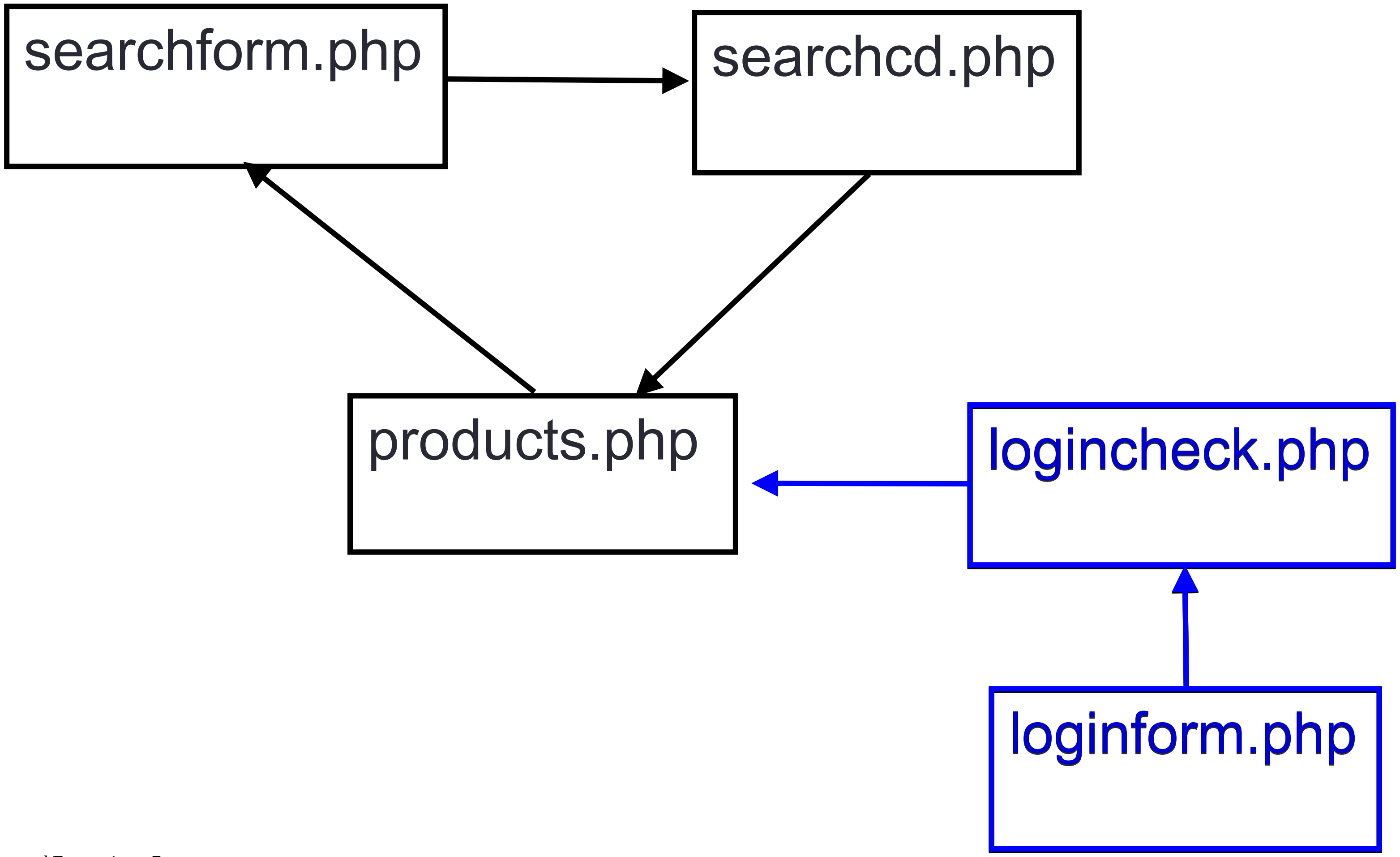
Dynamic Web Development

Lecture 18

Searching and Encryption

Searching

Page Map

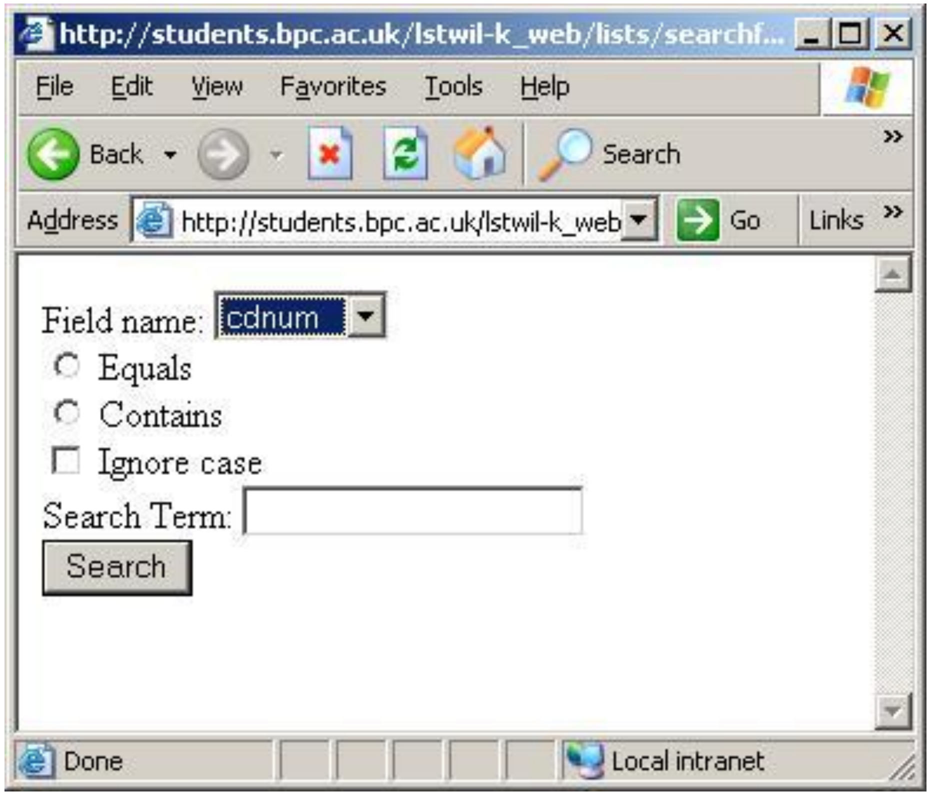

searchform.php
searchcd.php
products.php
logincheck.php
loginform.php

What should appear on search form?

Search Term

Search field

Comparison



http://students.bpc.ac.uk/lstwil-k_web/lists/searchf...
File Edit View Favorites Tools Help
Back Search
Address http://students.bpc.ac.uk/lstwil-k_web Go Links
Done Local intranet

Field name: cdnum

◯ Equals

◯ Contains

☐ Ignore case

Search Term:

Search

searchform.php (first part)

```php
<?php
session_start();

if ( !isset($_SESSION['username']) )
 {
  header("Location:loginform.php");
  exit();
 }

require "dbconn.php";

$query1 = "SELECT *
          FROM cdtable";

$results1 = $connect->query($query1);
```

Up to now, we have used the `mysqli_fetch_assoc()` function to extract the data values from the returned database rows.

However, the `query( )` returns more than just the data.

