

---

# Liboath API Reference Manual

---

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> Liboath API Reference Manual		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		December 27, 2010	

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>Liboath API Reference Manual</b>	<b>1</b>
1.1	oath . . . . .	1
<b>2</b>	<b>Index</b>	<b>7</b>

## Chapter 1

# Liboath API Reference Manual

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

Liboath and this manual are licensed under the LGPLv2.1+. This manual is actually automatically generated from the source code. See COPYING in the package for more licensing information.

## 1.1 oath

oath —

### Synopsis

```
#define OATHAPI
#define OATH_VERSION
#define OATH_VERSION_NUMBER
enum oath_rc;
#define OATH_DYNAMIC_TRUNCATION
int oath_init (void);
int oath_done (void);
const char * oath_check_version (const char *req_version);
int oath_hex2bin (char *hexstr, char *binstr, size_t *binlen);

#define OATH_HOTP_LENGTH
int oath_hotp_generate (digits, checksum, 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_authenticate_usersfile (const char *usersfile, const char *username,
```

```
const char *otp,  
size_t window,  
const char *passwd,  
time_t *last_otp);
```

## Description

## Details

### OATHAPI

```
#define OATHAPI
```

### OATH\_VERSION

```
# define OATH_VERSION "1.0.1"
```

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 0x010001
```

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 0x010203.

### 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_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\_DYNAMIC\_TRUNCATION**

```
#define OATH_DYNAMIC_TRUNCATION SIZE_MAX
```

#### **oath\_init ()**

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

#### **Returns :**

#### **oath\_done ()**

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

#### **Returns :**

---

## 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\_hex2bin ()

```
int          oath_hex2bin          (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\_LENGTH()

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

*digits:*

**checksum :**

## 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_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 from 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\_authenticate\_usersfile ()

[illegible]



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*.

***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 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.

---

## Chapter 2

# Index

### O

- oath\_authenticate\_usersfile, [5](#)
- oath\_check\_version, [4](#)
- oath\_done, [3](#)
- OATH\_DYNAMIC\_TRUNCATION, [3](#)
- oath\_hex2bin, [4](#)
- oath\_hotp\_generate, [4](#)
- OATH\_HOTP\_LENGTH, [4](#)
- oath\_hotp\_validate, [5](#)
- oath\_init, [3](#)
- oath\_rc, [2](#)
- OATH\_VERSION, [2](#)
- OATH\_VERSION\_NUMBER, [2](#)
- OATHAPI, [2](#)

---