



4G MDK インストールガイド

VeriTrans 4G

MDK for PHP8 Installation Guide
ver.1.0.3 (2025年10月～)

目次

1. 導入の前に.....	2
1.1 本ガイドの内容	2
1.2 著作権、および問い合わせ先	2
2. 導入準備.....	3
2.1 基本導入手順.....	3
2.2 実行環境の確認と準備	3
2.2.1 注意事項	4
2.3 PHP 用サポート・ライブラリの組込(有効化)	4
3. 設定方法.....	5
3.1 MDK の確認.....	5
3.2 MDK の設定.....	5
3.3 アプリケーションへの MDK の組み込み	7
4. サンプルプログラムの利用.....	8
4.1 サンプルプログラム(コマンドラインプログラム) 設定・動作確認.....	8
4.2 サンプルプログラム(Web アプリケーション) 設定・動作確認.....	9
5. 付録.....	11
5.1 MDK メソッドリスト	11
5.2 MDK で複数のマーチャント ID を使い分ける方法.....	11
6. 改定履歴.....	13

1. 導入の前に

1.1 本ガイドの内容

本ガイドは、株式会社 DG フィナンシャルテクノロジーが提供する VeriTrans4G を利用するための専用ソフトウェア MDK (Merchant Development Kit) をインターネット店舗などに導入する開発者向けのガイドです。MDK ファイル構成や設定方法、サンプルプログラム等について記載しています。

1.2 著作権、および問い合わせ先

[著作権]

本ドキュメントの著作権は株式会社 DG フィナンシャルテクノロジーが保有しています。

Copyright © 2025 DG Financial Technology, Inc., a Digital Garage company. All rights reserved.

[お問い合わせ先]

株式会社 DG フィナンシャルテクノロジー ベリトランス テクニカルサポート

電子メール: tech-support@veritrans.jp

2. 導入準備

2.1 基本導入手順

VeriTrans4G PHP 版 MDK の導入にあたり、以下の作業が必要となります。

1. 実行環境の確認と準備
2. PHP 用サポート・ライブラリの組込(有効化)
3. MDK 設定
4. MDK パッケージの組込

2.2 実行環境の確認と準備

MDK のご利用には、以下の環境が必要です。

表 1 PHP 実行環境要件

パッケージ	バージョン要件	説明
PHP	7.4 以上または 8.0、8.1、8.2、 8.3 以上	PHP 実行環境
Composer	2.2.7 以上	依存関係管理ツール

※本サンプルの Web アプリケーションを実行するためには PHP 8.2 以上、Laravel 11.9 以上が必要です。

有効になっている必要のある拡張モジュールは以下の通りです。

表 2 PHP 拡張モジュール要件

拡張モジュール	説明
マルチバイト文字列 (mbstring)	マルチバイト対応の文字列関数を提供するモジュール
JSON	JSON デコード、エンコード関数を提供するモジュール
OpenSSL	ソケットについての暗号化を有効にするためのモジュール

導入されている環境のバージョンおよび有効となっている拡張モジュールを `php -i` コマンドの出力結果、あるいは `phpinfo()` 関数から該当の箇所を確認してください。

拡張モジュールが未導入の場合は事前に導入してください。

また、本 MDK は `default_charset` および `internal_encoding` が UTF-8 となっている環境を前提としています。

PHP 5.6 以降では `default_charset` には既定値として UTF-8 が設定され、`internal_encoding` も `default_charset` の設定を利用するようになっていますので、変更せずそのままご利用ください。

2.2.1 注意事項

- (1) MDK が依存するライブラリだけではなく、ファイルやディレクトリに設定された権限、ネットワークトラブル等によって、MDK が正しく動作しない可能性もあります。弊社より提供しているサンプルプログラムでは、MDK 呼び出し部のエラーハンドリングを省略していますが、MDK 組込みの際には、MDK が実行時エラー (Exception)を返すことも想定して下さい。また、MDK 呼び出し部に関しても十分なテストを実施してください。
- (2) 本バージョンの MDK はロギングライブラリを同梱していないため、PSR-3 LoggerInterface を実装した Logger インスタンスを呼び出し元から TGMDK_Logger クラスにセットする必要があります。MDK のログは何かしらの問題が発生した際、調査のために必要なログとなりますので、サンプルコード等を参考に、正しくログが保存されるよう呼び出し元プログラムの実装を行ってください。

