Liboath API Reference Manual

COLLABORATORS			
	TITLE : Liboath API Reference Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 7, 2013	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Liboath API Reference Manual

Liboath is a shared and static C library for handling OATH related technology such as HOTP.

1.1 oath

oath —

Synopsis

```
#define             OATHAPI
#define             OATH_HOTP_DYNAMIC_TRUNCATION
#define             OATH_HOTP_LENGTH                    (digits,
                                                         checksum)
#define             OATH_TOTP_DEFAULT_START_TIME
#define             OATH_TOTP_DEFAULT_TIME_STEP_SIZE
#define             OATH_VERSION
#define             OATH_VERSION_NUMBER
int                 oath_authenticate_usersfile         (const char *usersfile,
                                                         const char *username,
                                                         const char *otp,
                                                         size_t window,
                                                         const char *passwd,
                                                         time_t *last_otp);
int                 oath_base32_decode                  (const char *in,
                                                         size_t inlen,
                                                         char **out,
                                                         size_t *outlen);
int                 oath_base32_encode                  (const char *in,
                                                         size_t inlen,
                                                         char **out,
                                                         size_t *outlen);
void                oath_bin2hex                        (const char *binstr,
                                                         size_t binlen,
                                                         char *hexstr);
const char *        oath_check_version                  (const char *req_version);
int                 oath_done                           (void);
```

```
int                 oath_hex2bin                        (const char *hexstr,
                                                         char *binstr,
                                                         size_t *binlen);
int                 oath_hotp_generate                  (const char *secret,
                                                         size_t secret_length,
                                                         uint64_t moving_factor,
                                                         unsigned  digits,
                                                         bool add_checksum,
                                                         size_t truncation_offset,
                                                         char *output_otp);
int                 oath_hotp_validate                  (const char *secret,
                                                         size_t secret_length,
                                                         uint64_t start_moving_factor,
                                                         size_t window,
                                                         const char *otp);
int                 oath_hotp_validate_callback         (const char *secret,
                                                         size_t secret_length,
                                                         uint64_t start_moving_factor,
                                                         size_t window,
                                                         unsigned  digits,
                                                         oath_validate_strcmp_function str
                                                         void *strcmp_handle);
#define             oath_hotp_validate_strcmp_function
int                 oath_init                           (void);
enum                oath_rc;
const char *        oath_strerror                       (int err);
const char *        oath_strerror_name                  (int err);
int                 oath_totp_generate                  (const char *secret,
                                                         size_t secret_length,
                                                         time_t now,
                                                         unsigned  time_step_size,
                                                         time_t start_offset,
                                                         unsigned  digits,
                                                         char *output_otp);
int                 oath_totp_validate                  (const char *secret,
                                                         size_t secret_length,
                                                         time_t now,
                                                         unsigned  time_step_size,
                                                         time_t start_offset,
                                                         size_t window,
                                                         const char *otp);
int                 oath_totp_validate2                 (const char *secret,
                                                         size_t secret_length,
                                                         time_t now,
                                                         unsigned  time_step_size,
                                                         time_t start_offset,
                                                         size_t window,
                                                         int *otp_pos,
                                                         const char *otp);
int                 oath_totp_validate2_callback        (const char *secret,
                                                         size_t secret_length,
                                                         time_t now,
                                                         unsigned  time_step_size,
                                                         time_t start_offset,
                                                         unsigned  digits,
                                                         size_t window,
                                                         int *otp_pos,
```

```
                                                    oath_validate_strcmp_function str
                                                    void *strcmp_handle);
int                 oath_totp_validate_callback    (const char *secret,
                                                    size_t secret_length,
                                                    time_t now,
                                                    unsigned  time_step_size,
                                                    time_t start_offset,
                                                    unsigned  digits,
                                                    size_t window,
                                                    oath_validate_strcmp_function str
                                                    void *strcmp_handle);
int                 (*oath_validate_strcmp_function)  (void *handle,
                                                    const char *test_otp);
```

