Mhash

Introduction

These functions are intended to work with » mhash. Mhash can be used to create checksums, message digests, message authentication codes, and more.

This is an interface to the mhash library. mhash supports a wide variety of hash algorithms such as MD5, SHA1, GOST, and many others. For a complete list of supported hashes, refer to the documentation of mhash. The general rule is that you can access the hash algorithm from PHP with MHASH_HASHNAME. For example, to access TIGER you use the PHP constant MHASH_TIGER.

Note
This extension is obsoleted by Hash.

Installing/Configuring

Requirements

To use it, download the mhash distribution from » its web site and follow the included installation instructions.

Installation

You need to compile PHP with the *--with-mhash[=DIR]* parameter to enable this extension. DIR is the mhash install directory.

Runtime Configuration

This extension has no configuration directives defined in *php.ini*.

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Here is a list of hashes which are currently supported by mhash. If a hash is not listed here, but is listed by mhash as supported, you can safely assume that this documentation is outdated.

- **MHASH_ADLER32**
- **MHASH_CRC32**
- **MHASH_CRC32B**
- **MHASH_GOST**
- **MHASH_HAVAL128**
- **MHASH_HAVAL160**
- **MHASH_HAVAL192**
- **MHASH_HAVAL256**
- **MHASH_MD4**
- **MHASH_MD5**
- **MHASH_RIPEMD160**
- **MHASH_SHA1**
- **MHASH_SHA256**
- **MHASH_TIGER**
- **MHASH_TIGER128**
- **MHASH_TIGER160**

Examples

Example #1 - Compute the MD5 digest and hmac and print it out as hex

```
<?php
$input = "what do ya want for nothing?";
$hash = mhash(MHASH_MD5, $input);
echo "The hash is " . bin2hex($hash) . "<br />\n";
$hash = mhash(MHASH_MD5, $input, "Jefe");
echo "The hmac is " . bin2hex($hash) . "<br />\n";
?>
```

This will produce:

```
The hash is d03cb659cbf9192dcd066272249f8412
The hmac is 750c783e6ab0b503eaa86e310a5db738
```

Mhash Functions

mhash_count

mhash_count -- Get the highest available hash id

Description

int **mhash_count** (void)

Gets the highest available hash id.

Return Values

Returns the highest available hash id. Hashes are numbered from 0 to this hash id.

Examples

Example #2 - Traversing all hashes

```
<?php

$nr = mhash_count();

for ($i = 0; $i <= $nr; $i++) {
   echo sprintf("The blocksize of %s is %d\n",
       mhash_get_hash_name($i),
       mhash_get_block_size($i));
}
?>
```

mhash_get_block_size

mhash_get_block_size -- Get the block size of the specified hash

Description

int **mhash_get_block_size** (int `$hash`)

Gets the size of a block of the specified *`hash`*.

Parameters

`hash`
The hash id. One of the *MHASH_XXX* constants.

Return Values

Returns the size in bytes or **FALSE**, if the *`hash`* does not exist.

Examples

Example #3 - mhash_get_block_size() Example

```
<?php

echo mhash_get_block_size(MHASH_MD5); // 16

?>
```

mhash_get_hash_name

mhash_get_hash_name -- Get the name of the specified hash

Description

string **mhash_get_hash_name** (int `$hash`)

Gets the name of the specified *`hash`*.

Parameters

`hash`
The hash id. One of the *MHASH_XXX* constants.

Return Values

Returns the name of the hash or **FALSE**, if the hash does not exist.

Examples

Example #4 - mhash_get_hash_name() example

```
<?php

echo mhash_get_hash_name(MHASH_MD5); // MD5

?>
```

mhash_keygen_s2k

mhash_keygen_s2k -- Generates a key

Description

string **mhash_keygen_s2k** (int `$hash`, string `$password`, string `$salt`, int `$bytes`)

Generates a key according to the *`hash`* given a user provided *`password`*.

This is the Salted S2K algorithm as specified in the OpenPGP document (» RFC 2440).

Keep in mind that user supplied passwords are not really suitable to be used as keys in cryptographic algorithms, since users normally choose keys they can write on keyboard. These passwords use only 6 to 7 bits per character (or less). It is highly recommended to use some kind of transformation (like this function) to the user supplied key.

Parameters

`hash`
: The hash id used to create the key. One of the *MHASH_XXX* constants.

`password`
: User supplied password.

`salt`
: Must be different and random enough for every key you generate in order to create different keys. That salt must be known when you check the keys, thus it is a good idea to append the key to it. Salt has a fixed length of 8 bytes and will be padded with zeros if you supply less bytes.

`bytes`
: The key length, in bytes.

Return Values

Returns the generated key as a string, or **FALSE** on error.

mhash

mhash -- Compute hash

Description

string **mhash** (int `$hash`, string `$data` [, string `$key`])

mhash() applies a hash function specified by *`hash`* to the *`data`*.

Parameters

`hash`
: The hash id. One of the *MHASH_XXX* constants.

`data`
: The user input, as a string.

`key`
: If specified, the function will return the resulting HMAC instead. HMAC is keyed hashing for message authentication, or simply a message digest that depends on the specified key. Not all algorithms supported in mhash can be used in HMAC mode.

Return Values

Returns the resulting hash (also called digest) or HMAC as a string, or **FALSE** on errors.