Electronic Card Transactions on the Internet

CIB Bank Zrt.

eCommerce

Internet-based Card Acceptance Service
Technical Documentation

Version no. 1.47

Documentation for CIB Bank’s electronic commerce solution based on internet technology
The methods and solutions described in this documentation constitute the basis for the eCommerce service provided by CIB. Within the framework of the TLS-based card acceptance system, CIB does not authorise the application of methods other than those described in this document.

CIB Bank Zrt. CIB Bank Ltd. H-1027 Budapest, Medve utca 4–14. H-1995 Budapest Telephone: (06 1) 423 1000 Fax: (06 1) 489 6500Court of Registration: Company Court of the Metropolitan Court of Budapest Company registration no.: Cg. 01-10-041004 Tax number: 10136915-4-44 Group tax number:: 17781028-5-44 Group Community VAT number: HU17781028 Stock exchange membership: Budapest Stock Exchange Ltd. Operating licence number: 957/1997/F, III/41. 044-10/2002. BIC (SWIFT) code: CIBHHUHB
Bank of INTESA SANPAOLO

CONTENTS

Introduction

This document contains the developer's description of the online card-acquiring service operated by CIB Bank Zrt.

CIB Bank Zrt. supports the 'three-party payment model', that is, the card data necessary for payment are only visible to the bank, whereas the customer's personal data are only visible to the merchant. The correlation between the two data sets is ensured by the transaction identifier, which must be unique for each payment.

Frequently used terms

Below are the definitions of the terms used in this documentation:

Merchant a natural or legal person who has a valid eCommerce contract

Webshop an IT service operated by the Merchant (or its agent), in which various products and services can be purchased using a bank card

Customer the person who executes the card payment, or his/her equivalent in IT terms (e.g. web browser)

Authorisation separation of the purchase amount on the customer's card

Debit finalisation of the authorisation, reservation of the separated amount (the crediting on the merchant's account does not take place immediately)

Reversal withdrawal of the authorisation, making the separated amount freely available

Refund reverse booking of the debited transaction

Transaction presence the full payment process, also including the steps that do not require the customer's

3D Secure an optional service that increases payment security by assigning to the transaction a unique code that the customer uses independently of the card (e.g. receives it from the issuing bank in an sms message).

Initialisation a message sent by the webshop to the bank's server regarding the request for payment

Rerouting changing the connection terminals of the customer's browser to the bank's server

Redirection changing the connection terminals of the customer's browser to the webshop

Query identifying the status of the transaction and the payment at any specific time

Closing confirming the authorisation phase of the transaction The bank always reverses any unclosed transactions.

Encryption key The file(s) made available by the bank to the Merchant, which, during the communication between the Merchant and the Bank, ensure confidentiality of the messages sent and received

(3)DES Data Encryption Standard. Algorithm of the encryption used during Merchant-Bank communication

TLS Transport Layer Security. Algorithm of the encryption used during Customer-Bank communication

