yubico
the key to the cloud

Yubikey Client COM API

YubiKey device client-side interface component

Version: 1.1

May 28, 2012

Introduction

Yubico is the leading provider of simple, open online identity protection. The company's flagship product, the YubiKey®, uniquely combines driverless USB hardware with open source software. More than a million users in 100 countries rely on YubiKey strong two-factor authentication for securing access to computers, mobile devices, networks and online services. Customers range from individual Internet users to e-governments and Fortune 500 companies. Founded in 2007, Yubico is privately held with offices in California, Sweden and UK.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design, and manufacturing. Yubico shall have no liability for any error or damages of any kind resulting from the use of this document.

The Yubico Software referenced in this document is licensed to you under the terms and conditions accompanying the software or as otherwise agreed between you or the company that you are representing.

Trademarks

Yubico and YubiKey are trademarks of Yubico Inc.

Contact Information

Yubico Inc
228 Hamilton Avenue, 3rd Floor
Palo Alto, CA 94301
USA
info@yubico.com

Contents

1   Document Information

1.1   Purpose

The purpose of the client interface component is to allow easy integration of Yubikey configuration functionality into client-side applications accessing the Yubikey challenge-response and serial number functionality introduced in Yubikey 2.2.

The component is not intended as a "stand-alone" utility kit and the provided sample code is provided as boilerplate code only.

1.2   Audience

Programmers and systems integrators.

1.3   Related documentation

- *The YubiKey Manual* – Usage, configuration and introduction of basic concepts
- *YubiKey Configuration Utility* – The Configuration Tool for the YubiKey
- *Yubikey Configuration API* – Yubikey configuration COM API
- *Yubikey Server API* – Server-side support COM API
- Yubico online forum – http://forum.yubico.com

1.4   Document History

Date	Version	Author	Activity
2007-06-09	1.0	JE	First draft
2012-05-14	1.1	ZD	Changed document template

1.5   Definitions

Table Header 1	Table Header 2
YubiKey device	Yubico's authentication device for connection to the USB port
USB	Universal Serial Bus
HID	Human Interface Description. A specification of typical USB devices used for human interaction, such as keyboards, mice, joysticks etc.
API	Application Programmer Interface
COM	Component Object Model – a component based programming model developed by Microsoft.
ActiveX	A definition on top of COM, primarily targeted for user interface extensions in a Web-browser.
Callback	Function in the User Program called by the API
AES	Advanced Encryption Standard. A NIST approved symmetric encryption algorithm.
OATH-HOTP	Initiative for Open Authentication (RFC 4226)
HMAC	Hash-based Message Authentication Code
SHA-1	Secure Hash Algorithm 1

2 Introduction

Starting with Yubikey firmware version 2.2, support has been added for programmatic challenge-response operations and serial number retrieval. Contrary to the standard Yubikey functionality, this requires support of an interface exchanging data programmatically with the Yubikey hardware in the USB port. Communication is provided by the means of HID Feature Reports so no low-level installable (WDM) drivers are required where the built-in HID class driver is used.

The challenge-response mode is enabled at configuration time and is set on a by-configuration basis. A configuration enabled for challenge-response mode cannot be used for normal OTP mode generation.

To simplify application development, Yubico provides a high-level device configuration component based on Microsoft's COM/ActiveX technology. With this approach, a wide range of programming languages, scripting environments and software packages can perform device configuration operations using a single unified interface.

This document assumes knowledge about the Yubico YubiKey, its functions and intended usage as well as basic challenge-response concepts.

The document further assumes working knowledge of COM, and at least one programming language that supports COM components. Provided examples are developed with Microsoft Visual Studio .NET 2008.

The component is designed for the Microsoft Windows Win32 environment and works with Windows versions from Windows 2000 and onwards. Integration with Microsoft's .NET programming model is straightforward. Refer to appropriate .NET documentation of integration of COM components.

