

# **Kagi Remote Post System**

## **External Development and Deployment Specification**

**How to develop and deploy a  
CGI that can receive near real-time  
order data from Kagi**

## Copyright Information

March 2004

Copyright © 2004 Kagi. All rights reserved.  
1442A Walnut Street PMB #392, Berkeley, CA 94709, USA.

Information in this document is subject to change without notice. For the newest version, refer to <<http://www.kagi.com/acg>>

No part of this specification may be changed or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise without the prior written permission of Kagi.

**Table 1 Document History**

| Date           | Who  | What                                     |
|----------------|--|--|
| April 5, 2004  | <a href="mailto:ty@kagi.com">ty@kagi.com</a> | Revised based on comments                |
| March 15, 2004 | <a href="mailto:ty@kagi.com">ty@kagi.com</a> | Initial release of version 2 of the KRPS |

## Disclaimer

Permission is granted to use this specification and code for development and deployment of a Kagi Remote Post CGI free of charge as long as Kagi sells your products.

No warranty is made as to the suitability of the specification or the code samples herein or even that specification or code samples are bug or error free.

BECAUSE THE SPECIFICATION AND CODE SAMPLES ARE LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE SPECIFICATION OR CODE SAMPLES, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE SPECIFICATION AND CODE SAMPLES "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. YOU ASSUME THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SPECIFICATION AND CODE SAMPLES. SHOULD THE SPECIFICATION OR CODE SAMPLES PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

**TABLE OF CONTENT**

**KAGI REMOTE POST SYSTEM..... 1**

**EXTERNAL DEVELOPMENT AND DEPLOYMENT SPECIFICATION..... 1**

**HOW TO DEVELOP AND DEPLOY A ..... 1**

**CGI THAT CAN RECEIVE NEAR REAL-TIME ..... 1**

**ORDER DATA FROM KAGI..... 1**

**COPYRIGHT INFORMATION.....I**

**DISCLAIMER.....I**

**MOSTLY BLANK .....III**

**KAGI REMOTE POST SYSTEM..... 1**

    ABSTRACT..... 1

    OVERVIEW ..... 1

    BEFORE WE BEGIN..... 1

*24x7 Needs*..... 1

*Retries* ..... 1

*Security* ..... 2

*Nuances of remote calls from Kagi* ..... 2

*Unit or line quantity*..... 3

*Duplicate Posts* ..... 3

*Test orders*..... 3

    EXTERNAL CGI PROGRAMS..... 4

*Development* ..... 4

*Returning Username and Registration Keys*..... 4

*CSV Safe Definition*..... 4

*Return Definition:* ..... 5

*Sample output*..... 6

*Debugging* ..... 6

*Help and Enhancements* ..... 7

    SUBMISSION ..... 7

*Example* ..... 8

**APPENDIX A     SAMPLE PERL SCRIPT FOR KRPS CGI..... 1**

**APPENDIX B     HTML TEST PAGE FOR KRPS CGI..... 1**

**APPENDIX C     INPUT PARAMETERS..... 1**

**INTRODUCTION..... 1**

**INPUT PARAMETERS ..... 1**

    WHAT IS A ACG:CUSTOMERSEED? ..... 5

    WHAT IS A ACG:OEMSEED?..... 6

    -8BIT ..... 6

    ACG:UNITPAYMENT ..... 7

    SOMETHING WAS MISSING ..... 7

    HOW TO CALCULATE THE ENCRYPTED PASSWORD ..... 7

*Formula Description and Pseudo Code* ..... 7

*Formula Example implementation* ..... 8

*Formula Results* ..... 9

    ACG:FLAGS ..... 10

*Test Flag* ..... 10

*EncodingPreserved* ..... 10

## **Mostly Blank**

# Kagi Remote Post System

## **Abstract**

This document covers the development and setup of a typical form processing Common Gateway Interface (CGI) <<http://www.w3.org/CGI/>>, hosted on an external web site, which can receive order data during the final stage of order processing by Kagi. This notification is initiated by the Kagi Remote Post System (KRPS), which is part of the final stage of order processing at Kagi. The CGI that receives the notification can initiate supplier business tasks such as: inserting the customer data into a database, calculating activation codes, or other business process that are desired. Optionally, it can return specific data to Kagi that can be placed in the Thanks For Your Purchase (TFYP) email message(s).

## **Overview**

Kagi has the ability to call a form processing CGI (written in Perl script, ASP, Java, C, or most other modern computer programming languages), which is hosted externally from Kagi during the generation of the TFYP message or as part of a KRM purchase process (Figure 1). The CGI can optionally return specific key-value pairs to Kagi whose values can be placed in the TFYP message(s) using replacement tags in a blurb or passed to the KRM client program. These data are typically the username and activation codes for the product.

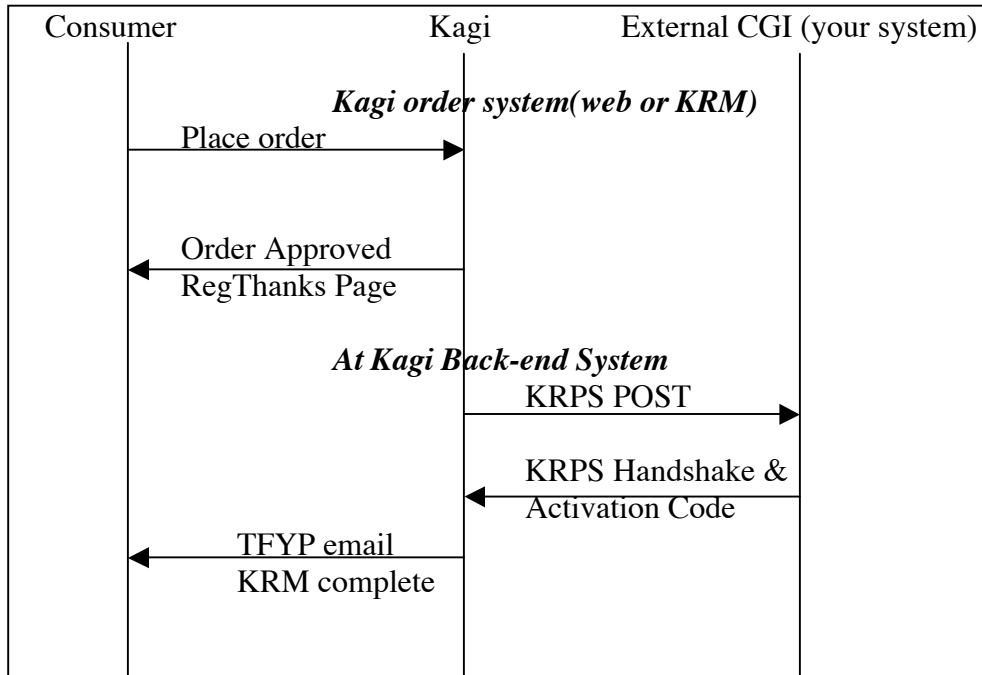
The KRPS was built using Kagi's Activation Code Generation (ACG) system. Kagi constructed an ACG that allows suppliers to specify a URL, username, password(shared secrete), and other information so the KRPS ACG can execute a CGI POST to the external CGI host system. A simple Perl CGI script is included (Appendix A) to help you get going and understand the flow of the process.

