Overview

Verestro Cloud Payments is a solution developed to facilitate adopting cloud-based payments for the Customers. VCP provides functionalities for User identification and verification, Payment Instruments digitization and User data management. Cloud payments enables a card to be digitized into a wallet application on a mobile device and used for payment without the need for a Secure Element (SE) or a Trusted Execution Environment (TEE) to protect the card's sensitive assets, such as the keys needed for calculating the Application Cryptogram. Master Keys for the digitized card are kept securely on remote servers (for plastic in the chip), hence the term 'cloud-based payment,' and a limited number of keys (where each key can only be used to perform a single transaction) are downloaded to the application.

Solution Components

Server components

- Wallet Server - backend component,
- Wallet Admin Panel - frontend component

Mobile components

- Wallet SDK - Android and iOS libraries,
- Wearable SDK - libraries for payments with Huawei Smartwatches

Benefits of Payment Token

The MCBP service is an easy and secure way to replace a plastic payment card with a payment token. Recognition to the tokenization and digitization process without leaving the house, we can add our payment card to the mobile application and use only a mobile device during purchases.

The benefits of tokenization are felt by every participant in the process:

- **Issuer** - by implementing the tokenization service, it will provide its customers with much higher and safer access to innovative payment solutions.
- **Card Holder** - can freely use innovative payment solutions. The tokenization service will allow free and secure payments using any devices connected to the internet.

MCBP Introduction

Mastercard Cloud-Based Payments (MCBP) is technology which enables a card to be digitized into an application on a mobile device. On plastic card, Master Keys needed for calculating cryptogram are stored in the Secure Element. Using MCBP there is no need for Secure Element since keys needed for calculating the cryptogram are kept securely on remote servers, hence the term 'cloud-based payment,' and a limited number of keys (where each key can only be used to perform a single transaction) are downloaded to the application. VCP is solution which provides functionalities for digitization, user data management and payments needed for final Customer to adopt MCBP.

MCBP High Level Architecture

MCBP High Level Architecture

MCBP Key Components

Component	Description
MPA	Mobile Payment Application (for Android and iOS Smartphones) provides frontend interface to the user and uses part of Verestro Wallet SDK which is responsible for payments using HCE. In case of wearable payments, MPA acts as companion app.
Verestro Wallet Server	Provides the backend services to support Mobile Payment Application via Verestro Wallet SDK and is responsible for managing users, devices, cards, Payment Tokens and communication with MDES. Wallet Server acts as Token Requestor on behalf of Issuer in context of digitization.
Verestro Wallet SDK	Provides all functionalities needed for MPA to perform all needed operations related to MCBP.
MDES	Token Service Provider which supports digitization (transforming the card into Payment Token) and is responsible for management, generation and provisioning of transaction credentials into mobile devices to enable simpler and more secure digital payment experience.
Remote Notification Service	Wallet Server communicates with the MPA also using Remote Notification Service. For Android is used Firebase Cloud Messaging.

Issuer	Issuer is responsible for card issuing, accepting authorization digitization requests and accepting transactions which uses token.

Description

Wallet Types

VCP supports following wallet types which can be used in the implementation:

- **OPEN** - user registers itself in the application and provides data like PAN etc.,
- **CLOSED** - user data are passed automatically from Customer servers without User interaction to Wallet Server.

Implementation Models

Verestro provides two different implementation models for products: **integrated** and **standalone** version.

Integrated

In this model Customer is owner of MPA. Verestro provides Wallet SDK and Wallet Server. Customer is responsible for direct User authentication and passes the result of the authentication to Wallet SDK. Online operations which need to be performed by User using Wallet Server require valid session on Wallet Server. To obtain user online session with Wallet Server, Customer needs to pass Trusted Identity.

Standalone

In this model Verestro provides MPA, Wallet SDK and Wallet Server. Furthermore, Verestro manages direct user authentication.

Architecture

Architecture

Server Components

Server components are applications which need to be deployed on remote server to make possible to connect them by network.

Deployment Models

Verestro offers two deployment models of server components: **On-premise** and **SaaS**.

On-premise

Server components also can be deployed on Customer infrastructure. Applications are designed to be deployed using Kubernetes as system for automating deployment, scaling, and management of containerized applications. For more details please contact Verestro representative.

SaaS

Server components are designed to be deployed in SaaS model. In this case everything is deployed and configured on Verestro side. Verestro is responsible for maintaining infrastructure, deploying applications and monitoring.