We can also access the field names, using the following functions:

```
mysqli_field_count( )
mysqli_fetch_field_direct( )
```

searchform.php (second part)

```
$field = $connect->field_count( $results1 );

for ( $i = 0; $i < $field; $i++ )
{
  $names[$i] = $results1->fetch_field_direct( $i );
}

?>
```

searchform.php (third part)

```
<html>
<body>
 <form name="searchcd" action="searchcd.php"
                                         method="get">
    Field name: <select name="fieldname">

<?php

for ( $x = 0; $x < $field; $x++)
{
echo "<option value='"
       .$names[$x]."'>".$names[$x]."</option>";
}

?>
                     </select>
```

html output by the loop

```
Field name:
<select name="fieldname">
  <option value='cdnum'>cdnum</option>
  <option value='cdname'>cdname</option>
  <option value='artist'>artist</option>
</select>
```

Searchform.php (fourth part)

```
  <br />
   <input type="radio" name="comparison" value="equal" /> Equals
   <br />
   <input type="radio" name="comparison" value="contain" /> Contains
   <br />
   <input type="checkbox" name="casetick" value="ignore" /> Ignore case
   <br />
   Search Term: <input type="text" name="searchterm"></input>
   <br />
   <input type="submit" name="Search" value="Search"></input>
</form>
</body>
</html>
```

searchcd.php (first part)

```
require "dbconn.php";

$field = $_GET['fieldname'];
$term = $_GET['searchterm'];
$comp = $_GET['comparison'];
$case = $_GET['casetick'];

if ($comp == "equal")
{
$query1 = "SELECT * FROM cdtable
              WHERE ".$field." = '".$term."'";
}
else
{
$query1 = "SELECT * FROM cdtable
              WHERE ".$field." LIKE '%".$term."%'";
}
```

will produce a query like this:

```
SELECT * FROM cdtable
WHERE artist = 'Hawkwind'
```

or

```
SELECT * FROM cdtable
WHERE artist LIKE '%awk%'
```

Ignoring Case

There is a mysql sql function called `UPPER()` which is used to convert all text to upper case. There is also one called `LOWER()`.

```
if ($case == "ignore")
{
$query1 = "SELECT * FROM cdtable
       WHERE UPPER(".$field.") = UPPER('".$term."')";
}
else
{
$query1 = "SELECT * FROM cdtable
              WHERE ".$field." = '".$term."'";
}
```

I'll leave it as an exercise for you to figure out how to combine both if..then statements into one jumbo set of nested if..then statements which can take care of both types of comparison and ignoring case.

Encryption

Encryption

A simple approach will store passwords in plain text.

This is obviously bad from a security point of view.