Description

Details

OATHAPI

```
#  define OATHAPI __attribute__((__visibility__("default")))
```

OATH_HOTP_DYNAMIC_TRUNCATION

```
#define OATH_HOTP_DYNAMIC_TRUNCATION SIZE_MAX
```

OATH_HOTP_LENGTH()

```
#define OATH_HOTP_LENGTH(digits, checksum) (digits + (checksum ? 1 : 0))
```

OATH_TOTP_DEFAULT_START_TIME

```
#define OATH_TOTP_DEFAULT_START_TIME    ((time_t) 0)
```

OATH_TOTP_DEFAULT_TIME_STEP_SIZE

```
#define OATH_TOTP_DEFAULT_TIME_STEP_SIZE~30
```

OATH_VERSION

```
#define OATH_VERSION "2.2.0"
```

Pre-processor symbol with a string that describe the header file version number. Used together with oath_check_version() to verify header file and run-time library consistency.

OATH_VERSION_NUMBER

```
#define OATH_VERSION_NUMBER 0x02020000
```

Pre-processor symbol with a hexadecimal value describing the header file version number. For example, when the header version is 1.2.3 this symbol will have the value 0x01020300. The last two digits are only used between public releases, and will otherwise be 00.

oath_authenticate_usersfile ()

```
int                 oath_authenticate_usersfile         (const char *usersfile,
                                                         const char *username,
                                                         const char *otp,
                                                         size_t window,
                                                         const char *passwd,
                                                         time_t *last_otp);
```

Authenticate user named *username* with the one-time password *otp* and (optional) password *passwd*. Credentials are read (and updated) from a text file named *usersfile*.

Note that for TOTP the usersfile will only record the last OTP and use that to make sure more recent OTPs have not been seen yet when validating a new OTP. That logics relies on using the same search window for the same user.

***usersfile*:** string with user credential filename, in UsersFile format

***username*:** string with name of user

***otp*:** string with one-time password to authenticate

***window*:** how many past/future OTPs to search

***passwd*:** string with password, or NULL to disable password checking

***last_otp*:** output variable holding last successful authentication

***Returns*:** On successful validation, OATH_OK is returned. If the supplied *otp* is the same as the last successfully authenticated one-time password, OATH_REPLAYED_OTP is returned and the timestamp of the last authentication is returned in *last_otp*. If the one-time password is not found in the indicated search window, OATH_INVALID_OTP is returned. Otherwise, an error code is returned.

oath_base32_decode ()

```
int                 oath_base32_decode                  (const char *in,
                                                         size_t inlen,
                                                         char **out,
                                                         size_t *outlen);
```

Decode a base32 encoded string into binary data.

Space characters are ignored and pad characters are added if needed. Non-base32 data are not ignored but instead will lead to an OATH_INVALID_BASE32 error.

The *in* parameter should contain *inlen* bytes of base32 encoded data. The function allocates a new string in **out* to hold the decoded data, and sets **outlen* to the length of the data.

If *out* is NULL, then **outlen* will be set to what would have been the length of **out* on successful encoding.

If the caller is not interested in knowing the length of the output data *out*, then *outlen* may be set to NULL.

It is permitted but useless to have both *out* and *outlen* NULL.

in **:** input string with base32 encoded data of length *inlen*

inlen **:** length of input base32 string *in*

out **:** pointer to output variable for binary data of length *outlen*, or NULL

outlen **:** pointer to output variable holding length of *out*, or NULL

Returns **:** On success OATH_OK (zero) is returned, OATH_INVALID_BASE32 is returned if the input contains non-base32 characters, and OATH_MALLOC_ERROR is returned on memory allocation errors.

Since 1.12.0

oath_base32_encode ()

```
int                 oath_base32_encode                  (const char *in,
                                                         size_t inlen,
                                                         char **out,
                                                         size_t *outlen);
```

Encode binary data into a string with base32 data.

The *in* parameter should contain *inlen* bytes of data to encode. The function allocates a new string in **out* to hold the encoded data, and sets **outlen* to the length of the data. The output string **out* is zero-terminated (ASCII NUL), but the NUL is not counted in **outlen*.