The KRPS supports both HTTP and HTTPS connection protocols and Basic Authentication for realms using a username and password. In addition, the KRPS supports a distinct CGI authentication system using a username and password (shared secrete) login, where the password can be sent in clear text or using a standard secure digest algorithm.

The external CGI is called when the Kagi back-end systems create the TFYP email or when any activation code needs to be generated for the customer, for instance, when KRM is completing the purchase request. By the time the external CGI is called the payment instrument, e.g. credit card, has been authorized for the funds and all anti-fraud tests have been complete.

**Figure 1**

Swim lane diagram of typical consumer interaction with Kagi and Kagi's interaction with KRPS external CGI. Customer contacts Kagi using <http://order.kagi.com> or via a client program that has a KRM client library. During the processing Kagi calls the external CGI on the supplier systems. This diagram is simplified, as it does not show any error handling nor recovery.



### ***Before we begin***

#### **24x7 Needs**

Customers order products from Kagi around the clock, every day of the year. Given this need, the hosting provider, for the external CGI, should have a "non-stop" environment to minimize any problems. This means that the hosting provider should have redundant power systems, redundant connections to the Internet and redundant web servers. If the hosting provider does not have these abilities, then you should think twice about using an externally hosted CGI and instead host your code generator at Kagi via the ACG system.

Any failure to contact the external CGI will stall completion of the order by delaying the generation and delivery of TFYP email message to the customer and the supplier.

#### **Retries**

If the external CGI fails, or cannot be contacted, the KRPS will attempt one or more retries before giving up, and then trying again after some delay. During this retry cycling, the customer's order is stalled at Kagi. This means that neither the customer nor

the product supplier(s) will receive the TFYP message. After a number of attempts, the KRPS will give up and send an email message to the supplier of the offending CGI, but no notification will be sent to the customer telling them the order is stalled. This message will indicate that there is a problem, and ask that some action be taken to determine what the problem is and to fix it. The email that is generated in this error state will NOT have any details about the error. The system that generates the error email cannot access the error information at this time. A future version of the system will have an error message with details to help determine the specific nature of the problem.

## Security

The KRPS does not encrypt the order data before sending them to the external CGI. If peer-to-peer encryption is desired, specify HTTPS as the protocol on the KRPS connection URL. The hosting server must have a valid SSL certificate or the POST call will fail.

To authenticate that Kagi is the one initiating the CGI invocation, the CGI supplier can also specify a username and password (shared secrete) to Kagi during the setup process, which is passed to the external CGI during the POST call. The CGI can then obtain these data and check to make sure they match, then taking appropriate actions. The KRPS can send the password (shared secrete) in clear-text or using one of these standard secure digest functions: MD5, Adler32, MD2, MD4, SHA, SHA-1, SHA-256, SHA-384, SHA-512, HMAC with MD5 and HMAC with SHA-1.

The hosting site may also utilize Basic Realm Authentication protection based on realm username and password. These username and password data are distinct from the CGI username and password (shared secrete).

**Warning:** The external CGI should never do any IP address checking. Kagi will be calling the external CGI from different locations around the globe. Checking IP addresses will very likely cause the POST to **fail**, so do not check the IP address!

## Nuances of remote calls from Kagi

In order to construct the KRPS external CGI correctly there are a few things that must be considered and planned for in the implementation. The external CGI supplier must determine if the CGI should be called once (per order Line) or possibly many times (once per Unit) for any given order; the CGI must be able to handle duplicate POSTS, and it must handle test orders correctly.

## Unit or line quantity

The Kagi back-end calling system can be setup to call the KRPS ACG (and therefore the KRPS external CGI) using a value of one ('1') in the ACG:QuantityOrdered parameter regardless of the number of copies ordered (InvocationType=Unit), or it can be called with the true purchase quantity (InvocationType=Line). This decision has direct bearing on how many times the external CGI will be invoked. When the request for hookup is submitted, the calling style of the external CGI must be specified. The default calling convention is: Line (true quantity). In InvocationType=Unit mode, the external CGI will be invoked N times, where N is the true quantity of the purchase.

## Duplicate Posts

It is uncommon, but it does happen, that the external CGI will receive duplicate invocations for the same product order. This happens sometimes when the customer indicates that the activation code that was delivered did not work, or when some other unspecified error occurs. The external CGI must handle duplicate invocations and return either new data or the same data it did the first time it was invoked. It should be pointed out that if the InvocationType is equal to Unit and the consumer purchases more than one copy, the CGI would receive what looks like duplicate orders. The only way this author can think of to tell the difference is the elapsed time between invocations and that is not a good way. It is best to configure your CGI to accept InvocationType=Line and return multiple activation codes in the one invocation, thus avoiding false duplicate POST calls.

## Test orders

Since test orders usually invoke some special business logic, the ACG system will mark (and thus the external CGI will receive) test orders using the flag (Test=1) passed via the ACG:Flags parameter. This parameter is discussed fully in Appendix C.

The external CGI must invoke any special business processes needed for test orders. On the Kagi side, we send the TFYP email to the supplier email account, (your @kagi.com address) only, not to the email address specified by parameter ACG:PurchaserEmail. KRM orders always receive a mangled activation code. So, if the external CGI sends email to the customer with activation codes, the external CGI should send the email to a special in-house account, and not to the email indicated in the ACG:PurchaserEmail parameter. Since anyone can place a test order at Kagi, this special case that must be addressed.

## **External CGI programs**

### Development

Creating a form processing CGI is easy in most development environments. Use your favorite IDE and programming language to create a normal CGI that will receive a POST invocation with a number of fields. The exact names of the fields that will be passed to the external CGI and their definitions are explained in Appendix C. If the reader is not already familiar with the parameters that a Kagi ACG can reference, please familiarize yourself now by viewing Appendix C now. Some basic understanding of those parameters will be needed for the details below.

As mentioned in the Overview section, the external CGI can be invoked with a CGI specific password (shared secrete) that is passed in either clear-text or in an encrypted form. If the external CGI is going to use an encrypted password, the formula for calculating the encrypted version is defined in Appendix C.

### Returning Username and Registration Keys

The external CGI **must** return at least one line of data to Kagi, the status line. This data line is required as it is the handshake between Kagi and the external CGI. Without the completed handshake, the KRPS will repeatedly try to invoke the CGI. After the handshake line, the external CGI can, optionally, return values for the <username> and <regnumber> replacement tags in the connected blurb. All values are returned using a simple key=value pair assignment where the value must be Comma Separated Value (CSV) safe.