The payment process

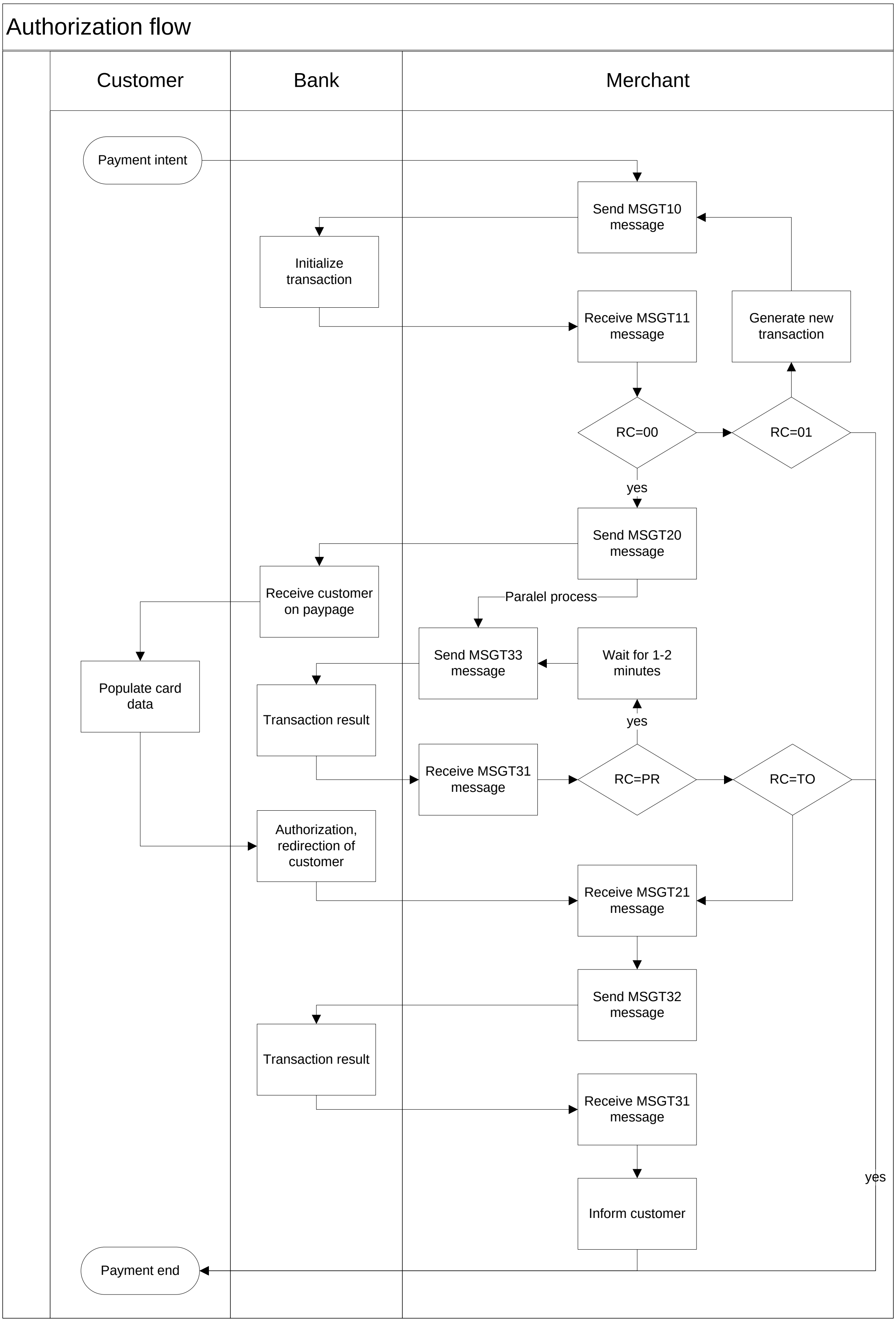
1.1. Authorisation

The process described below only covers the payment part of the full lifecycle (authorisation, in other words blocking or reservation) of the transaction. The transaction withdrawal/retransfer is described below.

1. The customer chooses the necessary goods in the merchant's webshop (hereinafter: "store"), registers and notifies the merchant of his/her intent to pay.
2. The store saves the data necessary for payment, assigs a unique identifier (TRID) to it and then sends it to the bank's server (MSGT10)
3. The bank verifies and registers the request, then sends a confirmation message on the result (MSGT11)
4. In the case of a negative confirmation, the store considers the transaction closed. Optionally, it may provide an opportunity for launching a new transaction.
5. In the case of a positive confirmation, the store directs the customer to the payment page on the bank's server (MSGT20)
6. On the payment page the customer enters the card data necessary for payment
7. Depending on the type of the bank card, the bank may send the customer to the card issuer for further verification (see section '3D Secure'). After verification, the customer returns to the bank
8. Based on the information received from the customer and from the issuing bank, the bank initiates a reservation on the customer's card.
9. After the reservation is completed, the bank returns the customer to the store, that is, to the address provided by the store (MSGT21)
10. The store requests information from the bank's server regarding the successful completion of the reservation (MSGT32)
11. The bank returns the reservation data to the store in a response message (MSGT31)
12. The store informs the customer of the data received from the bank

If the store does not close (MSGT32) the result of the transaction (10) within a pre-defined time (by default 10-15 minutes) following the return of the customer (9), the bank initiates the release of the reservation (reversal) at the issuing bank.
The store may, at any time following a successful initiation, inquire on the transaction status (MSGT33). The structure of the bank's response to this (MSGT31) is the same as the confirmation message, although it is different in that it does not entail the closing of the transaction. If this is supported by the store's application, an inquiry on the transaction status should be sent at regular intervals following the redirection of the customer. If the bank confirms a failure due to time-out, the store may also close it in its own system as an unsuccessful reservation attempt.

The steps in the payment process are shown in the following figure:


Authorization flow
Customer
Bank
Merchant
Payment intent
Send MSGT10 message
Initialize transaction
Receive MSGT11 message
Generate new transaction
RC=00
RC=01
yes
Send MSGT20 message
Receive customer on paypage
Paralel process
Populate card data
Send MSGT33 message
Wait for 1-2 minutes
Transaction result
yes
Receive MSGT31 message
RC=PR
RC=TO
Authorization, redirection of customer
Receive MSGT21 message
Send MSGT32 message
Transaction result
Receive MSGT31 message
yes
Inform customer
Payment end

Messages following authorisation

In the case of a successful authorisation the merchant may apply the following supplementary messages to the transaction:

- inquire on the transaction status at any time (MSGT70). The value of the STATUS field in the bank's response (MSGT71) indicates that it is under reservation, withdrawn, debited or retransferred (see description of the STATUS field in message 71).
- withdraw the reservation (MSGT74). No debiting takes place as a result of this, and the customer receives back the amount reserved on his/her account
- set a partial amount to be refunded from the debited transaction amount (MSGT80)
- refund a debited transaction amount (MSGT78), in which case the default amount or the present partial amount will be refunded

Message structure

The three-party model requires that, during the transaction (purchase process), only two of the three parties be in connection with each other at the same time. In the course of this, the customer typically communicates with the merchant and the bank via a website, while communication between the merchant and the bank takes place via encrypted messages, the contents of which will be detailed at a later point. The messages are sent to the bank, unless they refer to the redirection of the customer, using the HTTP GET or the POST method.

The general message format is as follows:

`https://server.dom/path/to/script.ext?key1=value1&key2=value2...`

where `key<n>` is the parameter name, and `value<n>` is the value assigned to it.

Encryption is 3DES-based in merchant-bank communication and TLS-based in customer-bank communication. The key for the 3DES encryption is provided by the bank, separately for test and live operations. The encryption module (sakicrypt) provided by the bank to the operator, as well as the program (sakide) suitable for testing it are part of the annexes. The steps of the encryption are described in section 5.2.

The bank's responses to the merchant's questions always arrive in the content section of the called page, even if the http response code is other than 200. The only exceptional case when the bank's message arrives in the parameters of the called URL is when the customer is returned from the paypage to the merchant (MSGT21).

1.1. *Field definitions*

Fields used in individual messages

Meaning of the field	ID	Length
Encryption type	CRYPTO	1
Encrypted message	DATA	Max. 255
Message type	MSGT	2
Shop ID	PID	7
Transaction ID	TRID	16
Customer ID	UID	11
Amount	AMO	Max. 16
Currency	CUR	3
Timestamp	TS	14
Response Code	RC	2
Response Text	RT	Max. 255
Authorisation Number	ANUM	Max. 6
Authorisation Type	AUTH	1
Card number	CNUM	Max 19
URL	URL	Max. 255
Language Code	LANG	2
History	HISTORY	Max. 255
Comment	EXTRA01	Max. 50

Detailed explanation of fields:

CRYPTO

Length: 1
Regular term: [13]

Description: The type of the encryption. This only appears in the encrypted message, indicating to the recipient the method used for encrypting the message. In merchant<->bank communication its value is always 1.

