Secure ePayments Card Processing

CPI specification

HSBC

The world's local bank

Introduction

The Cardholder Payment Interface (CPI) is one of the products within HSBC Secure ePayments, allowing you to take card payments from customers via the Internet.

A customer will browse your web pages selecting the item(s) they wish to purchase. When the customer is ready to pay, they will be transferred to the HSBC hosted payment pages for entry of card details. Transactional processing and cardholder authentication occurs while the customer is within the payment pages. The customer is then returned to your site to see if the transaction was successful.

To enhance security of transactional information, the CPI service employs the highest levels of encryption for connections and utilises a message authentication code (MAC – also known as the OrderHash). You are required to have access to secure web space.

The service allows you to participate in Verified by Visa and MasterCard SecureCode programs. These programs are designed to help protect both you and your customers when participating in an Internet transaction. In addition, you have the ability to use the Address Verification Service (AVS) to make a more informed decision as to the standing of an order.

Transactions processed through the CPI service pass through a complex set of fraud rules. This set of rules is maintained and updated by HSBC, transactions that meet certain criteria will be flagged for your attention.

The CPI service also provides access to the Virtual Terminal. This interface allows you to view order details, complete order processing, display historic data and review transactions that have been flagged for attention.

The CPI service is not designed for use within a call centre environment, or equivalent.
The customer must enter the details themselves in order for you to comply with the terms and conditions of the service.

The CPI service is not suitable for automatic recurring / instalment payments.

Integration requirements

The CPI service has the following fundamental requirements:

You must have access to secure web space (SSL / 128 bit), and must be able to use one of the supplied libraries to generate an OrderHash for each transaction.

Whilst integrating the service you will require the following:
StorefrontId (ClientId – sent by email)
CPI Hash Key ("shared secret" – sent by letter)
Username (for Virtual Terminal access, sent by email)
Password (for Virtual Terminal access, provided by the merchant during the sales process)

A small number of "test stores" are available for limited use – for full details please contact the Secure ePayments helpdesk.

CPI Hash Key

The CPI Hash Key (also known as the "shared secret") is sent in a letter to you after your CPI service is activated. The value is exactly 32 characters in length, case sensitive, and specific both to you and the associated CPI service. If you have more than one CPI service (e.g. multiple currencies) you will receive one CPI Hash Key for each CPI service.
It is important to keep this information confidential. The Hash Key in conjunction with the supplied libraries enables the generation of the OrderHash (Message Authentication Code).
The Hash Key will need to be securely embedded / located within the integration and not be externally retrievable/accessible.

OrderHash libraries and samples

There are three supplied libraries / interfaces for the generation of the OrderHash – C, Java and COM.

The C library (CcCpiTools.dll) is supported on the following platforms:
- HP-UX 11.0 and 11.11
- Sun Solaris 2.7, 2.8, and 2.9
- Red Hat Linux 7.3 or higher
- Windows 2000
- 32-bit Windows COM environment(s)

The Java library (CcCpiTools.jar) supports Java Virtual Machine 1.2.2 on the platforms listed below. In addition, Java 1.4.2 is supported under the 1.2.2 compiler on the same platforms:
- HP-UX 11.0 and 11.2
- Sun Solaris 2.7 and 2.8
- Red Hat Linux 7.3 or higher
- Microsoft Windows NT 4.0 and Windows 2000

The COM object (CcCpiCOM.dll) is supported on the following platforms:
- 32-bit Windows COM environment(s)

There are a number of requirements for the “installation” of the libraries – these are discussed below.

Additionally, a number of samples are supplied to demonstrate the functionality of the libraries with respect to OrderHash generation. The files are simple in nature and not intended for production use.

Deploying sample files

The procedure for deploying the files needed for testing an integration package varies, depending on the programming language used.