A note about when to send the handshake line: The CGI should send the handshake line after all internal processing is complete and not anytime sooner. If the handshake is sent before, say insert to a database is complete and the insert fails, Kagi will assume the CGI has the data and no other attempt to contact the external CGI will be made.

Below is Kagi's definition for CSV safe.

### CSV Safe Definition

- 1). Allowable ASCII characters within a CSV field include 0x09 (tab), 0x0A (linefeed), 0x0D (new line) and the inclusive range of 0x20 (space) through 0x7E (tilde).
- 2). A CSV string **may** be surrounded by double-quotes.
- 3). A CSV string **must** be surrounded by double-quotes to contain a comma.
- 4). A CSV string **must** be surrounded by double-quotes to contain an embedded double-quote, represented by a pair of consecutive double-quotes.
- 5). A CSV string **may** contain 0x0A (line feed [LF], \n), 0x0D (new line [CR], \r) or 0x0D,0x0A (carriage return [CR] | \r, line feed [LF] | \n), but

may not contain consecutive CRLF, i.e. 0x0D,0x0A,0x0D,0x0A as this delimits the KRPS line.

### Return Definition:

As stated above, all return values are key = value pairs. The keys *kagiRemotePostStatus*, *message*, *userName*, *regNumber*, and *quantity* have special meanings. The CGI supplier maybe define other keys, but they will be ignored at this time.

The syntax definition is: (Line feeds where added for clarity only)

```
KagiRemotePostStatus = GOOD | BAD
    [, message = text_you_provide]0x0D,0x0A,0x0D,0x0A
[[username = nameValue]
    [, regNumber = regValue]
    [, quantity = value]
    [,otherKeys = otherValues],...]CRLF
[[username = nameValue]
    [, regNumber = regValue]
    [, quantity = value]
    [,otherKeys = otherValues],...]r\nr\n]
```

The [] means that the data is optional, but once optional data is specified all previous optional data must be specified; the '|' character means either, but both of these values cannot be used.

Every external CGI that the KRPS invokes must return at least one line with the key: *kagiRemotePostStatus* with a value of *GOOD* or *BAD*. If the line and key are not returned, the KRPS will assume the remote CGI failed and will generate an internal error in the sales cycle. The key *message* is optional for the handshake, but strongly suggested for any return status.

The *kagiRemotePostStatus* of *GOOD* should be used when the remote CGI has completed all its works, and can safely exit. If there is any type of error, the value *BAD* should be returned and the CGI should log the error locally, send email the supplier with all the parameters and diagnostic information it can muster, so the error can be fixed as this return will trigger the KRPS to enter the retry cycle; described above. Again, it can not be stressed enough that the handshake line should be the last thing the CGI sends, so it has every opportunity to process the data and return a failure if need be; once the KRPS has received *kagiRemotePostStatus=GOOD* message, it will assume the processing is complete and send the TFYP email or complete the KRM purchase.

The key *userName* will be substituted for the TFYP blurb tag <username>, while the *regNumber* key will replace the TFYP blurb tag <regnumber>. These keys allow the external CGI to send back activation codes that will be forwarded to the consumer. The key *quantity* has no direct TFYP replace yet, but is designed as a pass-through for the ACG system. Other return keys that are specified will not show up in the TFYP message. An impending rewrite of the TFYP engine will allow access to other keys.

It is recommended that the remote CGI always return some information for the *userName* and *regNumber*, as this can aid in debugging the CGI if problems do arise. For example, if the remote CGI is to insert the purchase data into a database, the *userName* might be used to hold the primaryID for the table, or some other useful data. Remember the user does not always have to see this output; the <username> replacement tag can be removed before the blurb enters production.

### Sample output

Here are samples of HTML, in the form, that must be returned for a successful and failure POST invocations from the external CGI. The string CRLF denotes the ASCII character string 0x0D0x0A and is required by both the HTML CGI specification and the KRPS external CGI return specification. (Any line wraps are a result of the word processor and are not included in the real HTML.)

Example HTML of a failure return. The error is denoted in the message.

```
Content-type: text/text CRLFCRLF
kagiRemotePostStatus=BAD, message=Could not get CGI
started, disk full. CRLFCRLF
```

Example of a successful return, excluding *userName* and *regNumber* keys.

```
Content-type: text/text CRLFCRLF
kagiRemotePostStatus=GOOD, message=Transaction inserted.
CRLFCRLF
```

Example of a successful return, including *userName* and *regNumber* keys.

```
Content-type: text/text CRLFCRLF
kagiRemotePostStatus=GOOD, message=Transaction inserted.
CRLFCRLF
userName=acg@kagi.com, regNumber=myCodeHereCRLFCRLF
userName=acg@kagi.com, regNumber=myOtherCodeHereCRLFCRLF
```

### Debugging

Appendix B contains an HTML page that can be used to simulate POST calls to the external CGI during development and before releasing it. To start testing, change the action line to specify where the external CGI resides on the development system and any specific values needed for the ACG:OEMSeed and ACG:CustomerSeed parameters. When the external CGI is ready for use by Kagi, submit an email asking for a hookup, detailed in the next section.

## Help and Enhancements

If you need help developing the CGI or have questions about how something should be designed for the remote CGI, please send email to [acg@kagi.com](mailto:acg@kagi.com).

If you have suggestions, for how this document can be clarified or suggestions for enhancements to the KRPS please send them to <acg@kagi.com>.

### **Submission**

Once the external CGI is written and debugged, place it on any HTTP server that can be accessed from **any** Internet address. Make sure a browser can connect to it , do at least one test POST to it using the supplied testing page, Appendix B.

Then create a blurb or blurbs using the Blurb Admin section of supplier application at Kagi <<http://supplier.kagi.com>>. Use the <username> and <regnumber> replacement markers to indicate where the values for the keys *userName* and *regNumber* are to be placed in the blurb, if they are returned. Then send an email to Kagi with the hookup data, detailed below. For detailed instructions on how to set up blurbs, please read Formatting Blurbs at <<http://www.kagi.com/acg/formatblurbs.html>>.

Send an email to [acg@kagi.com](mailto:acg@kagi.com) with the following data:

Subject: KRPS install please

Body:

Please hook this up so it works via the KRPS facility.