Accepted values:

1: 3DES (only in merchant-bank communication)
3: TLS (only in customer-bank communication)

DATA

Length: max. 255
Regular term: [a-zA-Z0-9/%]{0,255}

Description: The encrypted version of the original message. This only appears in the encrypted message, together with the PID and the CRYPTO values.

PID

Length: 7
Regular term: [a-zA-Z]{3}[01][0-9]{3}

Description: The merchant’s identifier. contained by all the messages within the communication with the bank. The first 3 characters identify the store (service provider), and the other 4 figures identify the individual shops within the store. The first of the 4 figures also identifies the default currency of the virtual POS terminal. The following values are currently accepted:

0: HUF
1: EUR

Currently, the terminals only accept the default currencies (see the CUR field).

TRID

Length: 16
Regular term: [0-9]{16}

Description: Unique identifier of the transaction. It identifies the transaction in its whole length (from initialisation to closing), and it is a mandatory component of each unencrypted message. It is generated by the store prior to initialising an intended payment. The use of a pseudo-random number generator during the generation process is recommended. It must not contain only spaces.

UID

Length: 11
Regular term: [a-zA-Z0-9\-_]{11}

Description: The customer's identifier in the merchant's system. If registration is not obligatory in the store, the use of a constant value (e.g. CIB12345678) is also permitted. It must not contain a sequence of dashes.

AMO

Length: max. 16
Regular term: [0-9 \.]{16}

Description: The sum payable. The number of the obligatory decimals depends on the currency of the transaction (0 for HUF, 2 for EUR). The fractional part must be separated from the integer part by a point. In the case of an integer amount there is neither a decimal point, nor a fractional part.
An individual message may also contain multiple amounts, each of which has an AMO prefix.

Examples:

HUF 1,000: 1000
EUR 10: 10.00

CUR

Length: 3
Regular term: [a-zA-Z]{3}

Description: The currency of the transaction in the alphabetical format stipulated by the ISO 3166 standard.

Accepted values:

HUF (Hungarian forint)
EUR (Euro)

TS

Length: 14
Regular term: [0-9]{14}

Description: Time of transaction initialisation on the merchant's page. The time must be entered in the following format: YYYYMMDDHHMISS

Example:
2012 August 31, 23:59:59: 20120831235959

RC

Length: 2
Regular term: [A-Z0-9]{2}

Description: Result of the execution of the current request. The exact interpretation of the possible values depends on the message type, and therefore these are presented in detail under each message type.

Possible values:
RC values: Successful execution
Anything else: An error occurred during execution.

RT

Length: max. 255
Regular term: [.]{0,255}

Description: Textual interpretation of the response code in case of authorisation for the merchant and the customer. The language of the message is the same as the one defined in the LANG parameter, and may contain urlencoded latin2 (for european languages) or unicode (non-european) characters.

ANUM

Length: max. 6
Regular term: [a-zA-Z0-9]{0,6}

Description: Authorisation approval identifier on the issuer's side. Since ANUM is not a unique identifier at the issuer either, in case of any questions it should be associated with the amount and the time of the transaction.

CNUM

Length: max 19
Regular term: [0-9|X]{6}[X]+ [0-9|X]{4}

Description: Masked value of the credit/debit card entered on the paypage. Without explicit permission (default), all the card numbers are masked.

AUTH

Length: 1
Regular term: [0]

Description: Type of the authorisation channel. The 0 value indicates a web authorisation.

URL

Length: max. 255
Regular term: http[s]?://.+\..+/
Description: the merchant's url to which the bank directs the customer's browser after the payment is completed. This field cannot contain any parameters (`...?param1=value1&param2=value2...`), it must contain an absolute path and it must consist of a string that can be evaluated as an URL, in addition to the regular term (based on RFC 2396 and RFC 2732)

Example: `http://server.domain.com/path/to/script.ext`

LANG

Length: 2
Regular term: [a-zA-Z]{2}

Description: The code of the language used during the transaction. The payment page displayed in the customer's browser and the result of the authorisation can be read in this language.

Accepted values:

`HU`:	Hungarian
`EN`:	English
`DE`:	German
`IT`:	Italian
`FR`:	French
`ES`:	Spanish

PT: Portuguese
PL: Polish
CZ: Czech
SK: Slovak
RO: Romanian

HISTORY

Length: max. 255
Regular term: [0-9,]{0,255}

Description: A list of individual statuses of the processes executed during the transaction. The statuses are characterised by codes, reading is from left to right, and the individual statuses are separated by commas.

Example: 10,11,20,21,30

Possible values in the list:

10: the customer has arrived at the payment page
11: the customer has sent the completed payment page
12: transaction not approved by customer
14: the acceptor has rejected the authorisation request
20: the authorisation has started
21: the authorisation has been successful
22: the issuer has rejected the authorisation request
30: the merchant has received the result of the transaction
55: transaction selected for reversal due to time-out.
56: successful reversal
57: unsuccessful reversal

EXTRA01

Length: max. 50
Regular term: [0-9a-zA-Z áéíóöőúüűÁÉÍÓÖŐÚÜŰ"\+!\%/\(\)~`<>#\{\},\.\->*:_\\|\[łŁ$ß¤\]]{0,50}

Description: An optional extra parameter characteristic of the transaction, which helps in identifying the transaction during merchant settlements.

Contents of message types

The Bank's eCommerce server interprets the messages indicated in the following list. The listed parameters are included in the unencrypted message, provision of all the parameters is obligatory, except for EXTRA01:

Transaction initialisation (MSGT10)

- PID
- TRID
- MSGT, value: 10
- UID
- AMO
- CUR
- TS
- AUTH
- LANG
- URL
- EXTRA01

Transaction initialisation response (MSGT11)

- MSGT, value: 11
- PID
- TRID
- RC, possible values:
  - RC values: Successful initialisation
  - 01: Initialisation has failed due to other technical reasons
  - 02: TRID is already reserved

Redirecting the customer to the payment page (MSGT20)

- MSGT, value: 20
- PID
- TRID

Redirecting the customer from the payment page to the merchant (MSGT21)

- MSGT, value: 21
- PID
- TRID

Inquiring about the transaction outcome with confirmation and closing (MSGT32)

- MSGT, value: 32
- PID
- TRID
- AMO

Inquiring about the transaction outcome (MSGT33)

- MSGT, value: 33
- PID
- TRID
- AMO

Inquiring about the transaction outcome, response (MSGT31)

The format of the response is the same if either message 32 or message 33 is requested.