If *out* is NULL, then **outlen* will be set to what would have been the length of **out* on successful encoding.

If the caller is not interested in knowing the length of the output data *out*, then *outlen* may be set to NULL.

It is permitted but useless to have both *out* and *outlen* NULL.

in **:** input string with binary data of length *inlen*

inlen **:** length of input data *in*

out **:** pointer to newly allocated output string of length *outlen*, or NULL

outlen **:** pointer to output variable holding length of *out*, or NULL

Returns **:** On success OATH_OK (zero) is returned, OATH_BASE32_OVERFLOW is returned if the output would be too large to store, and OATH_MALLOC_ERROR is returned on memory allocation errors.

Since 1.12.0

oath_bin2hex ()

```
void                oath_bin2hex                        (const char *binstr,
                                                         size_t binlen,
                                                         char *hexstr);
```

Convert binary data to NUL-terminated string with hex data. The output *hexstr* is allocated by the caller and must have room for at least 2**binlen*+1, to make room for the encoded data and the terminating NUL byte.

binstr **:** input binary data

binlen **:** length of input binary data *binstr*

hexstr **:** output string with hex data, must have room for 2**binlen*+1.

Since 1.12.0

oath_check_version ()

```
const char *        oath_check_version                  (const char *req_version);
```

Check OATH library version.

See OATH_VERSION for a suitable *req_version* string.

This function is one of few in the library that can be used without a successful call to oath_init().

req_version **:** version string to compare with, or NULL.

Returns **:** Check that the version of the library is at minimum the one given as a string in *req_version* and return the actual version string of the library; return NULL if the condition is not met. If NULL is passed to this function no check is done and only the version string is returned.

oath_done ()

```
int                 oath_done                           (void);
```

This function deinitializes the OATH library, which were initialized using oath_init(). After calling this function, no other OATH library function may be called except for to re-initialize the library using oath_init().

Returns **:** On success, OATH_OK (zero) is returned, otherwise an error code is returned.

oath_hex2bin ()

```
int                 oath_hex2bin                        (const char *hexstr,
                                                         char *binstr,
                                                         size_t *binlen);
```

Convert string with hex data to binary data.

Non-hexadecimal data are not ignored but instead will lead to an OATH_INVALID_HEX error.

If *binstr* is NULL, then *binlen* will be populated with the necessary length. If the *binstr* buffer is too small, OATH_TOO_SMALL_
is returned and *binlen* will contain the necessary length.

hexstr **:** input string with hex data

binstr **:** output string that holds binary data, or NULL

binlen **:** output variable holding needed length of *binstr*

Returns **:** On success, OATH_OK (zero) is returned, otherwise an error code is returned.

oath_hotp_generate ()

```
int                 oath_hotp_generate                  (const char *secret,
                                                         size_t secret_length,
                                                         uint64_t moving_factor,
                                                         unsigned  digits,
                                                         bool add_checksum,
                                                         size_t truncation_offset,
                                                         char *output_otp);
```

Generate a one-time-password using the HOTP algorithm as described in RFC 4226.

Use a value of OATH_HOTP_DYNAMIC_TRUNCATION for *truncation_offset* unless you really need a specific truncation offset.

To find out the size of the OTP you may use the OATH_HOTP_LENGTH() macro. The *output_otp* buffer must be have room for that length plus one for the terminating NUL.

Currently only values 6, 7 and 8 for *digits* are supported, and the *add_checksum* value is ignored. These restrictions may be lifted in future versions, although some limitations are inherent in the protocol.

secret **:** the shared secret string

secret_length **:** length of *secret*

moving_factor **:** a counter indicating the current OTP to generate

digits **:** number of requested digits in the OTP, excluding checksum

add_checksum **:** whether to add a checksum digit or not

truncation_offset **:** use a specific truncation offset

output_otp **:** output buffer, must have room for the output OTP plus zero

Returns **:** On success, OATH_OK (zero) is returned, otherwise an error code is returned.

oath_hotp_validate ()

```
int                 oath_hotp_validate                  (const char *secret,
                                                         size_t secret_length,
                                                         uint64_t start_moving_factor,
                                                         size_t window,
                                                         const char *otp);
```

Validate an OTP according to OATH HOTP algorithm per RFC 4226.