**Table 2 KRPS hookup parameters for email request.**

| Specifier Name | Optionality | Description   |
|----------------|-------------|---|
| URL            | required    | The full form processing URL beginning with http:// or https://   |
| RealmUser      | optional    | The user name for basic realm authentication. It is best that the RealmUser use only ASCII characters.                |
| RealmPassword  | optional    | The password for basic realm authentication. It is best that the RealmPassword use only ASCII characters.             |
| UserName       | optional    | The user name for login to the CGI. It is best that the UserName use only ASCII characters.                           |
| PassPhrase     | optional    | The password or pass phrase or shared secret for CGI login. It is best that the PassPhrase use only ASCII characters. |
| Blurbname      | required    | The name or list of name of the blurb(s) just created.  |
| SupplierName   | required    | Your @kagi.com address  |
| SupplierCode   | required    | The 3-6 letter Kagi code  |

|                      |          |  |
|----------------------|----------|--|
| Cypher               | required | The cipher digest algorithm the external CGI is using. It maybe any of the following: Clear-Text, MD5, Adler32, MD2, MD4, SHA, SHA-1, SHA-256, SHA-384, SHA-512, HMAC_MD5 and HMAC_SHA-1. All these digest algorithms are standards. They have both known and unknown weaknesses. Choose the one that best meets the needs and that are be supported in the CGI environment. |
| InvocationType       | required | How the KRPS will be called, once for the order (Line), or once for every copy sold (Unit). Default is Line.   |
| ExpectUserAndRegcode | required | CGI returns the username and registration code, or just the status message. Values are Yes or No.  |

Once the hookup request is processed, a confirming email will be sent from Kagi, telling the requester that the install is complete. At that point, place a test order for the product(s) that have the blurb(s) named above connected to it. If there are multiple products for which KRPS notification desired, it is recommended that a different blurb be attached for each product. Please test one product at a time, that means, purchase only one of the products per order.

### Example

To: [acg@kagi.com](mailto:acg@kagi.com)  
From: [supplier@domain.com](mailto:supplier@domain.com)  
Subject: KRPS install please  
Body:

Please hook this up so it works via the KRPS facility.

```

URL          -- http://www.domain.com/cgi-bin/KRPS_myThingy.cgi
Username     -- Harvey
PassPhrase   -- Easter bunny and friend
Blurbname    -- supplierName_KRPS_Egg
SupplierName -- me__kagi
SupplierCode -- XXA
RealmUser    -- not used
RealmPassword -- not used
Cypher:      -- MD5
InvocationType -- Unit
ExpectUserAndRegcode -- Yes

```

## Appendix A      Sample Perl Script for KRPS CGI

```
#!/usr/bin/perl
#
#      This Perl CGI is a sample KRPS external CGI
#
#
#      Copyright © 2004 Kagi. All rights reserved.
#      1442A Walnut Street PMB #392, Berkeley, CA 94709, USA.
#
#
#      Permission is granted to use this specification and code for development and
#      deployment of a Kagi Remote Post CGI free of charge as long as Kagi sells
#      your products.
#
#      No warranty is made as to the suitability of code herein
#      or even this code is bug or error free.
#
#      BECAUSE THE CODE IS LICENSED FREE OF CHARGE, THERE IS NO
#      WARRANTY FOR THE CODE, TO THE EXTENT PERMITTED BY
#      APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING
#      THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDING THE
#      CODE "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER
#      EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE
#      IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
#      PARTICULAR PURPOSE. YOU ASSUME THE ENTIRE RISK AS TO THE
#      QUALITY AND PERFORMANCE OF THE CODE. SHOULD THE CODE
#      PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL-NECESSARY
#      SERVICING, REPAIR OR CORRECTION.

package KagiRemoteCGIPostReceiver;
use strict;
use Exporter;
use English;
use CGI;
use Digest::MD5;

        # end of line and return for CGI and KRPS
my ($cr,$lf) = ("\015", "\012");
my ($CrLf) = ("{$cr$lf}");

&main;
1;
exit;
sub main(){
        #
```

```

        # CGI programs must print their own HTTP
        # response headers
        #
printf ("Content-type: text/text%s%s", $crlf, $crlf);

my $cgi = new CGI;          #read cgi input parameters

        # does it start correctly?
if( !defined($cgi) ){
    printf( "kagiRemotePostStatus=BAD,
            message=Could not get CGI started.%s%s", $crlf, $crlf );
    return(0);
}

        # get the message and the digest for comparison
my $sharedSecret = "myTestPassword",
my $digest = md5Password($sharedSecret, $cgi);
if( !defined($digest) ){
    # status set in call, just return
    return(0);
}

        # Specifically get the password from the input parameters
my $passwd = $cgi->param('ACG:Password');

        #does the password match what I think it should be
if( $digest ne $passwd ){
    printf( "kagiRemotePostStatus=BAD,
            message>Password did not match calculated version. %s%s",
            $crlf, $crlf );
    return(0);
}

        # OK we passed the password test, now do something useful
my ($userName, $regNumber) = myalgo($cgi->param('ACG:PurchaserEmail));
printf( "kagiRemotePostStatus=GOOD,
        message=Authenticated call from Kagi.%s%s", $crlf, $crlf );
printf( "username=%s, regNumber=%s%s%s".
        $userName, $regNumber, $crlf, $crlf);

return(1);
}

```

```

sub md5Password($$){
    # Calculate the KRPS encrypted password message and return lower case version
    #
    # The result of this routine should match what is passed by Kagi if the MD5
    # encryption hash was specified during setup.
    #     Reference Appendix C for details on the implementation
    #
    my($sharedSecrete, $cgi) = @ARG;

        # build the password as MD5 Hash
    my $md5 = Digest::MD5->new();
    if( !defined($md5) ){
        printf("kagiRemotePostStatus=BAD,
            message=MD5 could not be initialized. %s%s",
            $CrLf,$CrLf)
    }

        # lowercase the data to be consistent, no spaces
        # this is step 1 and step 2 in Appendix C
    $md5->add( lc( $ sharedSecrete
        . $cgi->param("ACG:TransactionID")
        . $cgi->param("ACG:ProductName")
        . $cgi->param("ACG:UnitPayment")
        . $cgi->param("ACG:DateProcessed")
        . $cgi->param("ACG:QuantityOrdered")
        . $cgi->param("ACG:LicenseType")
    )
    );
        #we lowercase the password, this is step 3 and step 4 in Appendix C
    my $newHash = lc( $md5->hexdigest );

        # Perl adds spaces to output only, other libraries might add more
    my ($cleanHash,@stringArray,$character) = ("",undef,undef);

        # this is more of step 4 of Appendix C, Perl only, yours will vary.
    @stringArray = split(//,$newHash);

        # this is more of step 4 of Appendix C, Perl only, yours will vary.
    foreach $character (@stringArray){
        # only accept hex characters
        if( $character =~ /[a-f0-9]/ ){
            $cleanHash.= $character;
        }
    }
    return( $cleanHash);
}
1;