- MSGT, value: 31
- PID
- TRID
- AMO
- RC, possible values:
  - RC values: Successful authorisation
  - PR: The authorisation has not been completed yet
  - TO: Transaction failed due to time-out
  - Anything else: Unsuccessful authorisation (the error codes are classified in FAQ)
- RT
- ANUM
- CNUM (applies only to MSGT33 response)

Inquiry of transaction statuses (MSGT37)

- MSGT, value: 37
- PID
- TRID
- AMO

Inquiry on transaction statuses, response (MSGT38)

- MSGT, value: 38
- PID
- RC, possible values:
  - RC values: Successful inquiry
  - 01: Failed inquiry (in this case HISTORY is empty)
- HISTORY

Transaction status inquiry (MSGT70)

- MSGT, value: 70
- PID
- TRID
- AMO (the originally reserved amount)

Transaction status inquiry, response (MSGT71)

- MSGT, value: 71
- PID
- TRID
- AMO
- RC (result of the original authorisation)
- RT
- STATUS, possible values:
  - 10: The transaction has been authorised but not yet debited
  - 30: The transaction has been automatically debited
  - 40: The transaction has been reversed (using message 74)
  - 50: The transaction has been refunded
  - 60: The transaction has been closed
  - 99: An error occurred during processing
- CURAMO2, its value is the amount to be retransferred
- ANUM

Reversal of authorised transaction (MSGT74)

- MSGT, value: 74
- PID
- TRID
- AMO

Reversal of authorised transaction, response (MSGT75)

- MSGT, value: 75
- PID
- TRID
- AMO
- STATUS (see MSGT71)

Retransfer of debited transaction (MSGT78)

- MSGT, value: 78
- PID
- TRID
- AMO, its value is the debited amount

Retransfer of debited transaction, response (MSGT79)

- MSGT, value: 79
- PID
- TRID
- AMO
- RC (result of the original authorisation)
- RT
- STATUS (see MSGT71)
- ANUM

Setting a partial amount to be refunded from the debited transaction amount (MSGT80)

- MSGT, value: 80
- PID
- TRID
- AMOORIG, its value is the last set retransferable amount (default value: 0)
- AMONEW, its value is the retransferable amount intended to be set

Setting a partial amount to be refunded from the debited transaction amount, response (MSGT81)

- MSGT, value: 81
- PID
- TRID
- AMO, its value is the set retransferable amount
- STATUS (see MSGT71)

Other error codes undefined in the message type

If the bank's server is unable to interpret the received data, the response message will contain an unencrypted error code. These error codes have the following meanings:

RC=S01:	Error during acceptance of the received data.
RC=S02:	Data error.
RC=S03:	Undecipherable request.
RC=S04:	Database error.
RC=S05:	Processing error.
RC=S06:	Encryption error.
RC=D01:	Incorrect parameter.
RC=D02:	Undecipherable request.
RC=D03:	Wrong order. (Invalid message type request received.)
RC=D04	Unauthorised message type
RC=D05:	This message type request has already been served
RC=D06:	Unknown transaction.
RC=D07:	Incorrect data format.
RC=D08:	Data error

Typically, Sxx messages contain errors detected during decryption, and Dxx messages contain errors detected during the processing following a successful decryption (typically due to recording of incoming data in non-compliance with the specification).

Encryption of messages

Messages between the merchant and the bank must be sent in encrypted format. The ekiCrypt functions library and the related sakide application provided by the bank as an attachment to this documentation can be used for encrypting/decrypting messages. If neither solution is feasible, encryption may also be implemented in the manner detailed below, using native tools.

1.1. Encryption key

Encryption is 3DES-based. The encryption key file provided by the bank contains the following information:

Name	Length (byte)	Description
Key ID	4	File name (EKI'\0')
Key format version	2	0x00 0x02
STORE ID	4	This is the same as the file name (e.g. in the case of CIB.des: CIB'\0')
Time of key generation	4	number of seconds passed since 01.01.1970
First key	8	
Second key	8	
Initialisation vector	8	

In the case of native solutions only the last 3 elements need be taken into account. During the development cycle the store receives 3 keys:

- Test key: this is necessary for integration testing and it is only valid for the bank's test server
- Activation key: This is issued by the bank after successful testing, and it is only valid for the bank's activation server

Since the test key and the activation key have identical names, these should be kept separately (e.g. in separate directories).

Encryption algorithm

The use of the sakide utility included in the package and the sakicrypt functions library is recommended for encryption, although, if necessary, the protocol may also be replaced by native tools. The following is a description of a message encryption and the related partial result, step by step.

assembling an unencrypted message

The unencrypted message must contain the parameters indicated in the above description, in any order, in a parameter=value format, the parameters being combined with ’&’ signs (query string). The message used in this example is as follows:

```
PID=IEB0001&TRID=1234567812345678&MSGT=10&UID=IEB00000000&AMO=1000&CUR=HUF&TS=20131
231235959&AUTH=0&LANG=HU&URL=http://dev.bolt.hu/shop/frombank.asp
```

The key part of the key file used for encryption (the last 24 bytes in groups of 8) is:

```
00000000   54 E8 17 70 06 E1 18 77   T..p...w
00000008   51 57 C9 3A E0 0A A3 3D   QW.:...=
00000010   E4 48 CC 19 CD 62 EC 7E   .H...b.~
```

