4G MDK　インストールガイド

VeriTrans 4G

MDK for Ruby Installation Guide

ver.1.0.4 （2022 年 12 月～）

目次

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

［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 MDK（Ruby）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
2. Ruby 用サポート・ライブラリの組込（有効化）
3. MDK モジュールの組込
4. MDK 設定

2.2 実行環境の確認と準備

VeriTrans4G MDK（Ruby）の最新版およびサンプルプログラムを、以下の URL よりダウンロードしてください。

https://www.veritrans.co.jp/trial/login/

MDK のご利用には、以下の環境（TLS1.2 以上で通信可能な環境）が必要です。

表 1 コンポーネント(パッケージ)要件

パッケージ・コンポーネント	バージョン要件	説明
Ruby	2.7.0 以上	Ruby 実行環境
OpenSSL	1.0.1i 以上*1	SSL 暗号通信用ライブラリモジュール
log4r	1.1.8 以上	ログ出力ライブラリモジュール

導入されている Ruby 環境環境のバージョン及びパッケージサポート状態を、ruby –v コマンドの出力結果から該当の箇所を確認して下さい。

サポートパッケージが未導入の場合は事前に導入して下さい。

*1 TLS1.2 以上の通信をサポートする OpenSSL のバージョンは 1.0.1 以降となりますが、OpenSSL はいくつかの重大な脆弱性が発表されておりますので、本書ではバージョン 1.0.1i 以上のご利用を推奨しております。ただし、OpenSSL の脆弱性は、1.0.1i 以上のバージョンでも報告されていますので、システムをアップグレードする際には、できるだけ最新バージョンをお使い頂きますようお願い申し上げます。

2.3 Ruby 用サポート・ライブラリの組込（有効化）

Ruby 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します。

表 2 Ruby サポート要件

設定事項	設定コマンド
OpenSSL サポートの有効化	yum install openssl-devel
log4r サポートの有効化	gem install log4r

3. 設定方法

3.1 MDK の確認

ダウンロードした Mdk4G-ruby-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 3 Ruby 版 4G MDK ファイル一覧

ディレクトリ／ファイル名			説明
MdkRuby/	README.txt		MDK の導入前にお読みください
	tgMdk/	lib/	MDK 本体が格納されています。
		tg_mdk.ini	MDK 設定ファイル
		tg_mdk_log4r.xml	ログ出力用設定ファイル ※ご利用の Ruby バージョンが 2.5.0 以降の場合、こちらをご利用下さい。
		tg_mdk_log4r.yaml	ログ出力用設定ファイル
		tg_mdk_log4r.yaml4log4r119	ログ出力用設定ファイル ※ご利用の log4r バージョンが 1.1.9 以前の場合、こちらをご利用下さい。
	resources/	cert.pem	CA 証明書ストアファイル ※MDK の設定ファイルに設定したパスに配置します。

3.2 アプリケーションへの MDK の組み込み

tgMdk/ディレクトリを、店舗様のアプリケーションが参照可能な任意のディレクトリにコピーして下さい。

ファイルの実行権限については、アプリケーションに合わせて下さい。

以下に、設定ファイルの設定方法をご説明します。

3.3 MDK の設定

【基本設定ファイル】

4G MDK 設定ファイルは、**tg_mdk.ini** です。設定ファイルには重要な情報が記載されておりますので、第三者に見えないようにしてください。

(1)サービスに関連する設定

(A)ダミーモード

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

(B)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

(A)接続先ホスト

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

(B)接続タイムアウト値(秒)

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

(C)読み取りタイムアウト値(秒)

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。

(D)SSL 暗号用 CA(Certificate Authority)証明書ファイル名

同梱の CA 証明書ファイル(PEM 形式)を配置する絶対パスを指定します。

CA_CERT_FILE = xxxxxxxxxxxxx

(E)SSL プロトコルの設定

ベリトランスと SSL 暗号通信を行う際の SSL プロトコルを指定します。

※デフォルトでの設定のままご利用下さい。

SSL_PROTOCOL = xxxxxxx

(3)プロキシサーバの設定

(A)プロキシサーバ URL

プロキシサーバをご利用の場合、この項目にサーバホスト名／IP アドレス、ポート番号を設定下さい。

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

(B)プロキシサーバ接続認証用ユーザ ID

プロキシサーバをご利用で且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 = xxxxxxx
```

(C)プロキシサーバ接続認証用パスワード

プロキシサーバをご利用で且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 = xxxxxxx
```