Currently only OTP lengths of 6, 7 or 8 digits are supported. This restrictions may be lifted in future versions, although some limitations are inherent in the protocol.

secret **:** the shared secret string

secret_length **:** length of *secret*

start_moving_factor **:** start counter in OTP stream

window **:** how many OTPs after start counter to test

otp **:** the OTP to validate.

Returns **:** Returns position in OTP window (zero is first position), or OATH_INVALID_OTP if no OTP was found in OTP window, or an error code.

oath_hotp_validate_callback ()

```
int                 oath_hotp_validate_callback         (const char *secret,
                                                         size_t secret_length,
                                                         uint64_t start_moving_factor,
                                                         size_t window,
                                                         unsigned  digits,
                                                         oath_validate_strcmp_function  ↩
                                                             strcmp_otp,
                                                         void *strcmp_handle);
```

Validate an OTP according to OATH HOTP algorithm per RFC 4226.

Validation is implemented by generating a number of potential OTPs and performing a call to the *strcmp_otp* function, to compare the potential OTP against the given *otp*. It has the following prototype:

int (*oath_validate_strcmp_function) (void *handle, const char *test_otp);

The function should be similar to strcmp in that it return 0 only on matches. It differs by permitting use of negative return codes as indication of internal failures in the callback. Positive values indicate OTP mismatch.

This callback interface is useful when you cannot compare OTPs directly using normal strcmp, but instead for example only have a hashed OTP. You would then typically pass in the hashed OTP in the *strcmp_handle* and let your implementation of *strcmp_otp* hash the test_otp OTP using the same hash, and then compare the results.

Currently only OTP lengths of 6, 7 or 8 digits are supported. This restrictions may be lifted in future versions, although some limitations are inherent in the protocol.

secret **:** the shared secret string

secret_length **:** length of *secret*

start_moving_factor **:** start counter in OTP stream

window **:** how many OTPs after start counter to test

digits **:** number of requested digits in the OTP

strcmp_otp **:** function pointer to a strcmp-like function.

strcmp_handle **:** caller handle to be passed on to *strcmp_otp*.

Returns **:** Returns position in OTP window (zero is first position), or OATH_INVALID_OTP if no OTP was found in OTP window, or an error code.

Since 1.4.0

oath_hotp_validate_strcmp_function

```
#define oath_hotp_validate_strcmp_function oath_validate_strcmp_function
```

oath_init ()

```
int                 oath_init                           (void);
```

This function initializes the OATH library. Every user of this library needs to call this function before using other functions. You should call oath_done() when use of the OATH library is no longer needed.

Note that this function may also initialize Libgcrypt, if the OATH library is built with libgcrypt support and libgcrypt has not been initialized before. Thus if you want to manually initialize libgcrypt you must do it before calling this function. This is useful in cases you want to disable libgcrypt's internal lockings etc.

Returns **:** On success, OATH_OK (zero) is returned, otherwise an error code is returned.