```

## Appendix B HTML Test Page for KRPS CGI

```
<HTML>
<!-- -->
<!-- -->
<!-- Copyright © 2004 Kagi. All rights reserved.
      1442A Walnut Street PMB #392, Berkeley, CA 94709, USA.

      Permission is granted to use this specification and code for development and
      deployment of a Kagi Remote Post CGI free of charge as long as Kagi sells
      your products.

      No warranty is made as to the suitability of code herein
      or even that the code is bug or error free.

      BECAUSE THE CODE IS LICENSED FREE OF CHARGE, THERE IS NO
      WARRANTY FOR THE CODE, TO THE EXTENT PERMITTED BY
      APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING
      THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE
      CODE "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER
      EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE
      IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
      PARTICULAR PURPOSE. YOU ASSUME THE ENTIRE RISK AS TO THE
      QUALITY AND PERFORMANCE OF THE CODE. SHOULD THE CODE
      PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL-NECESSARY
      SERVICING, REPAIR OR CORRECTION.

-->
<HEAD>
  <TITLE>KRPS Test Page</TITLE>
</HEAD>
<BODY>
<P><FONT SIZE="+2" FACE="Arial">ACG test page</FONT></P>
<!-- -->
<!-- MODIFY THE ACTION BELOW -->
<!-- FOR YOUR SERVER ADDRESS AND CGI-BIN DIRECTORY -->
<!-- -->
<!-- -->

<P><!-- *****-->
<!-- * Modify the line below to point to the correct ACG or External CGI to test *-->
<!-- *****--></P>

<P><FORM ACTION="http://127.0.0.1/cgi-bin/ KagiRemoteCGIPostReceiver.pl"
METHOD=POST>
<P><!-- *****-->
<!-- *****-->
```

<!\_\*\*\*\*\*-->

</P>

<P><TABLE>

<TR>

<TD>

<P></P>

</TD>

<TD>

<P>The seed parameters are comma separated. Internal commas come escaped with a comma. The maximum length of the value for a given seed is 256 bytes</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:Flags</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:Flags" VALUE="Test=1" SIZE=55><BR>

</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:UserName</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:UserName" VALUE="enter value if needed" SIZE=55><BR>

</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:Password</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:Password" VALUE="myTestPassword" SIZE=55><BR>

</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:CustomerSeed</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:CustomerSeed"  
VALUE="Keyword: mykey - value, hardwareFingerprint=E96D-6CEE,  
userPreferredName=Softedge,, Inc." SIZE=255><BR>

</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:OEMSeed</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:OEMSeed"  
VALUE="Clear-Text=Password" SIZE=256><BR>

</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:DebugFlag</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:DebugFlag"  
VALUE="1" SIZE=30> Value 0 or 1

</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:InputVersion</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:InputVersion"  
VALUE="0200" SIZE=30> Value 0200<BR>

</P>

</TD>

</TR>

```
<TR>
  <TD>
    <P>ACG:Request</P>
  </TD>
  <TD>
    <P><SELECT NAME="ACG:Request">
      <OPTION VALUE=Generate>Generate
      <OPTION VALUE=Information>Information
      <OPTION>NULL
    </SELECT>Value is 'Generate' and 'Information'. See
    specification for meanings</P>
  </TD>
</TR>
```

```
<TR>
  <TD>
    <P>ACG:CardName</P>
  </TD>
  <TD>
    <P><INPUT TYPE=text NAME="ACG:CardName"
      VALUE="Card Name Here" SIZE=30><BR>
    </P>
  </TD>
</TR>
```

```
<TR>
  <TD>
    <P>ACG:CardName-8bit</P>
  </TD>
  <TD>
    <P><INPUT TYPE=text NAME="ACG:CardName-8bit"
      VALUE="CARD NAME HERE" SIZE=30><BR>
    </P>
  </TD>
</TR>
```

```
<TR>
  <TD>
    <P>ACG:PurchaserName</P>
  </TD>
  <TD>
    <P><INPUT TYPE=text NAME="ACG:PurchaserName"
      VALUE="Fred Fragola" SIZE=30><BR>
    </P>
  </TD>
```

</TR>

<TR>

<TD>

<P>ACG:PurchaserName-8bit</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:PurchaserName-8bit"  
VALUE="Fred Fragola" SIZE=30><BR>

</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:PurchaserEmail</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:PurchaserEmail"  
VALUE="email@domain.com" SIZE=30><BR>

</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:SQNM</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:SQNM"  
VALUE="9999" SIZE=30>Value  
0000 to 9999<BR>

</P>

</TD>

</TR>

<TR>

<TD>

<P>ACG:QuantityOrdered</P>

</TD>

<TD>

<P><INPUT TYPE=text NAME="ACG:QuantityOrdered"  
VALUE="1" SIZE=30><BR>

</P>

</TD>

</TR>

```
<TR>
  <TD>
    <P>ACG:LicenseType</P>
  </TD>
  <TD>
    <P><SELECT NAME="ACG:LicenseType">
      <OPTION VALUE=single>single
      <OPTION VALUE=site>site
      <OPTION VALUE=world>world
      <OPTION VALUE=upgrade>upgrade
      <OPTION VALUE=bonus>bonus
      <!-- -->
      <!-- Edit the value below for you special license type-->
      <!-- -->
      <OPTION VALUE="special=">special=(you must edit form and supply value)
      <OPTION>null
    </SELECT>See specification for meanings</P>
  </TD>
</TR>
```

```
<TR>
  <TD>
    <P>ACG:UnitPayment</P>
  </TD>
  <TD>
    <P><INPUT TYPE=text NAME="ACG:UnitPayment"
      VALUE="149.000" SIZE=30><BR>
    </P>
  </TD>
</TR>
```

```
<TR>
  <TD>
    <P>ACG:TransactionID</P>
  </TD>
  <TD>
    <P><INPUT TYPE=text NAME="ACG:TransactionID"
      VALUE="CH0123456789" SIZE=30><BR>
    </P>
  </TD>
</TR>
```

```
<TR>
  <TD>
    <P>ACG:Postal-8bit</P>
```

```
</TD>
<TD>
  <P><INPUT TYPE=text NAME="ACG:Postal-8bit"
    VALUE="POSTAL-8BIT 12134 Center St. Boston CA 87654"
    SIZE=60><BR>
  </P>
</TD>
</TR>

<TR>
<TD>
  <P>ACG:Postal</P>
</TD>
<TD>
  <P><INPUT TYPE=text NAME="ACG:Postal"
    VALUE="Postal 12134 Center St. Boston CA 87654" SIZE=60><BR>
  </P>
</TD>
</TR>

<TR>
<TD>
  <P>ACG:ProductName-8bit</P>
</TD>
<TD>
  <P><INPUT TYPE=text NAME="ACG:ProductName-8bit"
    VALUE="PRODUCT_NAME" SIZE=30><BR>
  </P>
</TD>
</TR>

<TR>
<TD>
  <P>ACG:ProductName</P>
</TD>
<TD>
  <P><INPUT TYPE=text NAME="ACG:ProductName"
    VALUE="Product_Name" SIZE=30><BR>
  </P>
</TD>
</TR>

<TR>
<TD>
  <P>ACG:UserPurchaseDate</P>
</TD>
```

```
<TD>
  <P><INPUT TYPE=text NAME="ACG:UserPurchaseDate"
    VALUE="2000-12-31" SIZE=30><BR>
  </P>
</TD>
</TR>

<TR>
<TD>
  <P>ACG:DateProcessed</P>
</TD>
<TD>
  <P><INPUT TYPE=text NAME="ACG:DateProcessed"
    VALUE="2000-12-31" SIZE=30><BR>
  </P>
</TD>
</TR>

<TR>
<TD>
  <P>ACG:TimeStamp</P>
</TD>
<TD>
  <P><INPUT TYPE=text NAME="ACG:TimeStamp"
    VALUE="49388234823" SIZE=30><BR>
  </P>
</TD>
</TR>

<TR>
<TD>
  <P>Submit</P>
</TD>
<TD>
  <P><INPUT TYPE=submit NAME=Submit
    VALUE="Do your ACG test"><BR>
  </P>
</TD>
</TR></FORM>
</TABLE>

</FORM></P>
</BODY>
</HTML>
```