2.1 Programming model

By using COM/ActiveX, most programming languages and third-party tools can interface to the Yubikey via the YubiClientAPI Component through a uniform interface with standard data representation. In other words, the component can be used by any programming language and development tool supporting COM/ActiveX. Examples include Visual C++, Visual Basic, Delphi, Microsoft Office (VBA) and Internet Explorer VB Script.

A COM programming tutorial is beyond the scope of this document, but application samples are provided for VBA (Excel), Visual C++/MFC and HTML/Internet Explorer.

3 Programming concepts

Contrary to the basic Yubikey usage where data is sent as keystrokes, the challenge-response model requires two-way communication supported with an interface component.
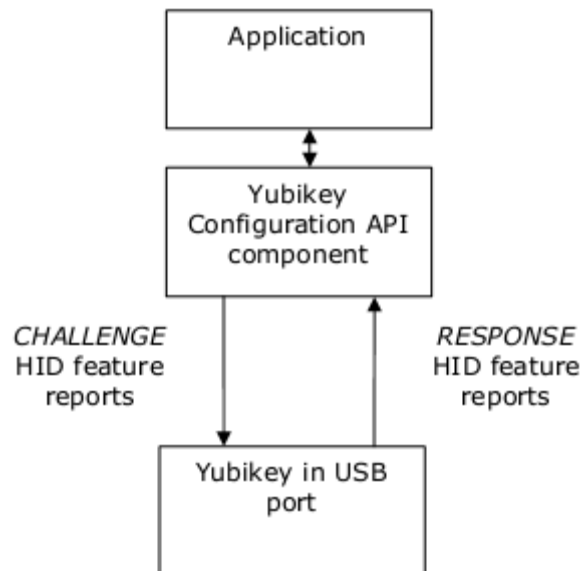
The purpose of the Yubikey Client API is to encapsulate the complexities of data exchange with the Yubikey hardware and to provide an easy to use interface that allows simple integration with any COM enabled application.

3.1 Yubikey Client API features

The Yubikey Client API implements the following Yubikey 2.2 features:

- **Yubikey OTP challenge-response**
  This method works like ordinary Yubikey OTP generation algorithm with the difference that a 6-byte challenge is XORed with the private ID prior to the Yubikey OTP calculation. The 128-bit "ticket" is then sent back as a 16-byte response.

- **HMAC-SHA1 challenge-response**
  This method allows a challenge of up to 64 bytes (512 bits) to be hashed using the HMAC-SHA1 algorithm with a 20-byte (160 bits) secret.

- **Factory-programmed device serial number read**
  The non-alterable factory programmable device serial number can be read. The function is always enabled for a blank Yubikey and must be explicitly enabled when a configuration is written.

- **Asynchronous device insert- and removal notifications**
  Asynchronous notifications when a Yubikey is inserted or removed can be enabled.

3.2 Synchronous vs. asynchronous model

Challenge-response operations have a non-predictable response time as the device may be configured to require user interaction (the user presses the Yubikey button) before the operation is completed and the response is sent back. In order not to block the main application, an asynchronous model is implemented.
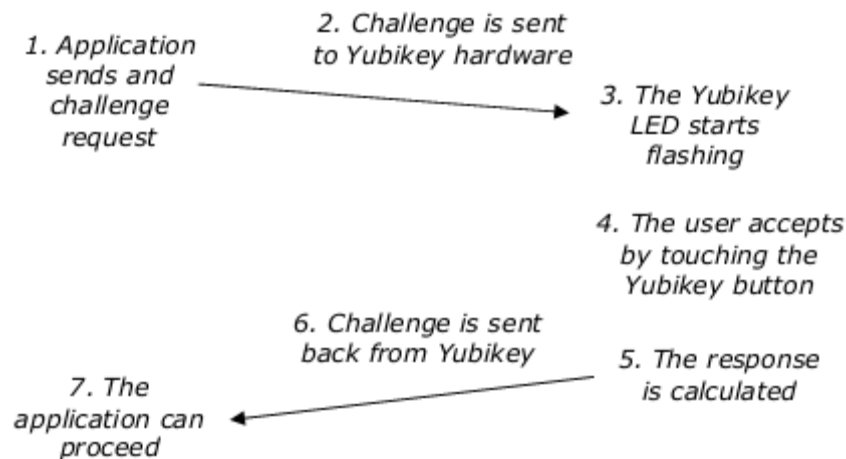
Note that the user touching the Yubikey button is a configurable option. If this option is not enabled, the challenge will be sent back directly. However, as there is some latency involved in sending out the challenge and getting the response back via HID feature reports, there is a 30-250 ms delay depending on the operation executed and the amount of data included in the transaction.