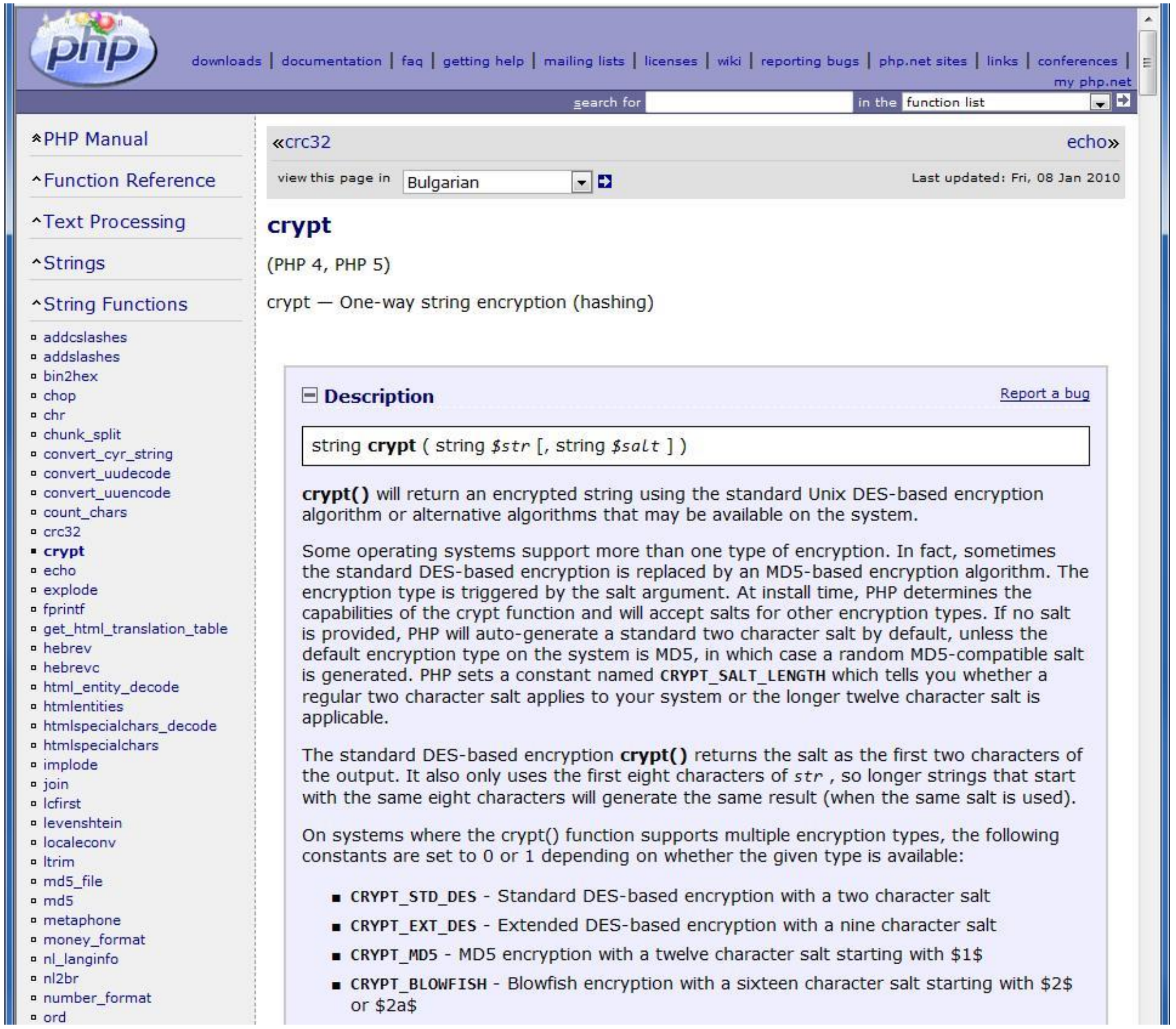
Especially considering this information is being transmitted across a network.

The use of packet capture software such as Wireshark could be used to examine other people's passwords.

PHP does provide functions which will:

– hash data	(one way)	crypt() function
– encrypt data	(two way)	various function libraries

http://uk2.php.net/manual/en/function.crypt.php



downloads | documentation | faq | getting help | mailing lists | licenses | wiki | reporting bugs | php.net sites | links | conferences | my php.net

search for [ ] in the function list

PHP Manual
Function Reference
Text Processing
Strings
String Functions

- addcslashes
- addslashes
- bin2hex
- chop
- chr
- chunk_split
- convert_cyr_string
- convert_uudecode
- convert_uuencode
- count_chars
- crc32
- **crypt**
- echo
- explode
- fprintf
- get_html_translation_table
- hebrev
- hebrevc
- html_entity_decode
- htmlentities
- htmlspecialchars_decode
- htmlspecialchars
- implode
- join
- lcfirst
- levenshtein
- localeconv
- ltrim
- md5_file
- md5
- metaphone
- money_format
- nl_langinfo
- nl2br
- number_format
- ord

«crc32 echo»

view this page in Bulgarian

Last updated: Fri, 08 Jan 2010

crypt

(PHP 4, PHP 5)

crypt — One-way string encryption (hashing)

Description Report a bug

string **crypt** (string *$str* [, string *$salt*])

crypt() will return an encrypted string using the standard Unix DES-based encryption algorithm or alternative algorithms that may be available on the system.