注意:

ログ出力の設定については、ログファイルの書き込み権限、ログ出力先ディレクトリパスの実行権限、ディレクトリへの書き込み権限等、ファイルの権限に関する設定に誤りがないようご注意ください。

ログファイルが出力されている場合でも、権限不足によりログローテーションに失敗する場合があります。

ログファイルのローテーションに関する設定についてもご確認下さい。

2.3 PHP 用サポート・ライブラリの組込(有効化)

PHP 実行環境として、下表に示すサポート要件を満たす必要があります。

各サポートの有効化方法については、環境によって異なりますので、店舗様にてご確認頂きますようお願いいたします。

【例】

・ソース・コンパイルの場合

```
./configure (中略) --with-openssl --enable-sockets --enable-hash --enable-mbstring
```

表 3 PHP サポート要件

設定事項	設定(PHP configure) オプション
OpenSSL サポートの有効化	--with-openssl
Socket サポートの有効化	--enable-sockets
hash サポートの有効化	--enable-hash
mbstring サポートの有効化	--enable-mbstring

有効化作業後、php -i コマンドラインインタフェースでの出力結果、または phpinfo()関数の出力結果にてサポートが有効(enabled)となっているか確認して下さい。

3. 設定方法

3.1 MDK の確認

ダウンロードした Mdk4G-php8-x.x.x.tar.gz を解凍します。(x.x.x は、MDK のバージョンを示します。)

表 2 PHP 版 4G MDK ファイル一覧

ディレクトリ/ファイル名		説明	
veritrans-tgmdk	README.txt	MDK の導入前にお読みください。	
	composer.json	プロジェクトの依存情報が記述されています。	
	composer.lock	依存パッケージのバージョンリストが記述されています。	
	src/	tgMdk/	MDK 本体が格納されています。
		dto	各 API の要求、応答データクラスが格納されています。
		3GPSMDK.properties	MDK 設定ファイル
resources/	cert.pem	CA 証明書ストアファイル ※必要に応じて MDK の設定ファイルに設定したパスに配置します。	

3.2 MDK の設定

【基本設定ファイル】

4G MDK 設定ファイルは、**3GPSMDK.properties** です。設定ファイルには重要な情報が記載されているため、第三者に見えないようにして下さい。

(1) サービスに関連する設定

- ダミーモードの指定

テスト用にダミー決済を発生させる場合に指定します。

```
DUMMY_REQUEST = x
```

「0」: ダミーモード OFF

「1」: ダミーモード ON

- MDK 固有エラーモード

決済サーバーに無関係な、MDK 固有のエラーテスト用に指定します。

`MDK_ERROR_MODE = x`

「0」:MDK エラーモード OFF

「1」:MDK エラーモード ON

※エラーモード ON の場合には vResultCode に "MA99" が返戻されます。

(2)接続に関連する設定

- 接続先 URL の設定

接続先サーバ(決済サーバー)の URL を設定します。

※デフォルトでの設定のままご利用下さい。

`HOST_URL = https://xxx.xxx.xxx.xxx:xxx`

- 接続タイムアウト値(秒)の設定

決済サーバーへの接続が確立できない場合に、接続試行を中断するまでの接続試行開始からの秒数を指定します。

`CONNECTION_TIMEOUT = XXX`

- 読み取りタイムアウト値(秒)の設定

決済サーバーからのレスポンスがない場合に、接続を切断するまでのリクエスト開始からの秒数を指定します。

`READ_TIMEOUT = XXX`

※READ_TIMEOUT が未設定の場合は、読み取りタイムアウト値(秒)に接続タイムアウト値(秒)が適用されます。

- SSL 暗号用 CA(Certificate Authority)証明書ファイル名の設定(オプション設定)