AN asynchronous operation is initiated by a call that returns immediately. The operation is then handled by a background thread and when completed, an asynchronous completion event is fired by the background thread together with a completion code. The application can then check the return code and read the response data if applicable.

In certain settings, implementing asynchronous calls increases the complexity. Therefore two synchronous call mechanisms are implemented:

- Blocking – The main thread is suspended until the response is received.
- Blocking-with-yield – The main thread is suspended until the response is received, but the message queue is polled and incoming Windows messages are processed.

4   Using the API

The YubiKey configuration API is provided as a COM/ActiveX component, where methods and properties are exposed. Asynchronous notifications are provided by the means of events.

The component follows the data types defined by the COM Automation model which provides maximum flexibility and interoperability.

A COM component needs to be registered with the operating system in order to be used. This is typically done by an installation tool, where the Self registration function is used. The component can be explicitly registered with the regsvr32 utility: Type **regsvr32 YubiClientAPI.dll** under the Start/Run menu.

Integration of COM components varies between different tools and languages, but the following steps describe the typical workflow of using the API of the YubiClientAPI Component:

1. **Provide a reference to the component**
    The development tool needs access to the YubiClientAPI component's Type Library, which contains the interface description. The Type Library is embedded in the component itself; there is no separate .tlb file.
2. **Instantiate the component**
    The instantiation phase gives a "handle" to the YubiClientAPI component.
3. **Set up and implement a callback/event interface (if required)**
    If asynchronous notifications for when a device has been inserted or removed is needed, set up and implement the "sink" interface.

The following examples uses a "pseudo-code" notation, omitting the COM object reference and its instantiation for the sake of clarity. Function prototypes are shown in VB notation such as

```
Property myFunction(myParameter As parameterType)As returnType
```

4.1 Data representation and data exchange

To support simple scripting languages and to allow maximum flexibility for each particular application, input- and output data is passed as COM/Automation VARIANTs. These VARIANTs can be configured to hold either strings (BSTRs), unsigned integers or byte arrays.

```
Property dataBuffer As VARIANT
```

A global property is provided to set the appropriate data encoding

```
Property dataEncoding As ycENCODING
```

The following data encoding formats are available for the VARIANTs used:

- Hexadecimal string – `ycENCODING_BSTR_HEX`
  Lower-case hexadecimal digits without spacing, e.g. `6b6c3132`

- Hexadecimal string with spaces – `ycENCODING_BSTR_HEX_SPACE`
  Lower-case hexadecimal digits with spacing, e.g. `6b 6c 31 32`

- Modhex string – `ycENCODING_BSTR_MODHEX`
  Yubico Modhex format, e.g. `hnhrebed`

- Base64 string – `ycENCODING_BSTR_BASE64`
  Base64 encoded string, e.g. `a2wxMg==`

- Ascii string – `ycENCODING_BSTR_ASCII`
  Ascii (MBCS / non-Unicode) encoded string, e.g. `k112`

- Unsigned 16-bit integer – `ycENCODING_UINT16`
  16-bit USHORT/UINT16 integer = 2 bytes

- Unsigned 32-bit integer – `ycENCODING_UINT32`
  32-bit ULONG/UINT32 integer = 4 bytes

- Byte array - – `ycENCODING_BYTE_ARRAY`
  SAFEARRAY of bytes (UINT8/UI1) with dynamic length

- NULL / Nothing
  Represents an empty string holding zero bytes

Input and output data is handled by the means of a global data buffer which can be set or read at any time. This model further allows data conversion "on the fly", such as the following pseudo-code

```
dataBuffer = 0x4711          -- Hexadecimal 4711

dataEncoding = ycENCODING_BSTR_MODHEX

print dataBuffer             -- prints fibb

dataEncoding = ycENCODING_BSTR_HEX_SPACE
```

```
print dataBuffer            -- prints 47 11

dataEncoding = ycENCODING_UINT32

print dataBuffer            -- prints 0x00004711
```

Integers (16- and 32 bits) are handled as Big Endian (high byte first), just as they appear in a byte string.