Some operating systems support more than one type of encryption. In fact, sometimes the standard DES-based encryption is replaced by an MD5-based encryption algorithm. The encryption type is triggered by the salt argument. At install time, PHP determines the capabilities of the crypt function and will accept salts for other encryption types. If no salt is provided, PHP will auto-generate a standard two character salt by default, unless the default encryption type on the system is MD5, in which case a random MD5-compatible salt is generated. PHP sets a constant named `CRYPT_SALT_LENGTH` which tells you whether a regular two character salt applies to your system or the longer twelve character salt is applicable.

The standard DES-based encryption **crypt()** returns the salt as the first two characters of the output. It also only uses the first eight characters of *str* , so longer strings that start with the same eight characters will generate the same result (when the same salt is used).

On systems where the crypt() function supports multiple encryption types, the following constants are set to 0 or 1 depending on whether the given type is available:

- `CRYPT_STD_DES` - Standard DES-based encryption with a two character salt
- `CRYPT_EXT_DES` - Extended DES-based encryption with a nine character salt
- `CRYPT_MD5` - MD5 encryption with a twelve character salt starting with 1
- `CRYPT_BLOWFISH` - Blowfish encryption with a sixteen character salt starting with 2 or $2a$

Use of a Salt value

Using a standard encryption algorithm has a weakness.

- You take a dictionary of common passwords and use it to generate the encrypted versions of each password.
- Also known as a rainbow table.

- Then run a brute force attack, by comparing the unknown encrypted password against every entry in your encrypted dictionary.

By providing a 'salt' value, an extra element of randomness is introduced into the encryption process.

The same password can be hashed thousands of different ways, which means it takes a lot longer to generate the rainbow table.

The same password can generate a large number of different hash values, depending on the value of the salt.

The crypt() function decides which algorithm to use, depending on the type of salt value you provide.

If you don't provide one, it auto-generates a two character salt by default, usually based on system time.

It outputs the salt as the first two characters of the hashed text.

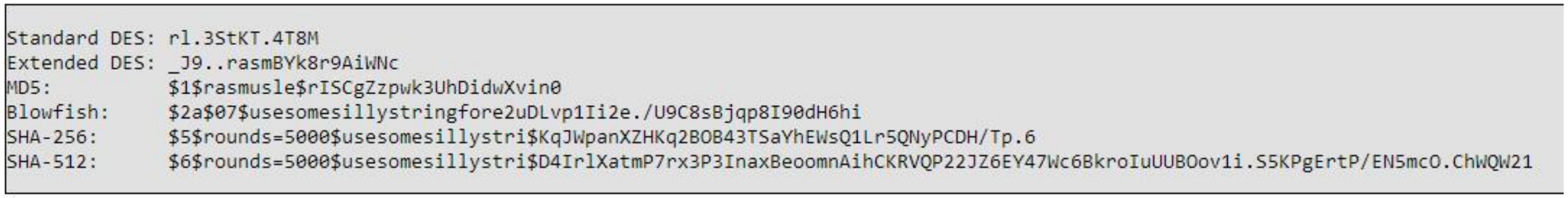
This means that the hashed text can then be used as the salt for comparison purposes.

Encrypted Versions of Text "rasmuslerdorf"

Standard DES - 2 char salt: `rl`

Extended DES - 9 char salt: `_J9..rasm`

MD5 - 12 char salt starting with $1$: `$1$rasmusle$`



```
Standard DES: rl.3StKT.4T8M
Extended DES: _J9..rasmBYk8r9AiWNc
MD5:          $1$rasmusle$rISCgZzpwk3UhDidwXvin0
Blowfish:     $2a$07$usesomesillystringfore2uDLvp1Ii2e./U9C8sBjqp8I90dH6hi
SHA-256:      $5$rounds=5000$usesomesillystri$KqJWpanXZHKq2BOB43TSaYhEWsQ1Lr5QNyPCDH/Tp.6
SHA-512:      $6$rounds=5000$usesomesillystri$D4IrlXatmP7rx3P3InaxBeoomnAihCKRVQP22JZ6EY47Wc6BkroIuUUBOov1i.S5KPgErtP/EN5mcO.ChWQW21
```