enum oath_rc

```
typedef enum {
  OATH_OK = 0,
  OATH_CRYPTO_ERROR = -1,
  OATH_INVALID_DIGITS = -2,
  OATH_PRINTF_ERROR = -3,
  OATH_INVALID_HEX = -4,
  OATH_TOO_SMALL_BUFFER = -5,
  OATH_INVALID_OTP = -6,
  OATH_REPLAYED_OTP = -7,
  OATH_BAD_PASSWORD = -8,
  OATH_INVALID_COUNTER = -9,
  OATH_INVALID_TIMESTAMP = -10,
  OATH_NO_SUCH_FILE = -11,
  OATH_UNKNOWN_USER = -12,
  OATH_FILE_SEEK_ERROR = -13,
  OATH_FILE_CREATE_ERROR = -14,
  OATH_FILE_LOCK_ERROR = -15,
  OATH_FILE_RENAME_ERROR = -16,
  OATH_FILE_UNLINK_ERROR = -17,
  OATH_TIME_ERROR = -18,
  OATH_STRCMP_ERROR = -19,
  OATH_INVALID_BASE32 = -20,
  OATH_BASE32_OVERFLOW = -21,
  OATH_MALLOC_ERROR = -22,
  OATH_FILE_FLUSH_ERROR = -23,
  OATH_FILE_SYNC_ERROR = -24,
  OATH_FILE_CLOSE_ERROR = -25,
  /* When adding anything here, update OATH_LAST_ERROR, errors.c
     and tests/tst_errors.c. */
  OATH_LAST_ERROR = -25
} oath_rc;
```

Return codes for OATH functions. All return codes are negative except for the successful code OATH_OK which are guaranteed to be 0. Positive values are reserved for non-error return codes.

Note that the oath_rc enumeration may be extended at a later date to include new return codes.

OATH_OK Successful return

OATH_CRYPTO_ERROR Internal error in crypto functions

OATH_INVALID_DIGITS Unsupported number of OTP digits

OATH_PRINTF_ERROR Error from system printf call

OATH_INVALID_HEX Hex string is invalid

OATH_TOO_SMALL_BUFFER The output buffer is too small

OATH_INVALID_OTP The OTP is not valid

OATH_REPLAYED_OTP The OTP has been replayed

OATH_BAD_PASSWORD The password does not match

OATH_INVALID_COUNTER The counter value is corrupt

OATH_INVALID_TIMESTAMP The timestamp is corrupt

OATH_NO_SUCH_FILE The supplied filename does not exist

OATH_UNKNOWN_USER Cannot find information about user

OATH_FILE_SEEK_ERROR System error when seeking in file

OATH_FILE_CREATE_ERROR System error when creating file

OATH_FILE_LOCK_ERROR System error when locking file

OATH_FILE_RENAME_ERROR System error when renaming file

OATH_FILE_UNLINK_ERROR System error when removing file

OATH_TIME_ERROR System error for time manipulation

OATH_STRCMP_ERROR A strcmp callback returned an error

OATH_INVALID_BASE32 Base32 string is invalid

OATH_BASE32_OVERFLOW Base32 encoding would overflow

OATH_MALLOC_ERROR Memory allocation failed

OATH_FILE_FLUSH_ERROR System error when flushing file buffer

OATH_FILE_SYNC_ERROR System error when syncing file to disk

OATH_FILE_CLOSE_ERROR System error when closing file

OATH_LAST_ERROR Meta-error indicating the last error code, for use when iterating over all error codes or similar.

oath_strerror ()

```
const char *        oath_strerror                       (int err);
```

Convert return code to human readable string explanation of the reason for the particular error code.

This string can be used to output a diagnostic message to the user.

This function is one of few in the library that can be used without a successful call to oath_init().

err : liboath error code

Returns : Returns a pointer to a statically allocated string containing an explanation of the error code *err*.

Since 1.8.0

oath_strerror_name ()

```
const char *        oath_strerror_name                  (int err);
```

Convert return code to human readable string representing the error code symbol itself. For example, oath_strerror_name(OATH_OK) returns the string "OATH_OK".

This string can be used to output a diagnostic message to the user.

This function is one of few in the library that can be used without a successful call to oath_init().

err : liboath error code

Returns : Returns a pointer to a statically allocated string containing a string version of the error code *err*, or NULL if the error code is not known.

Since 1.8.0

oath_totp_generate ()

```
int                 oath_totp_generate                (const char *secret,
                                                       size_t secret_length,
                                                       time_t now,
                                                       unsigned  time_step_size,
                                                       time_t start_offset,
                                                       unsigned  digits,
                                                       char *output_otp);
```

Generate a one-time-password using the time-variant TOTP algorithm described in RFC 6238. The input parameters are taken as time values.