(4)マーチャント認証用の設定

(A)マーチャント CCID の設定

ベリトランスより通知の店舗サイト用 CCID 文字列を指定します。

```
MERCHANT_CC_ID = xxxxxxxxxxxx
```

(B)マーチャント認証鍵の設定

ベリトランスより通知の、店舗サイト用認証鍵文字列を指定します。

```
MERCHANT_SECRET_KEY = xxxxxxxxxxxxxxx
```

(5)環境に関連する設定

(A)文字列のエンコードに関連する設定

要求電文に用いている文字列のエンコードを指定します。

```
DTO_ENCODE = xxxxxxxxxxxxxxx
```

リクエスト DTO にセットする全角文字列パラメータは、必ず UTF-8 である必要がありますが、DTO_ENCODE を指定すると、指定した UTF-8 以外のエンコーディングから UTF-8 への変換を MDK 内部で自動的に行います。

注1） 本項目を指定しない場合はマーチャントプログラム内で明示的に UTF-8 に変換する必要があります。
注2） 本項目を指定した場合はマーチャントプログラム内でエンコーディング処理を行わないでください。

ここで指定できる日本語エンコーディング名は以下の通りです。

・UTF-8（UTF-8 を指定した場合は変換無し）

・Shift_JIS（Windows 系）、SJIS（UNIX 系）

・EUC-JP

【ログ設定ファイル】

ログ設定ファイルは **tg_mdk_log4r.yaml**、もしくは **tg_mdk_log4r.xml** です。

両方の設定ファイルが存在する場合は YAML 形式のログ設定ファイルが優先して参照されます。

Ruby2.5.0 以上をご利用の場合については、log4r にて YAML 形式のファイルを扱うことができないため、XML 形式のログ設定ファイル（tg_mdk_log4r.xml）をご利用ください。

※tg_mdk_log4r.yaml を削除することで tg_mdk_log4r.xml が参照されるようにしてください。

（以降の記述についても、xml をご利用の場合は適宜読み替えて下さい。）

以下に yaml 形式を例に、設定方法を記載します。

（1）ログ出力レベルの指定

MDK のログ出力レベルを指定します。

指定可能値：「OFF／FATAL／ERROR／WARN／INFO／DEBUG（※）」

※より多くの情報を出力したい場合は、DEBUG を設定して下さい。

```
loggers:
～ 省略 ～
  - name: "Veritrans"
    type: "Log4r::Logger"
    level: "DEBUG"
    trace: "true"
    outputters:
      - "E1"
      - "F1"
```

（2）ログ出力ファイルの指定

ログ出力ファイルのパスを指定します。

```
outputters:
～ 省略 ～
  - name: "F1"
    type: "Log4r::FileOutputter"
```

```
filename: "/logfile_dir_change_here/mdk_ruby.log"
trunc: "false"
formatter:
  name: "P2"
  type: "Log4r::PatternFormatter"
  pattern: "%d [%l] %C(%c) - %M"
  pattern2: "%d [%l] %C(%t) - %M"
  date_pattern: "%Y/%m/%d %H:%M:%S"
```

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-ruby-x.x.x.tar.gz を解凍

ダウンロードした Mdk4G-Sample-ruby-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。）

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 MdkSample-Ruby/command 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