Wallet Server

Wallet Server is a backend component which consists of few internal services which are responsible for managing users, devices, cards, Payment Tokens, transaction history. It acts as Token Requestor on behalf of Issuer and is compliant with PCI Data Security Standard.

It exposes:

- **mobile API** - available via Wallet SDK to perform operations directly from mobile device,
- **LC API** - server API dedicated for Issuer to manage users and cards data on Wallet Server,
- **MDES Outbound API** - server API dedicated for MDES,
- **Verestro Cloud Payments External API** - dedicated for external clients (e.g. Issuer) to manage Payment Tokens,
- **Transaction History API** - API to receive and collect information on performed financial transactions.

Wallet Server operates with domain objects like:

- **User** - root of entity tree. User is identified in Wallet Server via some unique identifier which can be external id given by Customer. User can have access to his data and operations based on session. Session is created after pairing device and when is expired then User authentication needs to be performed. Session is valid for a configurable period of time.
- **Device** - belongs to User. When User starts using application after installation then device pairing is performed. After pairing device with some unique id (constant across installations and users), unique device installation id is generated and this installation is

assigned to particular User. It is possible to have one active installation on specific device for specific User. If other User starts using application on same device then another device pairing is performed and all data from previous installation will be wiped.
- **Card** - belongs to 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 (always or short period of time).
- **Payment Token** - after PAN digitization, device Payment Token is created also on Wallet Server side without any sensitive data. One PAN can have one device Payment Token on specific device installation at the same time which is INACTIVE, ACTIVE or SUSPENDED.

Wallet Server

Wallet Admin Panel

Web frontend application which is dedicated for back office to manage all User data.

Mobile Components

Wallet SDK

Verestro provides Software Development Kit (SDK) called Wallet SDK which can be used in Mobile Payment Application. 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 Cloud Payments: **MDC SDK** and **VCP SDK**.

MDC SDK is core Verestro module responsible for user data management: authentication, payment cards management - since these are main functionalities used in every product. VCP SDK is dedicated for performing digitization and payments using Payment Token. In payment context VCP SDK wraps Mastercard Cloud Based Payment SDK.

Wearables SDK

Product supports contactless payments using Huawei smartwatches. There are multiple SDKs to be implemented both - on smartphone and wearable device. Smartphone app serves as companion app in this process. Wearable payments are fully compliant with smartphone-only solution, and act alongside it. Wearables SDK is extension, but uses wallet SDK even if smartphone payments are not necessary in project.

Requirements

> **Important!** Wallet SDK has some mandatory requirements to make it work:
>
> - device cannot be rooted,

- Android OS image (ROM) should be genuine in version 6.0 (Marshmallow) or above,
- devices cannot have enabled debugging.

There are also some **not mandatory** requirements, but Customer needs to be aware of them to maintain functionalities:

- NFC module necessary for HCE payments,
- lock screen necessary for locally-verified user authentication.

Security

Wallet SDK was developed according to security requirements included in Security Guide MCBP SDK for Android. However Wallet SDK cannot guarantee full MPA protection and MPA must provide additional layer of security to protect user interface (mainly when PAN is manually entered in the application) and data processing within application. More detailed information can be found in *Wallet SDK API.* Moreover all sensitive data are passed as chars or bytes arrays. Wallet SDK copies the arrays and clears that copies just after processing. MPA should clear provided sensitive data immediately after passing them to Wallet SDK.

Security Checks and Data Clearing

On Wallet SDK side are performed security checks which includes static code analysis protection and dynamic analysis protection. Security checks consists of:

- root access detection,
- hooking protection,
- debugging protection,
- custom ROM protection,
- data tampering protection,
- man in the middle protection.

Security checks are performed periodically, if Wallet SDK detects any of above things all data hold by Wallet SDK will be cleared and security report will be sent to Wallet Server. MPA will be informed about such detection.

Communication with Wallet Server

Communication from genuine applications which are installed on genuine devices is accepted by the Wallet Server. Wallet SDK at the very beginning performs authentication of application and device to Wallet Server. This authentication may take advantage of Google Play Integrity which is a 3rd-party trusted side in whole authentication. Google Play Integrity verifies device and sign information about device and application. Signed data from Play Integrity are sent to the Wallet Server. Wallet Server verifies data and allow or does not allow for further communication. Application is verified according preconfigured application certificate digest used for signing application.

> **Important!** There is a limit of requests to Google Play Integrity API: 10 000 per day. If Customer predicts that there will be more installations per day then this limit needs to be increased during Google Project Setup.