4.2 Synchronous vs. Asynchronous calls

Function calls can either be synchronous or asynchronous (see 3.2)

Synchronous calls are the simplest to use, but with the downside that the calling thread is blocked until the response has been received from the Yubikey.

ysCALL_MODE_BLOCKING – This mode suspends the calling thread without maintaining the Windows message pump. This typically causes the application to "hang" until the response has been received.

ysCALL_MODE_BLOCKING_YIELD – This mode suspends the calling thread but maintains the Windows message pump by polling the message queue. Depending on the application, this can avoid the "hang" behaviour.

In the case the Yubikey is configured to wait for the user to physically touch the Yubikey button to confirm the request, the main thread may be blocked up to 15 seconds before the request is completed or times out.

Asynchronous calls on the other hand add a bit of complexity but allow the response to be passed asynchronously, without suspending the calling thread. Polling for the response is done by a background thread, transparent from the calling application's point of view. When the response has been received, an asynchronous completion call is fired from the polling thread.

ysCALL_MODE_ASYNC – This mode selects asynchronous calls. As the completion call is fired from a different thread, some applications may experience thread- or synchronization problems. In such cases, consider adding a thread-safe wrapper or use synchronous calls.

A second alternative if asynchronous calls are undesirable is polled operation. The **dataBuffer** property is cleared by the asynchronous call and set when completed. The application program may periodically poll the **dataBuffer** property for a change to determine if the call has been completed.

Pending asynchronous calls can be terminated by calling the **abortPending** function.

4.3 Yubico OTP challenge-response

A Yubico OTP challenge-response creates a 16 byte (128 bit) response from a 6 byte (48 bit) challenge using the Yubico OTP algorithm. The secret private id (UID) of the Yubikey is XORed with the challenge and then used as UID in the calculation.

The benefit of using the Yubico OTP algorithm is that it uses device-generated data together with the challenge data.

As the Yubico OTP algorithm relies on a finite counter field (useCounter), it is less useful for very frequent challenge-response operations.

The function may optionally require user interaction by pressing the Yubikey button in order to be processed. This function is enabled with the `CFGFLAG_CHAL_BTN_TRIG` configuration flag.

Synopsis

Property otpChallenge(config As Integer, mode as ycCALL_MODE) As ycRETCODE

The basic principle (using blocking calls) is

```
dataBuffer = challenge – First 6 bytes used

returnCode = otpChallenge(0, ycCALL_MODE_BLOCKING)

if returnCode = ycRETCODE_OK Then

     print dataBuffer -- Response is here

Else

     print "An ycRETCODE_xx error occurred"

EndIf
```

Using asynchronous call, the pseudo-code would be like

```
dataBuffer = challenge -- First 6 bytes used

returnCode = otpChallenge(0, ycCALL_MODE_ASYNC)

if returnCode <> ycRETCODE_OK Then

     there-should-be-no-error-here handler

End If


Event_handler operationCompleted(ycRETCODE returnCode)

If returnCode = ycRETCODE_OK Then

     print dataBuffer -- Response is here

Else

     print "An ycRETCODE_xx error occurred"

EndIf
```

4.4 HMAC-SHA1 challenge-response

A HMAC-SHA1 challenge-response creates a 20 byte (160 bit) response from a 0-64 byte (0-512 bit) challenge. The secret stored in the Yubikey is fixed 20 bytes (160 bits).

The benefit of using the HMAC-SHA1 algorithm is that it follows the standard FIPS PUB 198 / RFC 2104 specification and can be used with server-side applications supporting this standard. Furthermore, a longer challenge can be used compared with the **otpChallenge** function and there is no finite counter fields used.