MdkSample-Ruby/command/lib ディレクトリ直下に、MDK の MdkRuby ディレクトリを配置します。

（3）MDK の設定

「3.3 MDK の設定」に従い、MDK の設定をします。

（4）動作確認

MdkSample-Ruby/command/lib/command/[サービス名]/ 直下にあるサンプルプログラムを実行します。

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 MdkSample-Ruby/command/lib/
$ ruby command/card/cli_card_authorize.rb
...Transaction Successfully Complete
[Status]: success
[Result Code]: A001H00100000000
[Order ID]: dummy1444983947
$
```

- ✓ 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、tg_mdk.ini と tg_mdk_log4r.yaml の設定内容が適切であることをご確認下さい。
- ✓ 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、tg_mdk_log4r.yaml に指定したログファイルをご確認ください。

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

動作確認を行うためには、rails アプリケーションが動作する WEB サーバが必要です。

（1）Mdk4G-Sample-ruby-x.x.x.tar.gz を解凍

ダウンロードした Mdk4G-Sample-ruby-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 MdkSample-Ruby/web ディレクトリにあります。

(2)web ディレクトリのコピー

WEB サーバの適切なディレクトリに web ディレクトリをコピーします。

(3)4G MDK 本体のファイル群の配置

web/vendor ディレクトリ直下に、MDK の MdkRuby ディレクトリを配置します。

(4)MDK の設定

「3.3 MDK の設定」に従い、MDK の設定をします。

(5)実行権限の設定

環境及び実行ユーザに合わせ、Web サーバ上で実行可能な権限をファイルに設定します。

(6)「env4sample.yml」ファイルの設定

web/config ディレクトリ直下にある env4sample.yml ファイルを、環境に合わせて編集します。

1. [Common]セクションの base.url プロパティ値の値を、サイトのベース URL(サンプルのルート・ディレクトリまで)に変更します。

```
例:)
(標準設定)base.url=http://127.0.0.1
(変更後) base.url=http://testsite.yourdomain.com
```

2. 「PayPal 決済」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
PayPal:
  return_url: http://127.0.0.1/web/paypal/authorize/comfirm
  cancel_url: http://127.0.0.1/top
```

3. 「MPI ホスティング(3D-Secure)」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
MPI:
  redirection_url: http://127.0.0.1/web/mpi/authorize/search
  redirection_url2: http://127.0.0.1/web/mpi/re_authorize/search
```

4. 「永久不滅ポイント決済」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
SAISON:
```

```
  merchant_redirection_uri_pc: http://127.0.0.1/web/saison/capture
  merchant_redirection_uri_mb: http://127.0.0.1/web/saison/mb/capture
  pc_or_sp_settlement_method: PC
```

5. 「キャリア決済」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
Carrier:
  success_url: http://127.0.0.1/web/carrier/result?result=SUCCESS
  cancel_url: http://127.0.0.1/web/carrier/result?result=CANCEL
  error_url: http://127.0.0.1/web/carrier/result?result=ERROR
  success_url.mb: http://127.0.0.1/web/carrier/mb/result?result=SUCCESS
  cancel_url.mb: http://127.0.0.1/web/carrier/mb/result?result=CANCEL
  error_url.mb: http://127.0.0.1/web/carrier/mb/result?result=ERROR
  push_url: http://127.0.0.1/web/push/push_receipt/carrier
  push_url.mb: http://127.0.0.1/web/push/push_receipt/carrier
```

6. 「ショッピングクレジット決済」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
ORICOSC:
  merchant_redirection_url_pc: http://127.0.0.1/web/oricosc/authorize/complete
  merchant_redirection_url_mb: http://127.0.0.1/web/oricosc/mb/authorize/complete
```

7. 「銀聯ネット決済（UPOP）決済」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
UPOP:
  term_url: http://127.0.0.1/web/upop/authorize/authorize_result
```

8. 「Alipay 決済」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
Alipay:
  success_url: http://127.0.0.1/web/alipay/authorize/authorize_result
  cancel_url: http://127.0.0.1/web/alipay/authorize/authorize_result
```

9. 「ワンクリック継続課金サービス」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
PAYNOWID-MPI:
  redirection_url: http://127.0.0.1/web/paynowid/mpi/authorize/search
```

10. 「楽天 ID 決済」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
Rakuten:
  success_url: http://127.0.0.1/web/rakuten/result?result=SUCCESS
```

```
    error_url: http://127.0.0.1/web/rakuten/result?result=ERROR
    push_url: http://127.0.0.1/web/push/push_receipt/rakuten
```