URL coding of the message

Each character must be URL-encoded, except for the ’&’ and ’=’ characters.

```
00000000   50 49 44 3D 49 45 42 30  30 30 31 26 54 52 49 44   PID=IEB0001&TRID
00000010   3D 31 32 33 34 35 36 37  38 31 32 33 34 35 36 37   =123456781234567
00000020   38 26 4D 53 47 54 3D 31  30 26 55 49 44 3D 49 45   8&MSGT=10&UID=IE
00000030   42 30 30 30 30 30 30 30  30 26 41 4D 4F 3D 31 30   B00000000&AMO=10
00000040   30 30 26 43 55 52 3D 48  55 46 26 54 53 3D 32 30   00&CUR=HUF&TS=20
00000050   31 33 31 32 33 31 32 33  35 39 35 39 26 41 55 54   131231235959&AUT
00000060   48 3D 30 26 4C 41 4E 47  3D 48 55 26 55 52 4C 3D   H=0&LANG=HU&URL=
00000070   68 74 74 70 25 33 41 25  32 46 25 32 46 64 65 76   http%3A%2F%2Fdev
00000080   2E 62 6F 6C 74 2E 68 75  25 32 46 73 68 6F 70 25   .bolt.hu%2Fshop%
00000090   32 46 66 72 6F 6D 62 61  6E 6B 2E 61 73 70         2Ffrombank.asp
```

Calculating and attaching of a CRC32

A CRC32 checksum must be calculated from the URL-encoded string, and its binary format must be attached to the string.

The CRC32 value of the above message is: `2CAFE8F8`

Attached to the end of the message:

```
00000000   50 49 44 3D 49 45 42 30  30 30 31 26 54 52 49 44   PID=IEB0001&TRID
00000010   3D 31 32 33 34 35 36 37  38 31 32 33 34 35 36 37   =123456781234567
00000020   38 26 4D 53 47 54 3D 31  30 26 55 49 44 3D 49 45   8&MSGT=10&UID=IE
00000030   42 30 30 30 30 30 30 30  30 26 41 4D 4F 3D 31 30   B00000000&AMO=10
00000040   30 30 26 43 55 52 3D 48  55 46 26 54 53 3D 32 30   00&CUR=HUF&TS=20
00000050   31 33 31 32 33 31 32 33  35 39 35 39 26 41 55 54   131231235959&AUT
00000060   48 3D 30 26 4C 41 4E 47  3D 48 55 26 55 52 4C 3D   H=0&LANG=HU&URL=
00000070   68 74 74 70 25 33 41 25  32 46 25 32 46 64 65 76   http%3A%2F%2Fdev
00000080   2E 62 6F 6C 74 2E 68 75  25 32 46 73 68 6F 70 25   .bolt.hu%2Fshop%
00000090   32 46 66 72 6F 6D 62 61  6E 6B 2E 61 73 70 2C AF   2Ffrombank.asp,.
000000A0   E8 F8                                              ..
```

Pad for encryption

If the length of the resulting string is not an integer multiple of 8, the binary form of the number of characters necessary for the supplementation (e.g. in the case of '6': 0x06) must be added to the end of a string, the number of characters is often`... 0x06 0x06 0x06 0x06 0x06 0x06`).

```
00000000   50 49 44 3D 49 45 42 30  30 30 31 26 54 52 49 44   PID=IEB0001&TRID
00000010   3D 31 32 33 34 35 36 37  38 31 32 33 34 35 36 37   =123456781234567
00000020   38 26 4D 53 47 54 3D 31  30 26 55 49 44 3D 49 45   8&MSGT=10&UID=IE
00000030   42 30 30 30 30 30 30 30  30 26 41 4D 4F 3D 31 30   B00000000&AMO=10
00000040   30 30 26 43 55 52 3D 48  55 46 26 54 53 3D 32 30   00&CUR=HUF&TS=20
00000050   31 33 31 32 33 31 32 33  35 39 35 39 26 41 55 54   131231235959&AUT
00000060   48 3D 30 26 4C 41 4E 47  3D 48 55 26 55 52 4C 3D   H=0&LANG=HU&URL=
00000070   68 74 74 70 25 33 41 25  32 46 25 32 46 64 65 76   http%3A%2F%2Fdev
00000080   2E 62 6F 6C 74 2E 68 75  25 32 46 73 68 6F 70 25   .bolt.hu%2Fshop%
00000090   32 46 66 72 6F 6D 62 61  6E 6B 2E 61 73 70 2C AF   2Ffrombank.asp,.
000000A0   E8 F8 06 06 06 06 06 06                            ........
```

3DES encryption

The padded message must be encrypted using the keys available in the key file, based on the 3DES CBC method.

```
00000000   4A 48 7B 6A 81 4A 55 52  52 FC 91 14 D0 4A 6D 8E   JH{j.JURR....Jm.
00000010   28 4D 46 90 CA 99 66 EF  52 2C 14 3E 7F 85 4A BF   (MF...f.R,.>⌂.J.
00000020   C1 84 96 0B 18 D2 83 D2  1E 7F F7 77 E3 C3 1C 7E   .........⌂.w...~
00000030   AF A3 C4 0D 20 EF 8D 02  64 EA 0F 8E 0C BD 35 7F   .... ...d.....5⌂
00000040   FA C9 FA 44 A3 33 41 05  3B E4 19 E9 BF 11 4F E5   ...D.3A.;.....O.
00000050   AA C5 A4 40 A1 49 08 49  D5 CA 6F 70 BE F9 DF 80   ...@.I.I..op....
00000060   83 19 C9 7B 20 F7 FD 4F  CE E8 9E D8 3B 8A 61 62   ...{ ..O....;.ab
00000070   04 23 7F 23 8F 1A 86 76  40 22 D6 70 07 A3 DF FF   .#⌂#...v@".p....
00000080   B4 39 FF 17 ED 73 43 D6  38 DE 2F 8F 83 89 07 71   .9...sC.8./....q
00000090   A3 9E 23 AC 15 63 19 ED  83 C9 CB DE 6C 9F F7 55   ..#..c......l..U
000000A0   88 AD 52 8A 11 40 E4 30                            ..R..@.0
```