## Appendix C

## Input Parameters

**Table 3 Document History**

| Date       | Who  | Input Version | Change   |
|------------|--|---------------|--|
| 2004-05-05 | <a href="mailto:ty@kagi.com">ty@kagi.com</a> | 0202          | Revised document based on feedback   |
| 2004-03-17 | ty@kagi.com                                  | 0202          | Added Note field, to make data consistent with that which is passed via the TFYP email |
| 2004-03-10 | ty@kagi.com                                  | 0201          | Added UserName, Password, and Flags parameters   |
| 2004-03-01 | ty@kagi.com                                  | 0200          | Pulled from ACG Spec   |

### Introduction

This Appendix is written to those writing an ACG and to those writing an external KRPS CGI. Since the KRPS is an ACG that simply forwards all the parameters to the external CGI, all the notes, comments and data here in apply to both. To accommodate both audiences the word Program is used for both an ACG and the remote external CGI. The footnotes supply extra information, please read those.

### Input parameters

The input parameters are supplied to the Program via an HTTP POST. Each parameter listed in Table 4 will have a value that is URL encoded; i.e. spaces will be changed to '+', special characters encoded as their hex values (%hex) and parameters separated by '&'. For example, the date and name form values of 10/02/99 and Fred Flintstone would be encoded as:

...date=10%2F02%2F99&name=Fred+Flintstone...

It is not necessary to develop code to unravel the parameters; there are free libraries available in Perl, C, Java, Visual Basic and other languages to do the decoding. Parameters that are not required by the Program should be ignored. There is no specific defined order in which the input parameters will arrive into the Program.

**Table 4 Input values to ACG**

| Datum Name               | Null Allowed | Max Length | Notes   |
|--------------------------|--------------|------------|---|
| ACG:Request <sup>1</sup> | No           | 32         | Which action the ACG is being requested to execute. Values are 'Generate' or 'Information'. |

<sup>1</sup> May be ignored if the Program is an external CGI for the KRPS.

| Datum Name                              | Null Allowed | Max Length | Notes  |
|---|--------------|------------|--|
| ACG:InputVersion <sup>1</sup>           | No           | 4          | ACG input stream version number. Format is ####. ACG should check this version string to make sure it can accept the request. The current version is 0201. The Program should accept any value greater than 0200 but below 0300. |
| ACG:PurchaserName <sup>2 3 4</sup>      | Yes          | 256        | Name as typed into Register field by end-user; and converted to the 7-bit value.   |
| ACG:PurchaserName-8bit <sup>2 3 4</sup> | Yes          | 256        | Name as typed into Register field by user in full 8-bit character format.  |
| ACG:PurchaserEmail <sup>2 3 4</sup>     | Yes          | 256        | E-mail address as end-user typed it. If more than one email is supplied they will be passed to the ACG in a comma-separated list with no spaces.   |
| ACG:Postal <sup>3 4</sup>               | Yes          | 256        | Postal address as user typed it; and converted to the 7-bit value.   |
| ACG:Postal-8bit <sup>3 4</sup>          | Yes          | 256        | Postal address as user typed it in full 8 bit character format.  |
| ACG:TimeStamp <sup>5</sup>              | No           | 32         | Kagi timestamp in seconds.   |
| ACG:UserPurchaseDate <sup>5</sup>       | No           | 10         | Date purchased YYYY-MM-DD  |
| ACG:DateProcessed                       | No           | 10         | Date processed by Kagi YYYY-MM-DD  |

<sup>2</sup> One or more of these 5 input parameters will contain data. The ACG/CGI developer must determine which parameter to use if a “name” is needed as a seed value for the activation code.

<sup>3</sup> Supplied by end-user, do not depend on data value, format, or existence of datum.

<sup>4</sup> The end-user can supply more data than allowed according to length. In such a case, the data will be truncated to the specified length.

<sup>5</sup> This date is generated based on data supplied by the end-user machine and is not to be trusted. Kagi routinely gets date information from the early part of the 19<sup>th</sup> century.

| <b>Datum Name</b>                  | <b>Null Allowed</b> | <b>Max Length</b> | <b>Notes</b>   |
|------------------------------------|---------------------|-------------------|--|
| ACG:CardName <sup>2 3 4</sup>      | Yes                 | 100               | User name as it appears on credit card, if credit card was used in purchase; and converted to the 7-bit value.   |
| ACG:CardName-8bit <sup>2 3 4</sup> | Yes                 | 100               | User name as it appears on credit card, if credit card was used in purchase in full 8-bit character format.  |
| ACG:CustomerSeed <sup>3 4</sup>    | Yes                 | 256               | Data item(s) the end-user enters before purchase is submitted; i.e. a serial number or web-download number.  |
| ACG:OEMSeed <sup>4</sup>           | Yes                 | 256               | Seed data as specified by product table in Kagi database. This allows the OEM to use the same ACG with many products by varying some part of the algorithm by the value specified. |
| ACG:TransactionID                  | No                  | 32                | Kagi transaction ID as calculated by internal systems; Unique for each invoice transaction and varies in length from 12 to 32 characters.  |
| ACG:SQNM                           | No                  | 4                 | The sequence number for the blurb sent out with the product. There may not be a 1-1 correlation between SQNM and sales of a specific product. Range 0000-9999.                     |
| ACG:ProductName <sup>4</sup>       | No                  | 256               | Product name as specified in database; and converted to the 7-bit value.   |
| ACG:ProductName-8bit <sup>4</sup>  | No                  | 256               | Product name as specified in database in full 8-bit character format.  |

| <b>Datum Name</b>                | <b>Null Allowed</b> | <b>Max Length</b> | <b>Notes</b>   |
|----------------------------------|---------------------|-------------------|--|
| ACG:QuantityOrdered <sup>3</sup> | No                  | 3                 | Quantity of activation keys the ACG should generate during invocation. Value is between 1 and 250.   |
| ACG:LicenseType <sup>34</sup>    | No                  | 256               | The type of product license being purchased. Current values: 'single', 'site', 'bonus', 'world', 'upgrade' 'special=?'. Where the value of ?? is something the end-user supplied. In addition to the ACG:CustomerSeed, this has meaning inside the ACG. For example; ?? could be 'education' for education license generation. |
| ACG:UnitPayment <sup>3</sup>     | No                  | 11                | The amount, normalized to U.S. Dollars the end-user paid for each copy of the product. The value is rounded to 3 decimals <sup>6</sup> ; i.e. \$0.000. Values range from 0.000 to 9999999.999.   |
| ACG>Note                         | Yes                 | 1024              | A free form text field that contains both data from Kagi and data from the customer. The note is a concatenation of data, the TFYP email my display in distinct lines, but here they are just one long string.   |