The system parameter *time_step_size* describes how long the time window for each OTP is. The recommended value is 30 seconds, and you can use the value 0 or the symbol OATH_TOTP_DEFAULT_TIME_STEP_SIZE to indicate this.

The system parameter *start_offset* denote the Unix time when time steps are started to be counted. The recommended value is 0, to fall back on the Unix epoch) and you can use the symbol OATH_TOTP_DEFAULT_START_TIME to indicate this.

The *output_otp* buffer must have room for at least *digits* characters, plus one for the terminating NUL.

Currently only values 6, 7 and 8 for *digits* are supported. This restriction may be lifted in future versions.

secret **:** the shared secret string

secret_length **:** length of *secret*

now **:** Unix time value to compute TOTP for

time_step_size **:** time step system parameter (typically 30)

start_offset **:** Unix time of when to start counting time steps (typically 0)

digits **:** number of requested digits in the OTP, excluding checksum

output_otp **:** output buffer, must have room for the output OTP plus zero

Returns **:** On success, OATH_OK (zero) is returned, otherwise an error code is returned.

Since 1.4.0

oath_totp_validate ()

```
int                 oath_totp_validate                (const char *secret,
                                                       size_t secret_length,
                                                       time_t now,
                                                       unsigned  time_step_size,
                                                       time_t start_offset,
                                                       size_t window,
                                                       const char *otp);
```

Validate an OTP according to OATH TOTP algorithm per RFC 6238.

Currently only OTP lengths of 6, 7 or 8 digits are supported. This restrictions may be lifted in future versions, although some limitations are inherent in the protocol.

secret **:** the shared secret string

secret_length **:** length of *secret*

now **:** Unix time value to validate TOTP for

time_step_size **:** time step system parameter (typically 30)

start_offset : Unix time of when to start counting time steps (typically 0)

window : how many OTPs after/before start OTP to test

otp : the OTP to validate.

Returns : Returns absolute value of position in OTP window (zero is first position), or OATH_INVALID_OTP if no OTP was found in OTP window, or an error code.

Since 1.6.0

oath_totp_validate2 ()

```
int                 oath_totp_validate2                 (const char *secret,
                                                         size_t secret_length,
                                                         time_t now,
                                                         unsigned  time_step_size,
                                                         time_t start_offset,
                                                         size_t window,
                                                         int *otp_pos,
                                                         const char *otp);
```

Validate an OTP according to OATH TOTP algorithm per RFC 6238.

Currently only OTP lengths of 6, 7 or 8 digits are supported. This restrictions may be lifted in future versions, although some limitations are inherent in the protocol.

secret : the shared secret string

secret_length : length of *secret*

now : Unix time value to validate TOTP for

time_step_size : time step system parameter (typically 30)

start_offset : Unix time of when to start counting time steps (typically 0)

window : how many OTPs after/before start OTP to test

otp_pos : output search position in search window (may be NULL).

otp : the OTP to validate.

Returns : Returns absolute value of position in OTP window (zero is first position), or OATH_INVALID_OTP if no OTP was found in OTP window, or an error code.

Since 1.10.0

oath_totp_validate2_callback ()

```
int                 oath_totp_validate2_callback        (const char *secret,
                                                         size_t secret_length,
                                                         time_t now,
                                                         unsigned  time_step_size,
                                                         time_t start_offset,
                                                         unsigned  digits,
                                                         size_t window,
                                                         int *otp_pos,
                                                         oath_validate_strcmp_function  ↩
                                                             strcmp_otp,
                                                         void *strcmp_handle);
```

Validate an OTP according to OATH TOTP algorithm per RFC 6238.

Validation is implemented by generating a number of potential OTPs and performing a call to the *strcmp_otp* function, to compare the potential OTP against the given *otp*. It has the following prototype:

int (*oath_validate_strcmp_function) (void *handle, const char *test_otp);

The function should be similar to strcmp in that it return 0 only on matches. It differs by permitting use of negative return codes as indication of internal failures in the callback. Positive values indicate OTP mismatch.