Platform	Language	Sample HTML form	OrderHash Generation	Sample Results Page	Dependencies
Java	Java	java/sample/ sample.html	OrderHash.jsp	results.jsp	cpitools.jar
HP-UX	C	c/HP-UX/sample/ sample.html	CcOrderHash.e	CcResults.e	libCcCpiTools.sl
Linux	C	c/linux/sample/ sample.html	CcOrderHash.e	CcResults.e	libCcCpiTools.so
SunOS	C	c/SunOS/sample/ sample.html	CcOrderHash.e	CcResults.e	libCcCpiTools.so
Windows	C	c/winnt/sample/ sample.html	CcOrderHash.exe	CcResults.exe	CcCpiTools.dll
Windows	COM (ASP)	com/sample/ sample.html	OrderHash.asp	results.asp	CcCpiCom.dll (CcCpiTools.dll)
Windows	COM (CFML)	com/sample/ sample.html	OrderHash.cfm	results.cfm	CcCpiCom.dll (CcCpiTools.dll)

Java

Copy sample.html to a web enabled directory.

Copy OrderHash.jsp to a web enabled directory with execute permissions (the jsp pages will need to be able to read from and write to a file in the local directory).

Copy results.jsp to a secure (https) web enabled directory with execute permissions (the jsp pages will need to be able to read from and write to a file in the local directory).

Add CcCpiTools.jar to the CLASSPATH environment variable (either by suitable file location, or direct addition). It may be necessary for you to speak to your host regarding suitable location / installation of this file.

The jsp sample files look for the shared secret in a file called ss.txt within a subfolder called WEB-INF.

C

HP-UX

Copy sample.html to a web enabled directory.

Copy OrderHash.e to a web enabled directory with execute permissions and without “directory browse” permissions (the pages will need to be able to read from and write to a file in the local directory).

Copy results.e to a secure (https) web enabled directory with execute permissions and without "directory browse" permissions (the pages will need to be able to read from and write to a file in the local directory).

Add the path to libCcCpiTools.sl to the SHLIB_PATH environment variable (either by suitable file location, or direct addition). It may be necessary for you to speak to your host regarding suitable location / installation of this file.

The executable sample files look for the shared secret in a file called ss.txt within the same folder.

SunOS

Copy sample.html to a web enabled directory.

Copy OrderHash.e to a web enabled directory with execute permissions and without "directory browse" permissions (the pages will need to be able to read from and write to a file in the local directory).

Copy results.e to a secure (https) web enabled directory with execute permissions and without "directory browse" permissions (the pages will need to be able to read from and write to a file in the local directory).

Add the path to libCcCpiTools.so to the LD_LIBRARY_PATH environment variable (either by suitable file location, or direct addition). It may be necessary for you to speak to your host regarding suitable location / installation of this file.

The executable sample files look for the shared secret in a file called ss.txt within the same folder.

Linux

Copy sample.html to a web enabled directory.

Copy OrderHash.e to a web enabled directory with execute permissions and without "directory browse" permissions (the pages will need to be able to read from and write to a file in the local directory).

Copy results.e to a secure (https) web enabled directory with execute permissions and without "directory browse" permissions (the pages will need to be able to read from and write to a file in the local directory).

Add the path to libCcCpiTools.so to the LD_LIBRARY_PATH environment variable (either by suitable file location, or direct addition). It may be necessary for a merchant to speak to their host regarding suitable location / installation of this file.

The executable sample files look for the shared secret in a file called ss.txt within the same folder.

WinNT

Copy sample.html to a web enabled directory.

Copy OrderHash.exe to a web enabled directory with execute permissions and without "directory browse" permissions (the pages will need to be able to read from and write to a file in the local directory).

Copy results.exe to a secure (https) web enabled directory with execute permissions and without "directory browse" permissions (the pages will need to be able to read from and write to a file in the local directory).

Add the path to CcCpiTools.dll to the PATH environment variable (either by suitable file location, or direct addition). It may be necessary for you to speak to your host regarding suitable location / installation of this file.

The executable sample files look for the shared secret in a file called ss.txt within the same folder.

COM

Copy sample.html to a web enabled directory.

Copy OrderHash.asp (or OrderHash.cfm) and to a web enabled directory with execute permissions and without "directory browse" permissions (the pages will need to be able to read from and write to a file in the local directory).

Copy results.asp (or results.cfm) to a secure (https) web enabled directory with execute permissions and without "directory browse" permissions (the pages will need to be able to read from and write to a file in the local directory).