Blowfish - 29 char salt starting with $2a$:

`$2a$07$usessomesillystringfor`

SHA-256 - 16 char salt starting with $5$ and an optional rounds parameter:

`$5$rounds=5000$usesomesillystri`

SHA-512 - 16 char salt starting with $6$ and an optional rounds parameter:

`$6$rounds=5000$usesomesillystri`

Use of crypt() hashing function

```
<?php

$salt_and_hashed_password = crypt('mypassword');

// let the salt be automatically generated from the stored
// password

/* You should pass the entire results of crypt() as the salt
for comparing a password, to avoid problems when different
hashing algorithms are used */

if (crypt($user_input, $salt_and_hashed_password)
                                        == $hashed_password)
{
   echo "Password verified!";
}

?>
```

Use of crypt() hashing function

```
<?php

$salt_and_hashed_password = crypt('mypassword','$1$rasmusle$');

// This time the checking program supplies an MD5 salt value
// The same one that was used when the passwords were originally
// being stored in the database

/* You should pass the entire results of crypt() as the salt for
comparing a password to avoid problems when different hashing
algorithms are used */

if (crypt($user_input, $salt_and_hashed_password)
                                               == $hashed_password)
{
   echo "Password verified!";
}

?>
```

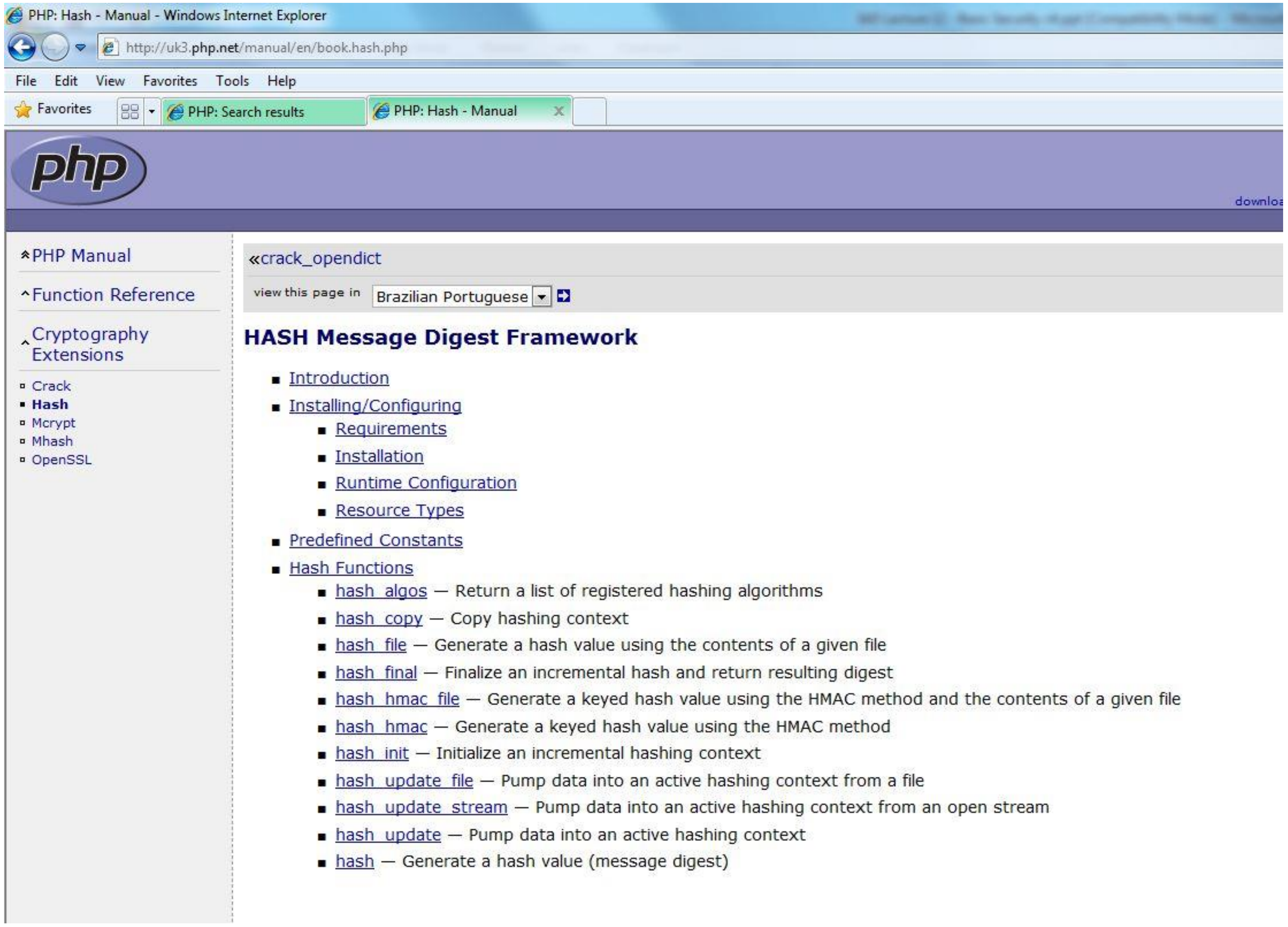
PHP documentation recommends using the hash_equals() function to do this comparison, to protect against timing attacks