This callback interface is useful when you cannot compare OTPs directly using normal strcmp, but instead for example only have a hashed OTP. You would then typically pass in the hashed OTP in the *strcmp_handle* and let your implementation of *strcmp_otp* hash the test_otp OTP using the same hash, and then compare the results.

Currently only OTP lengths of 6, 7 or 8 digits are supported. This restrictions may be lifted in future versions, although some limitations are inherent in the protocol.

secret : the shared secret string

secret_length : length of *secret*

now : Unix time value to compute TOTP for

time_step_size : time step system parameter (typically 30)

start_offset : Unix time of when to start counting time steps (typically 0)

digits : number of requested digits in the OTP

window : how many OTPs after start counter to test

otp_pos : output search position in search window (may be NULL).

strcmp_otp : function pointer to a strcmp-like function.

strcmp_handle : caller handle to be passed on to *strcmp_otp*.

Returns : Returns absolute value of position in OTP window (zero is first position), or OATH_INVALID_OTP if no OTP was found in OTP window, or an error code.

Since 1.10.0

oath_totp_validate_callback ()

```
int                 oath_totp_validate_callback         (const char *secret,
                                                         size_t secret_length,
                                                         time_t now,
                                                         unsigned  time_step_size,
                                                         time_t start_offset,
                                                         unsigned  digits,
                                                         size_t window,
                                                         oath_validate_strcmp_function  ↩
                                                             strcmp_otp,
                                                         void *strcmp_handle);
```

Validate an OTP according to OATH TOTP algorithm per RFC 6238.

Validation is implemented by generating a number of potential OTPs and performing a call to the *strcmp_otp* function, to compare the potential OTP against the given *otp*. It has the following prototype:

int (*oath_validate_strcmp_function) (void *handle, const char *test_otp);

The function should be similar to strcmp in that it return 0 only on matches. It differs by permitting use of negative return codes as indication of internal failures in the callback. Positive values indicate OTP mismatch.

This callback interface is useful when you cannot compare OTPs directly using normal strcmp, but instead for example only have a hashed OTP. You would then typically pass in the hashed OTP in the *strcmp_handle* and let your implementation of *strcmp_otp* hash the test_otp OTP using the same hash, and then compare the results.

Currently only OTP lengths of 6, 7 or 8 digits are supported. This restrictions may be lifted in future versions, although some limitations are inherent in the protocol.

secret **:** the shared secret string

secret_length **:** length of *secret*

now **:** Unix time value to compute TOTP for

time_step_size **:** time step system parameter (typically 30)

start_offset **:** Unix time of when to start counting time steps (typically 0)

digits **:** number of requested digits in the OTP

window **:** how many OTPs after start counter to test

strcmp_otp **:** function pointer to a strcmp-like function.

strcmp_handle **:** caller handle to be passed on to *strcmp_otp*.

Returns **:** Returns position in OTP window (zero is first position), or OATH_INVALID_OTP if no OTP was found in OTP window, or an error code.

Since 1.6.0

oath_validate_strcmp_function ()

```
int                 (*oath_validate_strcmp_function)    (void *handle,
                                                         const char *test_otp);
```

Prototype of strcmp-like function that will be called by oath_hotp_validate_callback() or oath_totp_validate_callback() to validate OTPs.

The function should be similar to strcmp in that it return 0 only on matches. It differs by permitting use of negative return codes as indication of internal failures in the callback. Positive values indicate OTP mismatch.

This callback interface is useful when you cannot compare OTPs directly using normal strcmp, but instead for example only have a hashed OTP. You would then typically pass in the hashed OTP in the *strcmp_handle* and let your implementation of *oath_strcmp* hash the test_otp OTP using the same hash, and then compare the results.

handle **:** caller handle as passed to oath_hotp_validate_callback()

test_otp **:** OTP to match against.

Returns **:** 0 if and only if *test_otp* is identical to the OTP to be validated. Negative value if an internal failure occurs. Positive value if the test_otp simply doesn't match.

Since 1.6.0

Chapter 2

Index