Pad for Base64 coding

The received encrypted string must be expanded, in the manner detailed above, to a length divisible by 3.

```
00000000   4A 48 7B 6A 81 4A 55 52  52 FC 91 14 D0 4A 6D 8E   JH{j.JURR....Jm.
00000010   28 4D 46 90 CA 99 66 EF  52 2C 14 3E 7F 85 4A BF   (MF...f.R,.>⌂.J.
00000020   C1 84 96 0B 18 D2 83 D2  1E 7F F7 77 E3 C3 1C 7E   .........⌂.w...~
00000030   AF A3 C4 0D 20 EF 8D 02  64 EA 0F 8E 0C BD 35 7F   .... ...d.....5⌂
00000040   FA C9 FA 44 A3 33 41 05  3B E4 19 E9 BF 11 4F E5   ...D.3A.;.....O.
00000050   AA C5 A4 40 A1 49 08 49  D5 CA 6F 70 BE F9 DF 80   ...@.I.I..op....
00000060   83 19 C9 7B 20 F7 FD 4F  CE E8 9E D8 3B 8A 61 62   ...{ ..O....;.ab
00000070   04 23 7F 23 8F 1A 86 76  40 22 D6 70 07 A3 DF FF   .#⌂#...v@".p....
00000080   B4 39 FF 17 ED 73 43 D6  38 DE 2F 8F 83 89 07 71   .9...sC.8./....q
00000090   A3 9E 23 AC 15 63 19 ED  83 C9 CB DE 6C 9F F7 55   ..#..c......l..U
000000A0   88 AD 52 8A 11 40 E4 30  03 03 03                  ..R..@.0...
```

Base64 coding

Base64-coding of the padded string based on RFC2045:

```
00000000   53 6B 68 37 61 6F 46 4B  56 56 4A 53 2F 4A 45 55   Skh7aoFKVVJS/JEU
00000010   30 45 70 74 6A 69 68 4E  52 70 44 4B 6D 57 62 76   0EptjihNRpDKmWbv
00000020   55 69 77 55 50 6E 2B 46  53 72 2F 42 68 4A 59 4C   UiwUPn+FSr/BhJYL
00000030   47 4E 4B 44 30 68 35 2F  39 33 66 6A 77 78 78 2B   GNKD0h5/93fjwxx+
00000040   72 36 50 45 44 53 44 76  6A 51 4A 6B 36 67 2B 4F   r6PEDSDvjQJk6g+O
00000050   44 4C 30 31 66 2F 72 4A  2B 6B 53 6A 4D 30 45 46   DL01f/rJ+kSjM0EF
00000060   4F 2B 51 5A 36 62 38 52  54 2B 57 71 78 61 52 41   O+QZ6b8RT+WqxaRA
00000070   6F 55 6B 49 53 64 58 4B  62 33 43 2B 2B 64 2B 41   oUkISdXKb3C++d+A
00000080   67 78 6E 4A 65 79 44 33  2F 55 2F 4F 36 4A 37 59   gxnJeyD3/U/O6J7Y
00000090   4F 34 70 68 59 67 51 6A  66 79 4F 50 47 6F 5A 32   O4phYgQjfyOPGoZ2
000000A0   51 43 4C 57 63 41 65 6A  33 2F 2B 30 4F 66 38 58   QCLWcAej3/+0Of8X
000000B0   37 58 4E 44 31 6A 6A 65  4C 34 2B 44 69 51 64 78   7XND1jjeL4+DiQdx
000000C0   6F 35 34 6A 72 42 56 6A  47 65 32 44 79 63 76 65   o54jrBVjGe2Dycve
000000D0   62 4A 2F 33 56 59 69 74  55 6F 6F 52 51 4F 51 77   bJ/3VYitUooRQOQw
000000E0   41 77 4D 44                                        AwMD
```

URL-encoding

The Base64-encoded string must then be URL-encoded (this time all the characters included in it).

```
00000000   53 6B 68 37 61 6F 46 4B  56 56 4A 53 25 32 46 4A   Skh7aoFKVVJS%2FJ
00000010   45 55 30 45 70 74 6A 69  68 4E 52 70 44 4B 6D 57   EU0EptjihNRpDKmW
00000020   62 76 55 69 77 55 50 6E  25 32 42 46 53 72 25 32   bvUiwUPn%2BFSr%2
00000030   46 42 68 4A 59 4C 47 4E  4B 44 30 68 35 25 32 46   FBhJYLGNKD0h5%2F
00000040   39 33 66 6A 77 78 78 25  32 42 72 36 50 45 44 53   93fjwxx%2Br6PEDS
00000050   44 76 6A 51 4A 6B 36 67  25 32 42 4F 44 4C 30 31   DvjQJk6g%2BODL01
00000060   66 25 32 46 72 4A 25 32  42 6B 53 6A 4D 30 45 46   f%2FrJ%2BkSjM0EF
00000070   4F 25 32 42 51 5A 36 62  38 52 54 25 32 42 57 71   O%2BQZ6b8RT%2BWq
00000080   78 61 52 41 6F 55 6B 49  53 64 58 4B 62 33 43 25   xaRAoUkISdXKb3C%
00000090   32 42 25 32 42 64 25 32  42 41 67 78 6E 4A 65 79   2B%2Bd%2BAgxnJey
000000A0   44 33 25 32 46 55 25 32  46 4F 36 4A 37 59 4F 34   D3%2FU%2FO6J7YO4
000000B0   70 68 59 67 51 6A 66 79  4F 50 47 6F 5A 32 51 43   phYgQjfyOPGoZ2QC
000000C0   4C 57 63 41 65 6A 33 25  32 46 25 32 42 30 4F 66   LWcAej3%2F%2B0Of
000000D0   38 58 37 58 4E 44 31 6A  6A 65 4C 34 25 32 42 44   8X7XND1jjeL4%2BD
000000E0   69 51 64 78 6F 35 34 6A  72 42 56 6A 47 65 32 44   iQdxo54jrBVjGe2D
000000F0   79 63 76 65 62 4A 25 32  46 33 56 59 69 74 55 6F   ycvebJ%2F3VYitUo
00000100   6F 52 51 4F 51 77 41 77  4D 44                     oRQOQwAwMD
```