Wallet SDK communicates with Wallet Server using TLS 1.2. Wallet SDK performs public key certificate pinning when it tries to establish connection with Wallet Server (similar with connection to MDES). Certificates for the pinning needs to be provided to SDK. Sensitive information are additionally encrypted and/or signed.

Versioning

Wallet SDK uses semantic versioning. It means that every release has own version which is MAJOR.MINOR.PATCH, where:

- MAJOR version increases when SDK has incompatible API changes,
- MINOR version increases when new functionality is added in a backwards compatible manner,
- PATCH version increases when new backwards compatible bug fixes are introduced.

MAJOR versions are supported 6 months and Customer needs to migrate to new version if they want to maintain support.

Remote Notification Service

Wallet SDK is responsible for remote notification processing. However MPA is responsible for obtaining FCM registration token, handling FCM token update and receiving remote notifications. Before passing remote notification to SDK, MPA needs to verify if given message is dedicated for SDK by checking sender Id. Sender Id is configured during onboarding. Verestro will create new FCM project and provide data needed to obtain FCM token for given project. Due to observing some issues with FCM token refresh notification from FCM service, additional check of new token availability is recommended (e.g. on application start). See more in *Wallet SDK API.*

Access

Wallet SDK is stored as artifacts in maven repository. Access there is provided during onboarding by Verestro representative.

Configuration

Whole product has configuration which needs to be fulfilled. This configuration also consists of data which are set in MDES. More details are described in *Wallet Configuration*.

User Experience for Contactless Transactions

MDES offers a few options for customers for defining user experience for contactless transactions. Final option is the choice of CDCVM type (Mobile PIN, Locally-verified CDCVM) and CVM model (Always, Flexible, Card-like).

CDCVM Types

There are two types of Consumer Device Cardholder Verification Method (CDCVM) which are supported by MDES.

Mobile PIN CDCVM

A PIN value (4–8 digits) that the cardholder enters on the mobile device and that is validated online by MDES during the transaction authorization process. Since Locally-verified is mostly preferable option, Mobile PIN is out of scope.

Locally-verified CDCVM

A CDCVM entered on and validated by the consumer's mobile device, for example system device PIN, pattern, password or biometric methods (such as fingerprint, iris or facial recognition). Swipe (slide to unlock) is not a valid cardholder verification method and must not be supported. These methods are commonly associated with a device unlock process and are validated on the cardholder's mobile device. The payment component embedded in the Mobile Payment Application will use the outcome of this authentication process. A Locally-verified CDCVM applies to all the payment tokens of a given Mobile Payment Application instance ("Wallet-level"). In some parts of system this type is also named as Custom.

CVM Models

For two CDCVM types customer can apply different user experience CVM Models.

Always CVM

In this model the card profiles supplied to the SDK are configured to indicate that the mobile device supports on-device cardholder authentication. When transactions are performed on a POS supporting CDCVM, the POS will delegate cardholder authentication to the mobile device the terminal will not request an Online PIN on the terminal. In POS which does not support CDCVM cardholder authentication is required using Online PIN. This model requires the cardholder's mobile device to authenticate the cardholder for all transactions (LVT, HVT, Transit). CDCVM can be performed using either a Mobile PIN or a Locally-verified CDCVM. MPA is expected to decline any transaction for which cardholder authentication is not performed or is unsuccessful.

Below are presented sample diagrams which show how the transactions can look like:

Always LVT, HVT - single tap

Always LVT, HVT - single tap

Always LVT, HVT - double tap

Always LVT, HVT - double tap

Flexible CVM

In this model the card profile also indicates that the mobile device supports on device cardholder authentication Mobile PIN or Locally-verified CDCVM. However rather than applying authentication for every transaction the MPA defines flexible criteria such as allowing multiple transactions between each authentication. This criteria are often named as Lost & Stolen options or velocity checks. For transit transactions cardholder authentication is not expected.

Below are presented sample diagrams which show how the transactions can look like:

Flexible LVT

Flexible LVT

Flexible HVT - single tap

Flexible HVT - single tap

Flexible HVT, LVT with Velocity counters - double tap

Flexible HVT, LVT with Velocity counters - double tap

Card-like CVM

In this model the card profiles supplied to the SDK are configured to indicate that the mobile wallet is not capable of supporting on device cardholder verification. This means that when transactions are performed with a Point of Sale (POS) terminal, the POS will treat the transaction in the same way as a card transaction. Typically this means that low value transactions (LVTs) will be processed without additional user authentication and if supported, high value transactions will require an online PIN to be provided on POS. This model is put here just for general information, however it is not preferred for issuer wallets.