The function either operates on a fixed 64-byte or a variable 0-63 byte challenge depending on the **CFGFLAG_HMAC_LT64** configuration flag.

The function may optionally require user interaction by pressing the Yubikey button in order to be processed. This function is enabled with the **CFGFLAG_CHAL_BTN_TRIG** configuration flag.

Synopsis

Property hmacSha1(config As Integer, mode as ycCALL_MODE) As ycRETCODE

The basic principle (using blocking calls) is

```
dataBuffer = challenge -- First 6 bytes used

returnCode = hmacSha1(0, ycCALL_MODE_BLOCKING)

if returnCode = ycRETCODE_OK Then

     print dataBuffer -- Response is here

Else

     print "An ycRETCODE_xx error occurred"

EndIf
```

Using asynchronous call, the pseudo-code would be like

```
dataBuffer = challenge -- First 6 bytes used

returnCode = hmacSha1(0, ycCALL_MODE_ASYNC)

if returnCode <> ycRETCODE_OK Then

     there-should-be-no-error-here handler

End If


Event_handler  operationCompleted(ycRETCODE  returnCode)

If  returnCode  =  ycRETCODE_OK  Then

     print dataBuffer -- Response is here
```

```
Else

      print "An ycRETCODE_xx error occurred"

EndIf
```

4.5 Serial number read

The non-alterable device serial number can be read (unless disabled in the device configuration) via an API call.

The call completes in around 50 ms, so generally an asynchronous call is typically not needed.

Synopsis

`Property readSerial(ycCALL_MODE mode) As ycRETCODE rc`

The basic principle (using blocking calls) is

```
returnCode = readSerial(ycCALL_MODE_BLOCKING)

if returnCode = ycRETCODE_OK Then

      dataEncoding = ycENCODING_UINT32

      print dataBuffer -- Response is here as 32-bits integer

Else

      print "An ycRETCODE_xx error occurred"

EndIf


Using asynchronous call, the pseudo-code would be like

returnCode = readSerial(ycCALL_MODE_ASYNC)

if returnCode <> ycRETCODE_OK Then

      there-should-be-no-error-here handler

End If

Event_handler  operationCompleted(ycRETCODE  returnCode)

If returnCode = ycRETCODE_OK Then

      print dataBuffer -- Response is here

Else

      print "An ycRETCODE_xx error occurred"

EndIf
```

4.6 Aborting asynchronous calls

Pending asynchronous calls can be aborted by calling the **abortPending** method. The function then fires an **operationCompleted** event with return code **ycRETCODE_FAILED**.

Synopsis
Sub abortPending()

4.7 Device present state

An application can synchronously check if a device is present or not by reading the **isInserted** property.

Synopsis

Property isInserted As ycRETCODE

The property returns

ycRETCODE_OK	One device is present
ycRETCODE_MORE_THAN_ONE	More than one device is present
ycRETCODE_NO_DEVICE	No device is present

In settings where asynchronous notifications if insert- and removal events is desired, refer to section 4.8.

4.8 Enabling insert- and removal events

Asynchronous notifications can be enabled by setting the **enableNotifications** property. A device insert triggers the **deviceInserted** event and a device removal fires the **deviceRemoved** event.

Synopsis

Property enableNotifications As ycNOTIFICATION_MODE

The property can be set to the following values

|	
ycNOTIFICATION_OFF	No notifications are fired
ycNOTIFICATION_ON	Notifications are fired when devices are inserted and removed
ycNOTIFICATION_ON_DISCARD_FIRST	Notifications are fired when devices are inserted and removed but the initial update event is discarded

Note that the deviceInserted event is fired when one device is inserted so only one device is inserted. The deviceRemoved event is fired when a device is removed so no Yubikeys are present in the system.

5 Test container tutorial

The MFC test container can be used to test the Yubikey, the Yubikey Client API functionality and to understand the challenge-response concepts.