PHP Encryption Libraries

These modules usually have to be installed separately.

Crack	These functions allow you to use the CrackLib library to test the 'strength' of a password, by checking length, use of upper and lower case and checked against the specified CrackLib dictionary.
Hash	Message Digest (hash) engine. Allows direct or incremental processing of arbitrary length messages using a variety of hashing algorithms.
Mcrypt	This is an interface to the mcrypt library, which supports a wide variety of block algorithms such as DES, TripleDES, Blowfish (default), 3-WAY, SAFER-SK64, SAFER-SK128, TWOFISH, TEA, RC2 and GOST in CBC, OFB, CFB and ECB cipher modes.
Mhash	Mhash supports a wide variety of hash algorithms such as MD5, SHA1, GOST, and many others. **Note**: This extension is made obsolete by Hash.
OpenSSL	This module uses the functions of OpenSSL for generation and verification of signatures and for sealing (encrypting) and opening (decrypting) data. OpenSSL offers many features that this module currently doesn't support. Some of these may be added in the future.

http://uk3.php.net/manual/en/book.hash.php


PHP: Hash - Manual - Windows Internet Explorer
http://uk3.php.net/manual/en/book.hash.php
File Edit View Favorites Tools Help
Favorites
PHP: Search results
PHP: Hash - Manual
php
downloa

PHP Manual

Function Reference

Cryptography Extensions

- Crack
- **Hash**
- Mcrypt
- Mhash
- OpenSSL

«crack_opendict

view this page in Brazilian Portuguese

HASH Message Digest Framework

- Introduction
- Installing/Configuring
  - Requirements
  - Installation
  - Runtime Configuration
  - Resource Types
- Predefined Constants
- Hash Functions
  - hash_algos — Return a list of registered hashing algorithms
  - hash_copy — Copy hashing context
  - hash_file — Generate a hash value using the contents of a given file
  - hash_final — Finalize an incremental hash and return resulting digest
  - hash_hmac_file — Generate a keyed hash value using the HMAC method and the contents of a given file
  - hash_hmac — Generate a keyed hash value using the HMAC method
  - hash_init — Initialize an incremental hashing context
  - hash_update_file — Pump data into an active hashing context from a file
  - hash_update_stream — Pump data into an active hashing context from an open stream
  - hash_update — Pump data into an active hashing context
  - hash — Generate a hash value (message digest)