: Finalize Authentication

Examples

Finalize Authentication

Request body with header: *X-Encryption-Public-Key*.

Method allows us to do finalize authentication using ThreeDs 2.0 protocol.

Request

Request headers

Request body with header: *X-Encryption-Public-Key*

Type	Value	Constraints	Description
Authorization	Mobile bG9naW46YWNrbWU=	Required	Device token with "Mobile " prefix
Product-Name	TestProduct	Required	Application product name
Content-Type	application/x-jwe-encryption-body+json	Optional	Header must be present if the request body is encrypted using the JWE standard.
X-Encryption-Public-Key		Optional	Header must be present if the response body is to be encrypted using the JWE standard. Public key must be encoded Base64.

Request fields

Response

Response fields

Base response fields

Path	Type	Description
authenticationId	String	Unique authentication identifier

authenticationStatus	String	Indicates whether a transaction qualifies as an authenticated transaction or account verification. Possible values are: **Y** - Authentication/account verification successful **N** - Not authenticated/account not verified; transaction denied **U** - Authentication/account verification could not be performed; technical or other problem as indicated in ARes or RReq **A** - Attempts processing performed; not authenticated/verified, but a proof of attempted authentication/verification is provided **C** - Challenge required; additional authentication is required using the CReq/CRes **R** - Authentication/account verification rejected; issuer is rejecting authentication/verification and request that authorization not be attempted **D** - Challenge required; decoupled authentication confirmed **I** - Informational only; ThreeDs Requestor challenge preference acknowledged The CRes message can contain only a value of Y or N. Values of D and I are only applicable for ThreeDs version 2.2.0.
transactionXId	String	This field indicates the transactionXid from recurring initial authentication.
cavv	String	This property is determined by the Access Control Server. This property will be valid if the TransactionStatus is "Y" or "A". The value may be used to provide proof of authentication.
eci	String	This property is determined by the Access Control Server. This property contains the two digit Electronic Commerce Indicator (ECI) value, which is to be submitted in a credit card authorization message. This value indicates to the processor that the customer data in the authorization message has been authenticated. The data contained within this property is only valid if the TransactionStatus is "Y" or "A".

Examples

Technical Documentation

SDK documentation Android

The Money Transfer Android SDK specification is divided into 3 main components (SDK's) listed in table below:

SDK link	Description
Receivers	This SDK is responsible for managing recipients
Transfers	This SDK is responsible for managing money transfer
QR	This SDK is responsible for processing and generating QR codes

Technical Documentation

SDK documentation iOS

The Money Transfer iOS SDK specification is divided into 3 main components (SDK's) listed in table below:

SDK link	Description
Receivers	This SDK is responsible for managing recipients
Transfers	This SDK is responsible for managing money transfer
QR	This SDK is responsible for processing and generating QR codes