Add the path to CcCpiTools.dll to the PATH environment variable (either by suitable file location, or direct addition). It may be necessary for you to speak to your host regarding suitable location / installation of this file.

Register the CcCpiCOM.dll file (either via MMC, regsvr32 or similar)

The executable sample files look for the shared secret in a file called ss.txt in the folder above their location (e.g. if the sample files are in c:\inetpub\wwwroot, ss.txt should be in c:\inetpub).

Edit sample html file

Open the sample.html file and edit the following details:

Change the form action so that it points to the OrderHash sample. For example:

action="https://MerchantServer/cgi-bin/cpisample/CcOrderHash.exe"

Change the value for the hidden field CpiUrl so that it points to the full URL of the CPI Server. For example:

value="https://www.cpi.hsbc.com/servlet"

Change the value for the text field CpiReturnUrl so that it points to the full URL of the Results sample. For example:

action="https://MerchantServer/cgi-bin/cpisample/CcResults.exe"

Change the value for the text field CpiDirectResultUrl so that it points to the full URL of the Results sample. For example:

action="https://MerchantServer/cgi-bin/cpisample/CcResults.exe"

Change the value for the text field StorefrontId so that it contains the StorefrontId configured at the CPI. For example:

value="UK12345678CUR"

Edit the shared secret file

Open (or create) the ss.txt file and enter the Hash Key that corresponds to the StorefrontId entered in sample.html above. Save the file in the correct location (see above for details).

Important: The shared secret must be stored in a secure location that is not accessible by an external user either by viewing page source or by attempting to enter a URL to the secret's location.

Run a Test Transaction

Use the following steps to run a test transaction:
Point a browser to the sample.html page.
Enter values as desired and click the Submit button.
Ensure that the OrderHash value has been created.
Click the Submit button to send the data to the CPI service.
Verify that CpiResponse.txt directory has entries for both the CpiDirectResultUrl and CpiReturnUrl.

The CpiResponse.txt file is used by the various "results" sample pages, and will contain a copy of the CPI results that were displayed by those pages. It is located as follows for the sample code:

Java example - In the ../logs directory relative to where the jsp pages are located.

C example - In the directory where the executable samples are located.

COM example - In the document root directory of the server.

Page and Data Flow

The following is a simplified overview of the page and data flow for the CPI service.

You gather transactional data and generate an OrderHash.

Transactional data and OrderHash is POSTed – customer is transferred to HSBC.

Customer is prompted to enter card details.

Customer Authentication is attempted (Verified By Visa / MasterCard SecureCode)

Customer given transaction summary and choice to proceed (or cancel).

Payment authorisation attempted – data POSTed to CpiDirectResultUrl (CPI waits for acknowledgement from your site)

Customer presented with page to return to your site.

Customer returned to your site for transaction result (customer returned to CpiReturnUrl, data POSTed to CpiReturnUrl)

Please note:

For a completed transaction the transaction result data is provided twice.

If a cardholder “cancels” they are returned to CpiReturnUrl.

HSBC do not inform the customer of the status of the order / transaction result.

When data is POSTed to CpiDirectResultUrl, the CPI will wait for acknowledgement of receipt of data (e.g. <html></html>) – if no such response is received the CPI will wait for a standard “time-out” before proceeding.

Options

You can choose to gather address related information within your site, or alternatively it can be taken within the HSBC hosted pages. Address details are not returned to your site.

You can choose whether you wish the “Welcome to HSBC Secure ePayments” (splash page) to be presented to your customers. This is always “active” by default – please contact the Secure ePayments helpdesk if you want to amend this option.

Implementing the OrderHash

Information between your web site and CPI is transmitted via secure POST (HTTPS) operations. Your web site and the CPI are required to generate an OrderHash to help ensure the validity of the data being transmitted.
The web site and the CPI perform the following steps to generate and validate data.

Your web site collects data about an order and information necessary for processing the order with the CPI.

The web site calls a program that generates an OrderHash value based on the data to be submitted to the CPI.

The web site sends the OrderHash value and the data used in its creation to the CPI using a secure POST operation.

As the CPI validates the input, it generates its own OrderHash value. If the submitted OrderHash value does not match the generated value, the order is immediately rejected.