Adding a prefix to the message

The following prefix must be inserted in front of the URL code:

```
PID=%PID%&CRYPTO=1&DATA=
```

Where `%PID%` is the identifier provided by the bank to the store. Consequently, the encrypted message will be the `DATA` value of the message to be sent.

```
00000000   50 49 44 3D 49 45 42 30  30 30 31 26 43 52 59 50   PID=IEB0001&CRYP
00000010   54 4F 3D 31 26 44 41 54  41 3D 53 6B 68 37 61 6F   TO=1&DATA=Skh7ao
00000020   46 4B 56 56 4A 53 25 32  46 4A 45 55 30 45 70 74   FKVVJS%2FJEU0Ept
00000030   6A 69 68 4E 52 70 44 4B  6D 57 62 76 55 69 77 55   jihNRpDKmWbvUiwU
00000040   50 6E 25 32 42 46 53 72  25 32 46 42 68 4A 59 4C   Pn%2BFSr%2FBhJYL
00000050   47 4E 4B 44 30 68 35 25  32 46 39 33 66 6A 77 78   GNKD0h5%2F93fjwx
00000060   78 25 32 42 72 36 50 45  44 53 44 76 6A 51 4A 6B   x%2Br6PEDSDvjQJk
00000070   36 67 25 32 42 4F 44 4C  30 31 66 25 32 46 72 4A   6g%2BODL01f%2FrJ
00000080   25 32 42 6B 53 6A 4D 30  45 46 4F 25 32 42 51 5A   %2BkSjM0EFO%2BQZ
00000090   36 62 38 52 54 25 32 42  57 71 78 61 52 41 6F 55   6b8RT%2BWqxaRAoU
000000A0   6B 49 53 64 58 4B 62 33  43 25 32 42 25 32 42 64   kISdXKb3C%2B%2Bd
000000B0   25 32 42 41 67 78 6E 4A  65 79 44 33 25 32 46 55   %2BAgxnJeyD3%2FU
000000C0   25 32 46 4F 36 4A 37 59  4F 34 70 68 59 67 51 6A   %2FO6J7YO4phYgQj
000000D0   66 79 4F 50 47 6F 5A 32  51 43 4C 57 63 41 65 6A   fyOPGoZ2QCLWcAej
000000E0   33 25 32 46 25 32 42 30  4F 66 38 58 37 58 4E 44   3%2F%2B0Of8X7XND
000000F0   31 6A 6A 65 4C 34 25 32  42 44 69 51 64 78 6F 35   1jjeL4%2BDiQdxo5
00000100   34 6A 72 42 56 6A 47 65  32 44 79 63 76 65 62 4A   4jrBVjGe2DycvebJ
00000110   25 32 46 33 56 59 69 74  55 6F 6F 52 51 4F 51 77   %2F3VYitUooRQOQw
00000120   41 77 4D 44                                        AwMD
```

That is, the message to be sent to the bank is as follows:

```
PID=IEB0001&CRYPTO=1&DATA=Skh7aoFKVVJS%2FJEU0EptjihNRpDKmWbvUiwUPn%2BFSr%2FBhJYLGNK
D0h5%2F93fjwxx%2Br6PEDSDvjQJk6g%2BODL01f%2FrJ%2BkSjM0EFO%2BQZ6b8RT%2BWqxaRAoUkISdXK
b3C%2B%2Bd%2BAgxnJeyD3%2FU%2FO6J7YO4phYgQjfyOPGoZ2QCLWcAej3%2F%2B0Of8X7XND1jjeL4%2B
DiQdxo54jrBVjGe2DycvebJ%2F3VYitUooRQOQwAwMD
```

Decryption requires that the inverse operation of each individual step (3DES is identical to its own inverse) be executed in reverse order.

Cryptographic functions library

The ekiCrypt library contains all the encryption algorithms necessary for bank communication. The library contains the following functions:

Encryption (ekiEncodeUrl)

```
INT ekiEncodeUrl (LPSTR inBuffer, INT inBufferSize,
                  LPSTR outBuffer, LPINT outBufferSize,
                  INT cryptoType, LPSTR keyFilePath)
```

The message generated in URL format is thoroughly checked, URL-encoded, DES-coded, uuencoded and adjusted to the URL format corresponding to the protocol. Under Win32, the DES key is stored in the registry or in a file; in other systems it is stored in a file.

Parameters:

`inBuffer:` The buffer containing the URL to be encrypted.

`InBufferSize:` The length of the URL to be encrypted.

`OutBuffer:` The buffer in which the encrypted URL will be stored.

`OutBufferSize:` Size of the result

Input: The size of the buffer reserved for the encrypted URL, minimum 4/3 * (inBufferSize + 12).

Output: The size of the encrypted URL.

`CryptoType:` The necessary encryption type.

0 - SSL

1 - DES

2 - RSA

3 - BROWSER

`KeyFilePath:` The path of the encrypting (DES) key file, if the key is stored in a file. NULL, if the key is stored in the registry (only under Win32)

Decrypting (ekiDecodeUrl)

```
INT ekiDecodeUrl (LPSTR inBuffer, INT inBufferSize,
                  LPSTR outBuffer, LPINT outBufferSize,
                  LPINT cryptoType, LPSTR keyFilePath);
```

Retrieves the encrypted data from a coded, URL-format input, uudecodes and DESes it, then performs various checks on it. It returns the message URL-encoded.

Parameters:

`inBuffer:` The buffer containing the encrypted URL.

`InBufferSize:` Length of the encrypted URL.

`OutBuffer:` The buffer in which the decrypted URL will be stored.

`OutBufferSize` Size of the result

Input: The size of the buffer reserved for the decrypted URL, minimum 4/3 * (inBufferSize + 12).

Output: Size of the decrypted URL.

`CryptoType`

If the value is not NULL, this buffer will store the encryption type of the resulting data, and the decrypted data will not contain the Crypto field.

If the value is NULL, the Crypto field will be left in the decrypted data.