---

<sup>6</sup> Rounding to three decimal places was chosen as a way to provide some extended data with currencies that have more decimal units (Euro) when converted to U.S. Dollars, that traditionally only have two decimals.

| Datum Name                | Null Allowed | Max Length | Notes   |
|---------------------------|--------------|------------|---|
| ACG:DebugFlag             | No           | 1          | 0 or 1. If 1, then debug mode, produce debug output.  |
| ACG:UserName <sup>7</sup> | Yes          | 64         | During an invocation to a host machine external to Kagi this will contain the value that was specified as the ACG Username. Otherwise, the value is null.   |
| ACG>Password <sup>7</sup> | Yes          | 256        | During an invocation to a host machine external to Kagi this will contain the value that was specified as the ACG Password. It will be encrypted or sent in clear text. Otherwise, the value is null. |
| ACG:Flags                 | Yes          | 256        | A set of key=value pairs, where the value is 0 or 1. Any key that is missing is assumed to have a value of 0.   |

### ***What is a ACG:CustomerSeed?***

The ACG:CustomerSeed parameter contains any customer unique data that the consumer was requested to enter on the registration form; whether using <http://order.kagi.com>, a piece of software equipped with the KRM system, or a register program. For example, the supplier might collect the 3Com Palm HotSyncID, or the shirt size, or the shirt color and size in two different fields on the register form. Any time an ACG:CustomerSeed is collected it is a required field on the client; but it should still check to make sure it received the seeds it required as there have been issues with some browser ignoring the requirement because of poor adherence to the http specification.

ACG:CustomerSeed data are passed to the Program in a comma-separated list, with the commas escaped with a second comma if a comma occurred within the datum entry.

Each keyword collected is placed in the following format string: Keyword: KEY – VALUE; where KEY is the unique name of the entry and the VALUE is the datum that was typed by the consumer. This format is from a legacy system. A pending rewrite will make them key=value fields, but you code does not have to deal with that, only the current format.

---

<sup>7</sup> Maybe ignored by Kagi Hosted ACG, externally hosted ACG will want to check this value.

For example, if <http://order.kagi.com> collected fields named "Keyword: title -" and "Keyword: SyncID -" with title = 'Dr. Mark Sloan, M.D' and SyncID = [acg@kagi.com](mailto:acg@kagi.com) then the data would be passed in ACG:CustomerSeed as "Keyword: title - Dr. Mark Sloan., M.D, Keyword: SyncID - [acg@kagi.com](mailto:acg@kagi.com)". Keyword fields are required entry fields on order.kagi.com so the data should always be there, if not the Program should fail. Therefore, the Program must check and deal with double commas in the parameter value to process the input correctly.

If the Program is using the ACG:CustomerSeed data, please write to [acg@kagi.com](mailto:acg@kagi.com). There is always some trick to doing this type of data usage and we want to help you succeed with as little work as possible.

### ***What is a ACG:OEMSeed?***

The ACG:OEMSeed is a parameter that is specified by the supplier at setup time. The Program can use the value(s) in calculating a registration code or for other business logic decisions that needs to be predetermined but unique for each setup. The parameter is a static string up to 255 bytes long. It can contain anything the supplier wants, including parameterized values; most suppliers pass key=value pairs that are used to initialize an activation key algorithm. The string is passed in the Program unaltered by the calling system. If the value of the key contains a comma, it will be escaped using the same mechanism described in the ACG:CustomerSeed section, a double comma replacement. Therefore, the Program must check and deal with double commas in the parameter value to process the input correctly.

### ***-8bit***

Some input parameters have the suffix of -8bit. This indicates that the byte values of the parameter may fall between 0x00 to 0xFF; the full byte range. A corollary to this is: parameters without the -8bit will fall within a subset of the ASCII range: 0x09, 0x0A, 0x0D, and 0x20 to 0x7F.

Given a choice, the Program should use the -8bit parameter when it is valid. To use the -8bit parameters the Program needs to be eight-bit-clean; which should not be a problem for most Program types. By using the -8bit parameter(s), the Program will be able to handle multi-byte characters (Chinese, Japanese,...) in calculations; which is a desirable behavior for the Program. The Program must also check the flag value *EncodingPreserved*, to make sure it can safely use and return -8bit data. Doing otherwise may generate errors when the consumer enters the code that is based on -8bit data.

Conversion between -8bit and subset of ASCII defined above is indeterminate. For example, the character 'œ' maybe converted to 'o'; 'e'; 'oe'; '#', or '?' depending on the ability of the conversion engine. The data at Kagi is stored in UTF-8, but the original charset is not always known; which leads to some conversion errors when the data is

stored or used later. At some future date, the Program user will be able to specify the encoding charset the –8bit data will be converted to before the Program is invoked.

If the ACG is not 8-bit-clean, i.e. it cannot accept all character input values in the range of 0x80 to 0xFF, then it should use regular ‘ASCII’ input parameters for all calculations.

### ***ACG:UnitPayment***

ACG:UnitPayment is calculated by Kagi as the total amount for a product line divided by ACG:Quantity for that product line. The total amount for the product line is not available in the Program, but can be calculated as  $ACG:Quantity * ACG:UnitPayment$ .

### ***Something was missing***

In Table 4, the column titled “Null Allowed” indicates if the ACG input value will always be supplied or is optional. If it is marked “Yes”, that indicates that the datum is derived from, the user input or data no associated directly with a customer purchase but something the supplier or Kagi has defined. If the ACG is going to use a “Null Allowed” parameter, it needs to have some reasonable way to assign a default value and continue processing when the user, Kagi, or the supplier does not supply it.

### ***How to calculate the encrypted password***

Any external Program, to Kagi, can be invoked with a Program specific password(shared secrete) that is passed in either clear-text or an encrypted form. If the external Program is going to use an encrypted form, the formula for calculating the encrypted version is given below. The remainder of this subsection can be skipped if the external Program is going to use the clear-text password. No matter how the shared secrete is passed, the Program must check for a match to authenticate that the invocation originated from a Kagi based server.