After the CPI has processed the order, it generates a new OrderHash value based on the return fields. The OrderHash value and other return fields are returned to you using another secure POST.

Your web site should generate a new OrderHash value based on the return data to confirm the validity of the data. (The returned OrderHash value must be excluded when generating the confirmation OrderHash value.) The web site should reject the order if the return OrderHash values do not match.

Only the named CPI fields (detailed below) must be included within the OrderHash generation process. These values must not change between OrderHash generation and transmission via secure POST.

Generating the Merchant POST

The order information is submitted, including the OrderHash, to the CPI using a secure (SSL) POST.

The POST can contain two types of data:

Merchant Security and Order Details

Billing and Shipping information

Merchant security and order details must always be submitted. The OrderHash is included in this category.

Billing and shipping information is optional. However, if any data from this section is submitted, then all required fields within this section must be supplied.

Note: The following sections describe the fields that can be sent to the CPI in the secure POST operation. Only these fields must be used within the OrderHash generation process. Including other information causes the CPI to generate an OrderHash value that does not match the value generated by the your site, and the transaction is immediately rejected.

Table 1 Merchant Security and Order Details

Field Name	Description	Value
CpiDirectResultUrl	**Required.** The URL for returning data to the merchant. Must be a secure (https) location.	1024 characters maximum Must begin with https://
CpiReturnUrl	**Required.** The web page to display on the customer's web browser after the CPI transaction is complete. Must be a secure (https) location.	1024 characters maximum. Must begin with https://
MerchantData	**Optional.** Any additional data the merchant wants to include in the hash, such as session data.	512 characters maximum
Mode	Must be one of the following values: **P** - Production mode. The customer will be billed for the order. **T** - Test mode. No money will be taken.	All merchants are initially set to Test mode within the CPI service. You must request to "go live" with the service prior to using the value **P**. Must be either **P** or **T**
OrderDesc	**Required.** A free-form description of the order.	1 character minimum. 54 characters maximum
OrderHash	**Required.** The value returned from the Hash Generator.	Exactly 28 characters.
OrderId	**Required.** A unique identifier for the order. You must provide this value.	1 character minimum. 36 characters maximum.

Field Name	Description	Value
PurchaseAmount	Required. The amount of the purchase, expressed as a positive integer. **If the currency contains decimals, omit the decimal point.** For example, a 9999999 value in Sterling (GBP), which has two decimal places, is interpreted as 99999.99 pounds.	Positive integer within the range 1 to 9999999.
PurchaseCurrency	**Required.** The currency code for the order. For example, 826 represents the United Kingdom.	3 digits. See note 2 below.
StorefrontId	Required. The StorefrontId of your CPI service. For example, UK12345678CUR.	13 characters.
TimeStamp	**Required.** Date and time of the purchase expressed as the number of milliseconds elapsed since January 1, 1970 00:00:00 GMT. For example, the TimeStamp for 00:00:00 GMT, 1st September 2003 is 1062374400000.	Positive integer value, currently 13 digits in length). See note 3 below.
TransactionType	**Optional.** Must be one of the following values: **Auth** or **Capture**	**Auth** or **Capture** See note 4 below.
UserId	**Optional.** A unique identifier associated with a user.	32 characters maximum.

Notes:

1. The valid character set for data is ISO8859-1.

2. A full list of PurchaseCurrency codes are detailed in the Secure ePayments Card Processing Reference Document.

3. TimeStamp – The window of opportunity for a valid TimeStamp is +/-1 hour from the based on the actual GMT time of submitting the Merchant POST.

4. TransactionType:

An *Auth* transaction places a reserve on the cardholders open-to-buy balance, the cardholders available balance remains unchanged. Once the goods have been confirmed as "shipped", you will use the Virtual Terminal to mark the order as "shipped". This process then automatically marks the funds ready for settlement. (This corresponds to a "PreAuth" in the Virtual Terminal.)

A *Capture* transaction verifies the cardholder's account to be in good standing, and automatically marks the funds ready for settlement. This is typically used for goods that do not need to be physically shipped (for example, a software download). (This corresponds to an "Auth" in the Virtual Terminal.)