同梱の CA 証明書ファイル(PEM 形式)を配置するパスを指定します。

通常は設定せずにご利用ください(デフォルトでは未設定)。決済サーバーとの SSL 暗号化通信でエラーが発生する場合、OS や PHP を安全なバージョンに更新したり、本設定で CA 証明書ファイルのパスを明示的に指定することで解決する場合があります。

`CA_CERT_FILE =`

※ver1.3.0 以前の MDK をご利用の場合は設定が必須になります。

- プロキシサーバ URL の設定

プロキシサーバを利用する場合、サーバホスト名/IP アドレス、ポート番号を設定して下さい。

`PROXY_SERVER_URL = http://xxxx.xxx.xxx.xxx:xxxx`

- プロキシサーバ接続認証用ユーザ ID

プロキシサーバを利用し、かつ接続に認証が必要な場合、プロキシサーバ接続用ユーザ名を設定下さい。

`PROXY_USERNAME = xxxxxxxx`

- プロキシサーバ接続認証用パスワード

プロキシサーバを利用し、かつ接続に認証が必要な場合、プロキシサーバ接続用パスワードを設定下さい。

```
PROXY_PASSWORD = xxxxxxxx
```

- マーチャント CCID の設定

店舗サイト用 CCID 文字列を指定します。

```
MERCHANT_CC_ID = xxxxxxxxxxxxxx
```

- マーチャント認証鍵の設定

店舗サイト用認証鍵文字列を指定します。

マーチャント CCID およびマーチャント認証鍵については MAP(マーチャント管理ポータル)の「API 設定情報」より確認が可能です。

```
MERCHANT_SECRET_KEY = xxxxxxxxxxxxxxxxxxxx
```

3.3 アプリケーションへの MDK の組み込み

本 MDK は Composer パッケージとなっています。

```
呼び出し元アプリケーションの composer.json に以下のように依存関係を設定してください。“repositories”:[  
  {  
    “type”: “path”,  
    “url”: “../local_packages/veritrans-tgmdk”,  
    “options”:{  
      “symlink”: false  
    }  
  }  
],
```

この例では、呼び出し元アプリケーションの配置パスと同階層(composer.json の 1 つ上の階層)に local_packages というディレクトリを作成し、その中に veritrans-tgmdk を配置した状態を想定しています。

4. サンプルプログラムの利用

4.1 サンプルプログラム(コマンドラインプログラム)設定・動作確認

(1) Mdk4G-Sample-php8-x.x.x.tar.gz を解凍

ダウンロードした Mdk4G-Sample-php8-x.x.x.tar.gz を解凍します。(x.x.x は、サンプルプログラムのバージョンを示します。)

※本サンプルの Web アプリケーションを実行するためには PHP 8.2 以上、Laravel 11.9 以上が必要です。

コマンドラインから動作確認が可能なサンプルプログラムは以下のディレクトリにあります。

```
veritrans-mdk-sample/command
```

(2) 4G MDK 本体のファイル群の配置

veritrans-mdk-sample/local_packages ディレクトリ直下に MDK の veritrans-tgmdk ディレクトリを配置します。

(3) MDK の設定

「3.2 MDK の設定」に従い、配置した veritrans-tgmdk ディレクトリ内の src/tgMdk/3GPSMDK.properties をエディタで開き、必要な項目の設定をします。

(4) MDK の組み込み

command ディレクトリに移動し、composer コマンドで依存関係をインストールしてください。

```
composer install
```

(5) 動作確認

app/command/[サービス名]/ ディレクトリにあるサンプルプログラムを php コマンドで実行します。

以下の例では、クレジットカード決済(与信)のサンプルプログラムを実行しています。

```
$ cd app/command/card/
$ php -f CLI_CardAuthorize.php
success
Transaction Successfully Complete
[Result Code]: A001H00100000000
[Order ID]: dummy1444907080
$
```

- ✓ エラーが発生する場合は、MDK が適切なディレクトリにコピーされていること、また、3GPSMDK.properties の設定内容が適切であることをご確認ください。
- 実行時のコンソールには、決済サーバーからのレスポンス値が出力されます。通信内容の詳細についてはログファイルをご確認ください。ログファイルは app/LoggerDefine.php にて、app/logs ディレクトリに