11. 「リクルートかんたん支払い」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
Recruit:
    success_url: http://127.0.0.1/web/recruit/result?result=SUCCESS
    error_url: http://127.0.0.1/web/recruit/result?result=ERROR
    push_url: http://127.0.0.1/web/push/push_receipt/recruit
```

12. 「LINE Pay」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
Linepay:
    success_url: http://127.0.0.1/web/linepay/result?result=SUCCESS
    cancel_url: http://127.0.0.1/web/linepay/result?result=CANCEL
    error_url: http://127.0.0.1/web/linepay/result?result=ERROR
    push_url: http://127.0.0.1/web/push/push_receipt/linepay
```

13. 「MasterPass」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
Masterpass:
    success_url: http://127.0.0.1/web/masterpass/authorize?result=SUCCESS
    cancel_url: http://127.0.0.1/web/masterpass/result?result=CANCEL
    error_url: http://127.0.0.1/web/masterpass/result?result=ERROR
```

14. 「PayPay」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
PayPay:
    success_url: http://127.0.0.1/web/paypay/result?result=SUCCESS
    cancel_url: http://127.0.0.1/web/paypay/result?result=CANCEL
    error_url: http://127.0.0.1/web/paypay/result?result=ERROR
    push_url: http://127.0.0.1/web/push/push_receipt/paypay
```

15. 「AmazonPay」用サンプルを使用される場合は、下記設定の部分を確認してください。

```
Amazonpay:
    success_url: http://127.0.0.1/web/amazonpay/result?result=SUCCESS
    cancel_url: http://127.0.0.1/top
    error_url: http://127.0.0.1/web/amazonpay/result?result=ERROR
```

16. 「銀行決済」用サンプルを使用される場合は、下記設定の部分を確認してください。

Bank:

term_url: http://127.0.0.1/web/bank/thanks/index

17. クレジットカードを利用した決済用サンプルを使用される場合は、下記設定の部分を確認してください。

Token:

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

token_api_url: https://xxx.xxx.xxx.xxx/4gtoken　※デフォルトの設定のままご利用下さい。

18. 消費者向け取引画面サンプルトップページ、管理用取引メニューサンプルトップページを確認します。

[消費者向け取引画面トップ]

http:// (導入サーバベース URL)/top

[管理用取引メニュー画面トップ]

http:// (導入サーバベース URL)/top/admin_menu

4.3 サンプルプログラム（結果通知受信用プログラム）設定・動作確認

サンプルプログラムでは、決済サーバーから送信される各種結果通知を、店舗側アプリケーションで受信する処理の動作を確認できます。動作確認を行うためには rails アプリケーションが動作する WEB サーバが必要です。また、WEB サーバ上で動作するプログラムのため、「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」に従い、配置を行って下さい。

以下、WEB サーバへの配置後の手順から説明します。

（1）受信データ保存ディレクトリの設定

ソースコードは web/app/controllers/web/push/push_receipt_controller.rb です。

ソースコード内の、受信データの保存場所項目を設定します。

例：

＜デフォルト設定＞

```
# プッシュデータ保存ディレクトリ
def initialize
  @save_path = "#{rails_root}/tmp/push/"
end
```

＜店舗様の環境に合わせて変更＞

```
# プッシュデータ保存ディレクトリ
def initialize
  @save_path = "/home/my_push/"
end
```

（2）通知受信 URL の設定

店舗側の通知受信 URL は、MAP（マーチャント管理ポータル）の「各種設定変更」より設定が可能です。詳しくは MAP のご利用ガイドをご参照ください。

決済サービスによっては、ダミー決済時に限り、決済申込時（与信時）に通知受信 URL（通知 URL）を設定することが可能です。詳しくは、VeriTrans4G 開発ガイド（サービス毎のインターフェース詳細）をご参照ください。

例： 店舗様 Web サーバの /push/ ディレクトリにマッピングした例です。

MPI プッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/mpi

銀行プッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/bank

コンビニプッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/cvs

電子マネープッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/em

PayPal プッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/paypal

キャリアプッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/carrier

ショッピングクレジットプッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/orocosc

楽天 ID プッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/rakuten

リクルートかんたん支払いプッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/recruit

LINE Pay プッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/linepay

銀聯ネット決済（UPOP）プッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/upop

Alipay プッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/alipay

PayPay プッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/paypay

AmazonPay プッシュ URL: http://xxx.xxx.xxx.xxx/web/push/push_receipt/amazonpay

結果通知データを受信すると、サンプルプログラムのソースコードに設定した場所に CSV 形式のファイルが作成されます。

（結果通知項目の詳細は、VeriTrans4G 開発ガイドを参照して下さい）

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 RDoc 形式で出力し、ダウンロードサイトに公開しています。