Table 2 Billing and Shipping information

Field Name	Description	Value
BillingAddress1	**Required.** The first line of the customer's address. This field typically includes the street name and number or box number.	60 characters maximum.
BillingAddress2	**Optional.** Additional address information.	60 characters maximum
BillingCity	**Required.** The city in which the customer resides.	25 characters maximum.
BillingCountry	**Required.** The 3-digit country code. For example, 826 represents the United Kingdom.	3 characters. See note 2 below.
BillingCounty	**Optional.** The state or county in which the customer resides.	25 characters maximum.
BillingFirstName	**Required.** The customer's first name.	32 characters maximum.
BillingLastName	**Required.** The customer's last name.	32 characters maximum.
BillingPostal	**Required.** The customer's postal code.	20 characters maximum.
ShopperEmail	**Required.** The customer's e-mail address. Must be a valid email form (i.e. a@b.com)	64 characters maximum.
ShippingAddress1	**Required.** The first line of the recipient's address. This field typically includes the street name and number or box number.	60 characters maximum.
ShippingAddress2	**Optional.** Additional address information.	60 characters maximum.
ShippingCity	**Required.** The city in which the recipient resides.	25 characters maximum.
ShippingCountry	**Required.** The 3-digit country code. For example, 826 represents the United Kingdom.	3 characters. See note 2 below.
ShippingCounty	**Optional.** The state or county in which the recipient resides.	25 characters maximum.
ShippingFirstName	**Required.** The recipient's first name.	32 characters maximum.
ShippingLastName	**Required.** The recipient's last name.	32 characters maximum.
ShippingPostal	**Required.** The recipient's postal code.	20 characters maximum.

Notes:

1. The valid character set for data is ISO8859-1.

2. A full list of BillingCountry and ShippingCountry codes are detailed in the Secure ePayments Card Processing Reference Document.

3. An AVS check requires BillingAddress1 and BillingPostal to contain valid data.

4. ShopperEmail must be in a valid email format (a@b.com)

CPI Return POST

When the CPI has completed its series of tasks, data is POSTed to the your server. A completed transaction will send data to both CpiDirectResultUrl and CpiReturnUrl.

The PurchaseDate and CpiResultsCode fields are always returned via a secure POST to your web site.

Table 3 CPI Return POST

Field Name	Description	Value
CpiResultsCode	Enforced return. Transaction Status	Numeric value between 0 and 16. See separate table for more information
PurchaseDate	Enforced return. Processing date and time of the purchase expressed as the number of milliseconds elapsed since January 1, 1970 00:00:00 GMT.	Positive integer value, currently 13 digits in length).
MerchantData	Optional return. Value provided by you during Merchant POST.	512 characters maximum
OrderHash	Optional return. System Generated.	Exactly 28 characters.
OrderId	Optional return. Value provided by you during Merchant POST.	36 characters maximum
PurchaseAmount	Optional return. Value provided by you during Merchant POST.	18 digits maximum.
PurchaseCurrency	Optional return. Value provided by you during Merchant POST.	3 characters.
ShopperEmail	Optional return. Value optionally provided by you during Merchant POST.	64 characters maximum.
StorefrontId	Optional return. Value provided by you during Merchant POST.	13 characters.

In order to avoid the possibility of data tampering, you are advised to check the following:

1. Data provided within the Merchant POST matches the corresponding data in the CPI Return POST
2. A returned OrderHash is valid when generated from the accompanying data. (The returned OrderHash is constructed from the accompanying data in a similar method to the value created by your site for use in the Merchant POST).

CpiResultsCode Breakdown

The CpiResultsCode field indicates whether the transaction succeeded. The following table lists the possible return values of the CpiResultsCode field:

Table 4 CpiResultsCode Values

Code	Description
0	The transaction was approved.
1	The user cancelled the transaction.
2	The processor declined the transaction for an unknown reason.
3	The transaction was declined because of a problem with the card. For example, an invalid card number or expiration date was specified.
4	The processor did not return a response.
5	The amount specified in the transaction was either too high or too low for the processor.
6	The specified currency is not supported by either the processor or the card.
7	The order is invalid because the order ID is a duplicate.
8	The transaction was rejected by FraudShield.
9	The transaction was placed in Review state by FraudShield (see note 1 below).
10	The transaction failed because of invalid input data.
11	The transaction failed because the CPI was configured incorrectly.
12	The transaction failed because the Storefront was configured incorrectly.
13	The connection timed out.
14	The transaction failed because the cardholder's browser refused a cookie.
15	The customer's browser does not support 128-bit encryption.
16	The CPI cannot communicate with the Payment Engine.