出力するよう設定されています。

4.2 サンプルプログラム (Web アプリケーション) 設定・動作確認

(1) Mdk4G-Sample-php8-x.x.x.tar.gz を解凍

ダウンロードした Mdk4G-Sample-php8-x.x.x.tar.gz を解凍します。

Web アプリケーションとして実装されているサンプルプログラムは laravel_sample ディレクトリにあります。

(2) 4G MDK 本体のファイル群の配置

veritrans-mdk-sample/local_packages ディレクトリ直下に MDK の veritrans-tgmdk ディレクトリを配置します。

(3) MDK の設定

「3.2 MDK の設定」に従い、MDK の設定をします。

(4) PHP の設定

Laravel のセッション設定はデータベースセッションドライバがデフォルトになっています。

また、本サンプルアプリケーションでは config/database.php をデフォルトのまま利用していますので、拡張モジュールとして pdo_sqlite と sqlite3 が有効になるよう php.ini で設定してください。

```
extension=pdo_sqlite
extension=sqlite3
```

(5) 「sample_setting.php」ファイルの設定

laravel_sample/config ディレクトリの中にある sample_setting.php ファイルを、環境に合わせて編集します。

■「MPI ホスティング」を利用する場合

「MPI ホスティング」のサービス用サンプルを利用する場合は、以下を設定して下さい。

•redirect_url : 決済サーバーから消費者ブラウザ経由で店舗ヘリダイレクトする際の URL を設定

```
'mpi' => [
    'redirect_url' => 'http://127.0.0.1:8000/mpi/result'
],
```

■「MDK トークン」を利用する場合

クレジットカードを利用した決済のサービス用サンプルを利用する場合は、以下を設定して下さい。

•token_api_key : 店舗サイト毎に発行されたトークン取得用のキーを設定

```
'token' => [
    'token_api_key' => 'aaaaaaaa-0000-1111-bbbb-abcdef012345'
]
```

(6) 動作確認

laravel_sample ディレクトリに移動し、composer で必要となるパッケージをインストールします。

```
composer install
```

その後、Laravel artisan コマンドでアプリケーションを起動します。

```
php artisan serve
```

上記のコマンド例では、ポート 8000 への http 要求を待ち受ける状態になりますので、ブラウザで <http://127.0.0.1:8000/menu> を開いてください。

5. 付録

5.1 MDK メソッドリスト

MDK のクラス、メソッドの一覧を phpdoc 形式で出力して公開しています。

ダウンロードサイトより「MDK_MethodList_PHP」を入手し、「index.html」よりご参照ください。

5.2 MDK で複数のマーチャント ID を使い分ける方法

(1) 決済要求毎にマーチャント ID を切り替える

MDK は通常、設定ファイル(3GPSMDK.properties)に記述されたマーチャント CCID と SECRET_KEY(以下、認証情報)を利用して署名を算出し、決済要求時に付与して送信します。署名の算出は、処理要求単位(トランザクション単位)で行います。

マーチャント ID の切り替えを行うためには、決済要求の実行前に、MDK が保持しているメモリ上の認証情報を上書きする必要があります。

※MDK は、初回の呼び出し時に設定ファイルから読み込んだ認証情報を、スタティックな情報として保持しています。そのため、設定ファイル内の認証情報を書き換えても、アプリケーションの再起動を行うまで認証情報は反映されません。

認証情報を上書き設定する手順を以下に示します。

1. 利用する認証情報を取得する
2. メモリ上の認証情報を、取得した情報で上書き設定する(*1)
3. トランザクションオブジェクトを生成する
4. トランザクションを実行する

以下の点に注意して実装してください。

- ✓ MDK の仕様上、設定ファイル(3GPSMDK.properties)内のマーチャント CCID と SECRET_KEY には、適当な値(半角英数字)を設定してください。未設定の場合、MDK の初期化時にエラーとなります。
- ✓ 認証情報の上書き設定は、必ずトランザクションオブジェクトを生成する「前に」行ってください。
- ✓ 前回利用したマーチャント情報の設定が変数に残っている等で、意図しないマーチャント ID で決済が行われてしまうような事故を防止するため、トランザクションを実行する都度、毎回必ず認証情報の設定を行ってください。
- ✓ 本機能をご利用の際は、マーチャント ID が正しく切り替わっているかどうか、十分にテストを実施してください。