In each of the sub sections below (description & pseudo code, example, results) there are 4 step defined. Each section builds on the previous one, and the step markers are consistent to help understand how the formula is defined and implemented.

### **Formula Description and Pseudo Code**

The password (shared secrete) supplied on setup is concatenated to create a string (message, in secure algorithm speak) using the following formula:

(Step 1) message = Shared\_secrete\_given\_by\_you + ACG:TransactionID  
+ ACG:ProductName + ACG:UnitPayment + ACG:DateProcessed  
+ ACG:QuantityOrdered + ACG:LicenseType

**Note:** the ‘+’ symbol is used to denote a string concatenation, do not add the ‘+’ character into the message that will be digested by the specified function.

Once the message is constructed, the message is converted to lowercase. This is done to make the message consistent on both ends.

```
(Step 2) message = lowercase(message)
```

Then the message is sent to the digest function specified during the hookup request:

```
(Step 3) digest = digest_function(message);
```

The digest result is then passed through a routine that will lowercase the output and remove ALL NON-HEX DIGITS, where a hex-digit must be one of the following [0-9,a-f].

```
(Step 4) cleanDigest = hexOnly(digest)
```

Then and only then, can it be compared to what was sent in the parameter list to the Program.

```
if (ACG:Password not equal to cleanDigest )  
    then { problem, exit with error. Log and alert supplier }  
    else { do something useful }
```

### Formula Example implementation

Using the data presented, the pseudo code above, are outputs from each step defined. Use these outputs to help you test your implementation.

Data:

|                             |                   |
|-----------------------------|-------------------|
| Shared_secrete_given_by_you | => MyTestPassword |
| ACG:TransactionID           | => CH0123456789   |
| ACG:ProductName             | => Product_Name   |
| ACG:UnitPayment             | => 149.000        |
| ACG:DateProcessed           | => 2000-12-31     |
| ACG:QuantityOrdered         | => 1              |
| ACG:LicenseType             | => single         |

**Note:** Line breaks below are for visual appearance only. While the '+' symbol is used to denote a string concatenation, do not add the '+' character nor line break into the message that will be digested by the specified function.

Pseudo code:

```
(Step 1 begin) message = Password_specified_during_hookup  
    + ACG:TransactionID  
    + ACG:ProductName  
    + ACG:UnitPayment  
    + ACG:DateProcessed  
    + ACG:QuantityOrdered
```

+ ACG:LicenseType  
(Step 2) message = lowercase(message)  
(Step 3) digest = digestFunction(message)  
(Step 4) cleanDigest = hexOnly(digest)

## Formula Results

(Step 1 results) – message = MyTestPasswordCH0123456789  
Product\_Name149.0002000-12-311single

(Step 2 results) – message = mytestpasswordch0123456789product\_name  
149.0002000-12-311single

(Step 3 & 4 results using an MD5 algorithm)  
cleanDigest = f04f35da7c517f9190c5d955102c9cea

(Step 3 & 4 results using an Adler32 algorithm)  
cleanDigest = a50e13e5

(Step 3 & 4 results using an HMAC\_MD5 algorithm)  
cleanDigest = 01444d44e25b621c6761b1b88c228065

(Step 3 & 4 results using an HMAC\_SHA-1 algorithm)  
cleanDigest = b61dd33d24bedf3e00a00e6ee04fd18af2eed21a

(Step 3 & 4 results using an SHA-1 algorithm)  
cleanDigest = a3a845ab0d5abe089e9503b2a41dc402

(Step 3 & 4 results using an MD4 algorithm)  
cleanDigest = 5dda17c12ff589b9b11559452d2cb7e3

(Step 3 & 4 results using an SHA-1 algorithm)  
cleanDigest = a9b5c46cd2ad5b00249a4e8501a49f1127619595

(Step 3 & 4 results using an SHA-256 algorithm)  
cleanDigest = ad06ae402ee20088270bbb399964c1aa  
bdc1eaa8509ea2d0bcb794226ae85a96

(Step 3 & 4 results using an SHA-384 algorithm)

```
cleanDigest = 27fbc0b33660d70bdc2229327a1cfac7
              ac163aa01ffda1affa39a1d4bcd01a85
              21b9ecb1755997f32d4c5e008a52343b
```

(Step 3 & 4 results using an SHA-512 algorithm)

```
cleanDigest = ada2ddb81a43355c1878a0b6afef8b44
              18b241cc93c3042481b8ed3aa4d896bd
              1c75cbeff256daff93bf8525d465ca99
              dc028f10246677be66cc39c949b616af
```

### **ACG:Flags**

The ACG:Flags parameter is used by Kagi to pass specific Boolean flags with values of “0” or “1”, to the Program that might be of interest. The value of the parameter is a comma separated list of key=value flag pairs. If a specific key is missing, the value must be assumed to have value of zero (“0”), or FALSE; Table 5 defines the keys and their meaning.

**Table 5**      **ACG:Flag keys**

| <b>Flag Name</b>  | <b>Definition</b>  |
|-------------------|--|
| Test              | Defined with value of “1” when the purchase is a test order.                               |
| EncodingPreserved | Defined with value of “1” when the client is capable of sending and receiving 8-bit values |

#### **Test Flag**

Since test orders usually invoke some special business logic, the ACG system will mark (and thus the Program will receive) all test orders with the flag (Test=1) that is passed via the ACG:Flags parameter.

The Program must invoke any special business processes needed for test orders. On the Kagi side, we send the TFYP email to the supplier email account, (your @kagi.com address) only, not to the email address specified by parameter ACG:PurchaserEmail. KRM orders always receive a mangled activation code. So, if the Program sends email to the customer with activation codes, the external CGI should send the email to a special in-house account, and not to the email indicated in the ACG:PurchaserEmail parameter. Since anyone can place a test order at Kagi, this special case that must be addressed.

#### **EncodingPreserved**

Set to a value of “1” when the client, and all intermediate systems, will preserve charset encoding. Currently, only the Kagi Registration Module (KRM) is capable of this feature. TFYP emails will not preserve charset encodings, as neither the sending mail client nor the receiving mail client will guarantee correct charset conversion and encoding. The Program should use this flag to determine if the 8-bit data would be safe to use and return an 8-bit based activation key. For example if UTF-8 was used as the 8-

bit charset, then the data must be passed back to the client (your program) using UTF-8 charset. Most email clients do not honor the charset definition of emails and will change the values as needed; they convert to system local if possible or replace unconvertible characters to '?' or '\*'. Thus, the activation key displayed to the user will not contain the correct values to activate the software. The keys may look alike, but they are not encoded the same. For Programs that are going to use the -8bit interface, they must check for the presents of this flag before using the data. For maximum compatibility the -8bit interface should be avoided.

## **Mostly Blank**