Below are presented sample diagrams which show how the transactions can look like:

Card like LVT

Card like LVT

Card like HVT

Card like HVT

Lost & Stolen Options

The Lost & Stolen options are dedicated to control performing transactions allowed before requiring cardholder authentication. This limits fraud risk if the cardholder's mobile device is lost or stolen. Lost & Stolen options can be applied only for Flexible CDCVM. These options also are known as velocity check counters. Wallet SDK provides interface which is invoked during transaction. Transaction information like range, rich transaction type, amount are provided within this interface. MPA can implement various checks to support velocity check counters using transaction information. MPA for example can count LVT transactions and allow only some predefined number of LVT transactions without cardholder authentication.

Transit Transactions

Transit transactions are transactions with given Merchant Category Code performed e.g. on traffic gates like: underground. It is up to Customer if wants to enable such transactions or not (option selected during MDES onboarding). Transit transactions are enabled in every CVM model, however for Always CDCVM needs to be performed for every transaction, for Card-Like and Flexible CDCVM can be skipped.

Tokenization/Digitization

Tokenization is a process which enable to replace sensitive data, e.g. card number, with another string of characters - a secure payment token - as a result of which card data remains inaccessible. Payment tokens are assigned to a given device of the card owner, which means that an unauthorized person, even if he obtains the token data, will not be able to use them via another device. It is also secured inside the SDK. As a result of the tokenization process, the customer comes into possession of Transaction Credentials in a mobile application on his device.

Tokenization/Digitization

Digitization decisions

Green

APPROVED

Approve the digitization request without further cardholder interaction.

Yellow

REQUIRE_ADDITIONAL_AUTHENTICATION

Approve the digitization request but seek additional cardholder authentication.

Red

DECLINED

Decline the digitization.

Main processes

User and cards registration into wallet server

User with unique identifier known on issuer side is registered along with cards in the Wallet Server. After that process, device can be paired.

Pairing device for particular user by trusted identity

Authentication of the device in context of given user. Process needed to allow access to wallet server from that particular device.

In the integrated model signed user identifier passed from issuer into wallet server via SDK is used to authenticate given user.

Digitization of the card

When device is paired user can have access to his own data: cards. After that he can digitize chosen card which was previously added into wallet server.

Two digitization approaches are supported:

One-Step Digitization

Simplified process for implementations requiring no additional authentication. Expects **APPROVED** or **DECLINED** outcome only.

Multi-Step Digitization

Includes eligibility checking, terms & conditions display, and digitization. May result in **APPROVED**, **DECLINED**, or **REQUIRE_ADDITIONAL_AUTHENTICATION** outcome.

Digitized card profile provisioning on the device

After successful digitization, digitized card profile is delivered to device. Since only limited number of keys for transaction is delivered to the device, SDK triggers replenishment to cover up the number of transaction credentials.

Transaction credentials replenishment

Transaction Credentials are unique per-transaction keys used to calculate cryptograms. Each set of credentials can only be used for one transaction.

Three replenishment strategies are supported:

Automatic

After every transaction, Wallet SDK checks if the number of transaction credentials has fallen below a preconfigured threshold. If so, replenishment is triggered automatically.

Initial

Occurs directly after successful token activation without requiring any MPA action.

Manual

Allows explicit user-initiated credential refresh when automatic processes fail, particularly when internet connectivity is limited.

Payment

Solution delivers multiple options to process and complete payments. Contactless smartphone payments, wearable contactless payments, DSRP, one-tap, two-tap.

Payment history (Optional)

It is possible to store transactions on wallet server. Retrieving data is possible using one of SDK functionalities or dedicated API methods.

Main steps and implementation

Issuer integration with MDES.

Completing BPMS/ICG file - configuration for issuers in MDES:

- PAN ranges allowed for digitization,
- channel for the authorization cards for the digitization: predigitization API/ISO 8583 messages,
- digitization path,

Completing PCG file - configuration for token requestors (wallet configuration):

- parameters for connection,
- transactions user experience,

Exchanging certificates:

- connection,
- external wrapping key,
- payload encryption to mdes,
- tav,

Adopt LC API to pass users and cards into wallet server and manage them later

Integrate SDK into issuer application

Implement signing user identity to authenticate user on wallet server via SDK

Revision #24
Created 15 April 2026 13:46:09 by Beata Rzemieniak
Updated 18 May 2026 14:46:50 by Beata Rzemieniak