`KeyFilePath:` The path of the encrypting (DES) key file, if the key is stored in a file. NULL, if the key is stored in the registry (only under Win32)

Key structure

The structure detailed below is not identical to the binary content of the key file, since it only contains a part of it for the `ekiGetKeyInfo` function.

```
typedef struct
{
   char           fname[13];       /* file name */
   INT            keySize;         /* key size */
   char            id[4];            /* internal ID - theoretically this is the same
as the marketId*/
   unsigned short version;         /* version */
   char           marketId[4];     /* store ID */
   time_t         creation_time;   /* time of key generation */
} TKeyInfo;
```

Key information inquiry (ekiGetKeyInfo)

```
INT EKIAPI ekiGetKeyInfo(TKeyInfo *ekiKeyInfo, char *keyFilePath,
                              char *boltId);
```

Returns the data of the encryption key in the TKeyInfo structure.

Parameters:

`ekiKeyInfo:`	Key data
`keyFilePath:`	Location of key file
`boltId:`	STORE ID

Possible return values:

UER_OK:	Ok.
UER_NOMEM:	Insufficient memory.
UER_BADSIZE:	Output buffer size (outBufferSize) too small.
UER_NOKEY:	Key not found.
UER_BADKEY:	Wrong key structure.
UER_NOFILE:	Key file not found.

Version number inquiry (ekiGetLibVersion)

INT EKIAPI ekiGetLibVersion(char *outBuffer, LPINT outBufferSize);

Inquiry of the encryption module version number.

Parameters:

`outBuffer:`	This is where the version number is written in x.y.z format, terminated by \0.
`outBufferSize:`	outBuffer size in bytes, leaving sufficient space for closing zeros.

Possible return values:

UER_OK:	Ok.
UER_NOMEM:	Insufficient memory.
UER_BADSIZE:	Output buffer size (outBufferSize) too small.

Encryption application (sakide)

Sakide is a command-line sample program that uses the library functions of ekiCrypt. Usage:

```
sakide [-e|-d] [-c <cryptotype>] [-i <-m <keyfile>>]
       [-p <path>] [-v] [-s <string>] [-S] [-u] [-V]
```

where

`-e` encryption operating mode

`-d` decryption operating mode

`-c <cryptotype>` the type of the encryption (typically 1)

`-m <keyfile>` key file name

`-i` provides information on the content of the key file

`-p <path>` path to the key

`-v` writes extra information on STDERR

`-s <string>` the messages to be processed

`-S` provides system information (if GNU libc is used)

`-u` performs a URL decoding of data before decryption.

`-V` indicates the version number of the application

If invoked without parameters, the application encrypts the data received from the STDIN, using the encryption key of the store identifier contained in the PID parameter indicated in the message (if PID=CIB0001 in the message, it uses the CIB.des file available in the current library).

Examples of usage of the sakide application:

```
./sakide –e –s „PID=CIB0001&TRID=1234123412341234&MSGT=20”

echo „PID=CIB0001&TRID=1234123412341234&MSGT=20”|./sakide

cat cleartext.dat|./sakide –v 2>&1
```

The latter provides information to the STDOUT channel (in practice, the STDERR is directed to STDOUT, which is useful in error detection).

Two executable files are available for most of the supported operating systems, one of which is dynamically linked and one is statically linked (the name of this latter one is sakide.static or sakide_static). This latter one does not require any functions libraries at system level, that is, it can be applied independently of the version of the operating system.

3D Secure

For security purposes, CIB Bank’s eCommerce acceptance server is provided with a 3D Secure module. This service is ensured to all those cardholders who have a card provided with a Verified by Visa (VISA) and SecureCode (MasterCard or Maestro) security component. In practice, 3D Secure means a transfer from the bank’s payment page to the website of the card-issuing bank (or the card company commissioned by it), where the customer can enter his/her password related to the card that has been previously agreed with the issuing bank. The method of agreement can vary from bank to bank. A successful transaction can only be created by providing the correct password, which is a way of protecting the cardholder ant his/her card.

Possible errors, suggestions

- Prior to decoding, the data received from the bank must not be URL-decoded. Incoming data must first be decoded by transferring them to ekiDecodeUrl, and then the resulting data have to be URL-decoded. With some web servers it may happen that message 21 obtained in step [9] is transferred to the server-side program URL-decoded. At this step, it is strongly recommended that the obtained parameters be URL-encoded prior to processing.
- If connection is interrupted at any point of the protocol (e.g. the customer disconnects the line or changes URLs, or there is a broken line between the bank and the store), the transaction will terminate in the bank by time-out. Although the period of time-out in the bank is 10-15 minutes, this period must, based on experience indicating that this is not always appropriate, be programmed as modifiable. The control is not returned by the bank to the store. If it is not clear to the bank whether the transaction has been successful, the bank launches a ‘reversal’ after time-out, then closes the transaction, and the transaction data will no longer be available to the counterparties. In this case the transaction must end with a time-out in the store as well. This means that a transaction can only be considered successfully closed if all the authorisation steps have been completed (the last one is MSGT=31 received in response to MSGT=31).
- The transaction status must be monitored by both parties (the BANK and the STORE) so that they can both establish with certainty which state a transaction is currently in, and whether the defined period has elapsed. If the transaction times out in any of its phases, the bank’s duty is to reverse any successful authorisations and to close any unsuccessful transactions, and the store’s duty is to delete the order and to close the registration of the transaction on the store side. The status can be monitored using message 33.
- It is important for the store to log each transaction step, making it possible for both the bank and the store to retrospectively search the various steps through which a given transaction has reached its final status.

Contact information

Name	Phone	E-mail	Availability	Area
Customer call centre	+36-1-399-8899		Daily, between 0:00 - 24:00	Request for information, forwarding of technical inquiries
Bank's merchant	See:www.cib.hu, list of branches	See:www.cib.hu, list of branches	During the bank's open hours	Contracts, invoice-related issues
Developer support		ecommerce@cib.hu	During the bank's open hours	Technical assistance during development / testing