The MFC/VC++ source code is provided as a "boilerplate" template that can be used for test and/or further application development. Microsoft Visual Studio 2008 or later is required to build the test application. A reference to the YubiClientAPI.dll COM component is done at the #import statement in the MFCTestDlg.h file. Change the project search path to the target location if necessary.

Start the MFC test container executable
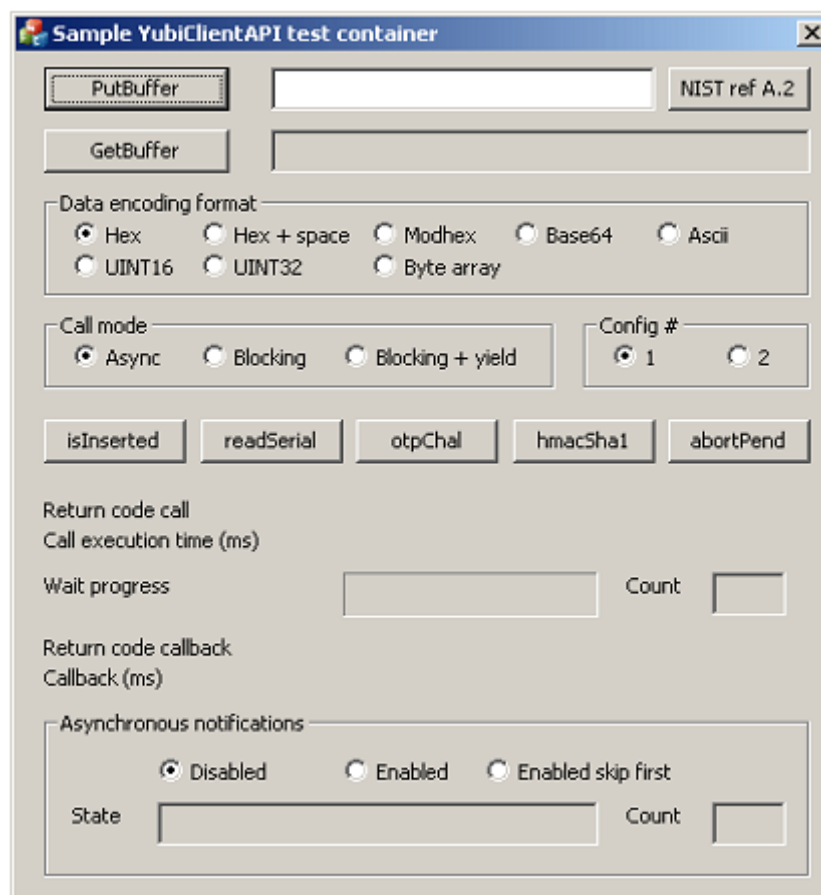
5.1 Data buffer and encoding

Enter a hexadecimal string in the PutBuffer text field. Push GetBuffer and the same string will appear in the GetBuffer read-only text field. The returned data type is appended by the application for debug purposes.

1. Click Ascii in the Data encoding format box
2. Enter an Ascii string such as ABC123 in the PutBuffer field and push PutBuffer
3. Click Hex in the Data encoding format box
4. Click GetBuffer and the hexadecimal representation 414243313233 appears together with the `VARIANT return type VT_BSTR`

This functionality is used to exchange data in the challenge-response transactions but can also be used for simple conversion between different string formats and data types.

Set data:

1. Set dataEncoding to the appropriate `ycENCODING_xxx` type
2. Set dataBuffer to the desired data

Get data:

1. Set dataEncoding to the appropriate `ycENCODING_xxx` type
2. Get the dataBuffer holding data in the selected format

5.2 Device insert- and removal detection

Synchronous mode

1. Insert a Yubikey and press isInserted
2. `ycRETCODE_OK` is displayed in the Return code call field
3. Remove the Yubikey and press isInserted
4. `ycRETCODE_NO_DEVICE` is displayed in the Return code call field

Asynchronous mode

1. Click the Enabled radio button in the Asynchronous notification box
2. The current inserted or removal state is displayed in the State field. Clicking Enabled skip first instead suppresses this first notification
3. Remove and insert a Yubikey and the state is updated accordingly

5.3 Serial number read

The serial number retrieval by API calls must be enabled or the Yubikey must be un-configured in order for the serial number to be read.