Note:

1. When a transaction is placed in Review state by FraudShield, it can be authorised, but it cannot be settled until it is reviewed by you (or one of your staff) and either accepted or rejected. For more information on reviewing transactions, refer to the Secure ePayments Card Processing Virtual Terminal Guide.

Sample code

The following examples demonstrate the use of the provided libraries within custom code. They are not designed for use within a production environment.

Java Code Sample

The Java package contains the com.clearcommerce.CpiTools.security.HashGenerator class, which contains one public method, generateHash(). This method generates a base64-encoded hash from the specified vector and key.

The params argument is a vector of strings. The key argument is the CPI Hash Key sent by letter to the Merchant. The syntax of this method is as follows:

public static String generateHash(Vector params, String key)

The following example code illustrates the use of the HashGenerator class.

```
import java.lang.*;
import java.util.*;
import
com.clearcommerce.CpiTools.security.HashGenerator;

class HashTest
{
    public static void main( String argv[] ) throws Exception
    {
                if ( argv.length < 2 )
                {
                        System.err.println(
                        "Usage: HashTest encryptedKey hashElement..." );
                        return;
                }
                String encryptedKey = argv[ 0 ];
                Vector hashElements = new Vector( );
                for ( int i=1; i<argv.length; i++ )
                {
                        hashElements.addElement( argv[ i ] );
                }
/* hash generation call */
                String hashValue = HashGenerator.generateHash(
                hashElements, encryptedKey );
/* end call */

                System.out.print( "Hash value: " );
                System.out.println( hashValue );
    }
}
```

C Code Sample

The CcCpiTools.h file contains two functions. The GenerateHash function creates a base64-encoded hash of the specified parameters using a specified key. The function returns a pointer to the hash if successful, or NULL if the case of a failure. Its syntax is as follows:

GenerateHash(char **params, const char *key);

The params argument is a NULL-terminated array of C strings. The key argument is the CPI Hash Key sent by letter to the Merchant. The DestroyHash function deletes a hash created by GenerateHash function. Its syntax is as follows:

DestroyHash(char *hash);

The following sample code illustrates the use of the OrderHash functionality.

```
/* TestHash.c */
#include <CcCpiTools.h>
#include <stdio.h>

int main( int argc, char **argv )
{
        char *strEncryptedKey;
        char **ppHashElements;
        char *strHashValue;
        int rc = 0;
        if ( argc < 3 )
        {
                fprintf( stderr, "Usage: TestHash encryptedKey
                hashElement...\n" );
                return 1;
        }
        strEncryptedKey = argv[ 1 ];
        ppHashElements = &argv[ 2 ];

        strHashValue = GenerateHash( ppHashElements, strEncryptedKey );

        if ( !strHashValue )
        {
                fprintf( stderr, "Error generating hash!\n" );
                rc = 2;
        }
        else
        {
                fprintf( stdout, "Hash value: %s\n", strHashValue );
        }

        DestroyHash( strHashValue );
        return rc;
}
```

COM Object example

The COM object contains one public method, GenerateOrderHash(). This method generates a base64-encoded hash from the specified array and key.

The params argument is a vector of strings. The key argument is the CPI Hash Key sent by letter to the Merchant. The syntax of this method is as follows:

public String GenerateOrderHash(String params(), String key)

The params argument is an array of strings.