(*1)

決済要求の都度、使用する情報を設定する形で実装してください。例えば、設定ファイルには認証情報 A が記述されているので、認証情報 B を使いたいときだけ上書きを行う、という実装では正しく切替ができない可能性があります。

【サンプルコード】

ここでは、認証情報がプロパティファイル(merchant_info.ini)に保存されているものとします。

```
merchant_info.ini
MERCHANT_CC_ID_1           = A10000000000000000001CC
MERCHANT_SECRET_KEY_1     = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

MERCHANT_CC_ID_2           = A10000000000000000002CC
MERCHANT_SECRET_KEY_2     = YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY

MERCHANT_CC_ID_3           = A10000000000000000003CC
MERCHANT_SECRET_KEY_3     = ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

認証情報の上書き処理の実装例を以下に示します。

```
// ファイルへのパスは適宜調整してください
$merchant_infos = parse_ini_file("merchant_info.ini", false);

// サンプルとして、ユニーク値を 1 としています。
$merchantUniqueValue = "1";
$props["merchant_ccid"] = $merchant_infos["MERCHANT_CC_ID_".$merchantUniqueValue];
$props["merchant_secret_key"] = $merchant_infos["MERCHANT_SECRET_KEY_".$merchantUniqueValue];

$transaction = new TGMDK_Transaction();
// TGMDK_Transaction クラスの execute の第 2 引数に連想配列を渡す
$response_data = $transaction->execute($request_data, $props);
```

(2) 結果通知における content-hmac の検証

VeriTrans4G より送信される結果通知の HTTP ヘッダーには、改ざんチェック用の HMAC 値が content-hmac というヘッダー名で設定されています。この HMAC 値は、SECRET_KEY を利用してリクエストボディを SHA256 ハッシュ化した値です。

この content-hmac のチェックを行うことで、リクエストが改竄されていないかどうかをチェックすることが可能ですが、算出に SECRET_KEY の値を利用するため、content-hmac のチェック時には注意が必要となります。

content-hmac のチェックには、TGMDK_MerchantUtility クラスの checkMessage 関数を利用します。

```
TGMDK_MerchantUtility::checkMessageBySecretKey($secretKey, $msgBody, $sContentHmac)

$secretKey      : マーチャント認証鍵(SECRET_KEY)を指定します。
$msgBody        : リクエストボディを指定します。
$sContentHmac   : content-hmac ヘッダの値を指定します。
```

※マーチャント ID の切替が不要な場合は、引数が2つの関数 checkMessage(\$body, \$hmac) を利用します。

6. 改定履歴

2021/09 :Ver1.0.0 リリース

※ PHP 7.4/8.0, Composer 2.0.8 以上に対応

2021/09 :Ver1.0.1 リリース

- ・ ログ出力に関する注意事項の記載箇所の変更
- ・ MDK 設定ファイルの DTO_ENCODE パラメータの廃止に伴う記載削除
- ・ 2.2 に文字コードに関する注意事項を追記し、「3.2 文字コード調整」を削除
(以降の章番号は繰り上げ)

2023/01 :Ver1.0.2 リリース

- ・MDK 設定ファイルの CA_CERT_FILE パラメータの説明を修正
- ・「5.2 MDK で複数のマーチャント ID を使い分ける方法」を追加
- ・ダウンロードサイトの URL の記載を削除(本書への掲載は中止)

2025/10 :Ver1.0.3 リリース

- ・要求 PHP のバージョンを Laravel に合わせて修正
- ・MDK 設定ファイルの CA_CERT_FILE パラメータの説明を修正
- ・サンプルプログラム(Web アプリケーション)実行時の URL を修正
- ・「3.2 MDK の設定」の SSL 暗号用 CA 証明書ファイル名の設定をオプション設定に変更