Synchronous mode

1. Click the Blocking radio button in the Call mode box
2. Click the UINT32 radio button in the Data encoding box
3. Click the readSerial button
4. The serial number appears in the GetBuffer field and the execution time in milliseconds is displayed in the Call execution time field.

Asynchronous mode

1. Click the Async radio button in the Call mode box
2. Click the readSerial button
3. The function returns directly (Call execution time is zero)
4. The asynchronous call is fired when the serial number is read. The return code and execution time is displayed in the Return code callback and Callback (ms) fields.

If the function fails (shown in the Return code call field), the Yubikey does not have the serial number API read function enabled or is not a Yubikey 2.2 or later firmware.

5.4 Yubico OTP challenge-response

Yubico OTP challenge-response can be performed on a configuration where the Yubico OTP challenge-response mode is enabled (corresponding configuration bits set).

Assume the second configuration is configured for Yubico OTP challenge-response, without user intervention (button press) being configured.

Synchronous mode

1. Click the Blocking radio button in the Call mode box
2. Click the 2 radio button in the Config # box
3. Enter a 0-6 byte challenge in the PutBuffer field and press PutBuffer
4. Press otpChal
5. The response appears in the GetBuffer field. The return code and execution time is displayed in the Return code callback and Callback (ms) fields.

Asynchronous mode (Assume user intervention configuration bit being set)

Click the Async radio button in the Call mode box

Press otpChal

1. The function returns directly (Call execution time is zero)
2. Note how the progress bar counts down and the trigger counter increments as the Yubikey waits for user interaction
3. Touch the Yubikey button
4. The response appears in the GetBuffer field. The return code and execution time is displayed in the Return code callback and Callback (ms) fields.

Setting the Data encoding to Modhex prior to pressing the otpChal will return an OTP in Modhex format, allowing the OTP tobe validated by legacy validation code. However, the challenge in such cases should be set to all zeroes to allow private ID matching.

Alternatively, the Yubico Server API can be used toverify the OTP.

If the function call fails, the configuration selected is not configured for Yubico OTP challenge-response or the Yubikey is nota Yubikey 2.2 or later firmware.

5.5 HMAC-SHA1 challenge-response

HMAC-SHA1 challenge-response can be performed on a configuration where the HMAC-SHA1 challenge-response mode is enabled (corresponding configuration bits set).

Assume the first configuration is configured for Yubico OTP challenge-response, without user intervention (button press) being configured. Further assume that the HMAC secret is set to the NIST PUB 198 A.2 test vector.

Synchronous mode

1. Click the Blocking radio button in the Call mode box
2. Click the 1 radio button in the Config # box
3. Click the NIST ref A.2 button to insert the NIST PUB198 A.2 challenge in the PutBuffer field
4. Press hmacSha1
5. The response appears in the GetBuffer field. The return code and execution time is displayed in the Return code callback and Callback (ms) fields.

Asynchronous mode (Assume user intervention configuration bit being set)

1. Click the Async radio button in the Call mode box
2. Click the NIST ref A.2 button to insert the NIST PUB198 A.2 challenge in the PutBuffer field
3. Press hmacSha1
4. The function returns directly (Call execution time is zero)
5. Note how the progress bar counts down and the trigger counter increments as the Yubikey waits for user interaction
6. Touch the Yubikey button
7. The response appears in the GetBuffer field. The return code and execution time is displayed in the Return code callback and Callback (ms) fields.

Compare the output with the NIST vector which is `0922d3405faa3d194f82a45830737d5cc6c75d24` . If there is a mismatch, the Yubikey secret is incorrectly set.

If desired, the Yubico Server API can be used to verify the HMAC-SHA1.

If the function call fails, the configuration selected is not configured for HMAC-SHA1 challenge-response or the Yubikey is not a Yubikey 2.2 or later firmware.