The following example code illustrates the use of the GenerateOrderHash method.

```
<%
        Option Explicit
%>
<HTML>
<HEAD>
<TITLE>Sample OrderHash Generator ASP Page</TITLE>
<SCRIPT>
function singleSubmit(trgForm)
{
        trgForm.submitButton.disabled = true;
}
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
        <H1>Sample OrderHash</H1>
<%
        On Error Resume Next
        dim err, errMsg, strFieldTable, hash, ss
        ss=" zEeWQNKelqPE2DRFueuDq1QrASjux2lM"          'Dummy value

        'Build table entries and array of form items
        dim list, item
        dim delimiter
        dim valueArray, wrapper
        delimiter = "^"                    'Rarely used character to delimit strings
        for each item in Request.form
                if len( Request.form( item ) ) > 0 then
                        strFieldTable = strFieldTable & "<TR><TD>" & item &
":</TD><TD><INPUT type=""text"" name=""" & item & """ value=""" & Request.form( item ) &
"""></TD></TR>" & vbcrlf
                        list = list & Request.form( item ) & delimiter
                end if
```

```
        next
        list = left( list, len(list) - 1 )'Remove extraneous delimiter
        valueArray = split( list, delimiter )'Quick way to convert a set of strings to an array

        Set wrapper = CreateObject("CcCpiCOM.OrderHash")
        if err <> 0  then
                errMsg = "<H3>ERROR: Failed to create the CcCpiCOM.OrderHash
object.</H3><BR><input type=""submit"" name=""backButton"" value=""Back""
onClick=""window.history.back(1);"">"
        else
                hash = wrapper.GenerateOrderHash( valueArray, ss )
                if err <> 0 then
                        errMsg = "<H3>ERROR: Failed to obtain an order hash
value.</H3><BR><input type=""submit"" name=""backButton"" value=""Back""
onClick=""window.history.back(1);"">"
                end if
        end if
        Set wrapper = Nothing
        if len( errMsg ) > 0  then
                Response.write errMsg
        else
%>
        <p>This page creates an OrderHash of submitted items.<BR>
        <BR>
        <TABLE>
        <TR><TD>OrderHash:</TD><TD><INPUT type="text" name="OrderHash"
value="<% Response.write hash %>"</TD></TR>
        <% Response.write strFieldTable %>
        </TABLE>
<%
        end if
%>
</CENTER>
</BODY>
</HTML>
```

OrderHash examples

In the following examples the shared secret is set to the dummy value from the COM object example (zEeWQNKelqPE2DRFueuDq1QrASjux2lM). Quotes are used to surround an item which contains whitespace for visual reference only– they are not part of the data item.

Data	**OrderHash**
aaa	m4uGmXow+lcJYycNrbM7M8xS3No=
aaa bbb	zaM6DPT7GkEJtg2OWYzSePUa0rc=
aaa bbb “ccc ddd”	1sMZc6TaxcMnQthq2Dp0QKe7obk=
aaa bbb “ccc ddd” “eee fff ggg”	6FLZlCBhHTj5Tr9XbgFhuZVsh7M=

The “testhash” program (created by compiling the Java or C example above) can be used to recreate the above values.

The program runs from the command line and has the following syntax:

testhash <shared_secret> <data1> <data2> <data3> …

Data that contains whitespace must be enclosed in suitable quotes to maintain data integrity.

Supported encryption technologies

The CPI currently supports secure POSTs via SSL3.0 and TLS1.0 technologies (SSL2.0 is not supported).

Only a limited set of cipher suites are enabled for the two security protocols, these are outlined below:

TLS1.0

RC4 encryption with a 128-bit key and an MD5 MAC

TLS_RSA_WITH_RC4_128_MD5

FIPS 140-1 compliant triple DES encryption and a SHA-1 MAC

TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA

Triple DES encryption with a 168-bit key and a SHA-1 MAC

TLS_RSA_WITH_3DES_EDE_CBC_SHA

SSL3.0

RC4 encryption with a 128-bit key and an MD5 MAC

SSL_RSA_WITH_RC4_128_MD5

FIPS 140-1 compliant triple DES encryption and a SHA-1 MAC

SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA

Triple DES encryption with a 168-bit key and a SHA-1 MAC

SSL_RSA_WITH_3DES_EDE_CBC_SHA

Issued by HSBC Bank plc. We are a principal member of the HSBC Group, one of the world's largest banking and financial services organisations with over 9,700 offices in 77 countries and territories.

HSBC Bank plc
Commercial Service and Sales Centre
51 De Montfort Street
Leicester
LE1 7BB

www.ukbusiness.hsbc.com