以下の URL よりダウンロードしてください。

https://www.veritrans.co.jp/trial/login/

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(tg_mdk.ini)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(tg_mdk.ini)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。
- ✓ MDK の設定はメモリ上に保持されますので、毎回認証情報を設定してください。スレッドが再利用されるケースでは、前回上書きした値が保持されますので、意図しないマーチャント ID で決済が行われる可能性があります。
- ✓ 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)

基本的に、スレッドスタティックな形で保持している情報が上書きされますが、切り換える場合は毎回、使用する情報を設定する形で実装する必要があります。※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ことではありません。

【サンプルコード】

```
@request_data = Veritrans::Tercerog::Mdk::CardAuthorizeRequestDto.new
・・・

#=========================================================================
#  4G 認証情報を設定
#=========================================================================
tmp_config = MdkConfig.instance
tmp_config.overwrite("MERCHANT_CC_ID", "A10000000000000000001CC")
tmp_config.overwrite("MERCHANT_SECRET_KEY", "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX")

#=========================================================================
#  実施
#=========================================================================
tran = Veritrans::Tercerog::Mdk::MdkTransaction.new
@response_data = tran.execute(@request_data)
```

補足：Ruby 版 MDK における認証情報のスコープ

MdkConfig クラスは Singleton として実装されております。システム構成によっては不都合が生じる可能性がありますので、ご注意ください。

（2）結果通知における content-hmac の検証

VeriTrans4G より通知される結果通知の POST リクエストには、content-hmac というヘッダが設定されています。この値はリクエストボディから算出されます。

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、MdkMerchantUtility クラスの check_message_by_secret_key?メソッドを利用します。デフォルトのサンプルでは、content-hmac（検証用に利用する署名）のチェックに MdkMerchantUtility クラスの check_message?メソッドで実装していますが、複数のマーチャント ID を使い分ける場合には、MdkMerchantUtility クラスの check_message_by_secret_key?メソッドをご利用ください。

```
# 第１引数：マーチャント認証鍵（マーチャントパスワード）
# 第２引数：電文の本文
# 第３引数：content-hmac
# 戻り値：true＝改ざんされていない、 false：改ざんされている
Veritrans::Tercerog::Mdk::MdkMerchantUtility::check_message_by_secret_key?(merchant_secret_key,
request.body.read, @content_hmac)
```

6. 改訂履歴

2017/04 ：Ver1.0.0 リリース

※ 以下、「3G MDK インストールガイド」 Ver 3.0.2 からの更新分を記載します。

「3.1 MDK の確認」の「表 3 Ruby 版 4G MDK ファイル一覧」について、

・tgMdkRuby/を MdkRuby/に修正

「4 サンプルプログラム利用」各項のサンプルプログラムのアーカイブ名、ディレクトリ名を修正

「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」にトークンの設定を追加

2018/02 ：Ver1.0.1 リリース

「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」の MPI ホスティング（3D-Secure）の設定項目から決済種別（payment_mode）を削除

2018/11 ：Ver1.0.2 リリース

「3.1 MDK の確認」の表 3 に tg_mdk_log4r.xml の説明を追加

「3.3 MDK の設定」のログ設定ファイルの説明に tg_mdk_log4r.xml の説明を追加

「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」「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」に記載されていた tgMdkRuby を MdkRuby に修正

2020/07 ：Ver1.0.3 リリース

「2.2 実行環境の確認と準備」を TLS1.1 廃止に伴い修正

「4. サンプルプログラムの利用」に AmazonPay と PayPay に関する情報を追加

2022/12 ：Ver1.0.4 リリース

「2.2 実行環境の確認と準備」の Ruby のバージョン要件を 2.7.0 以上に修正

「5.2 MDK にて、動的にマーチャント ID を切り換える方法」を追加