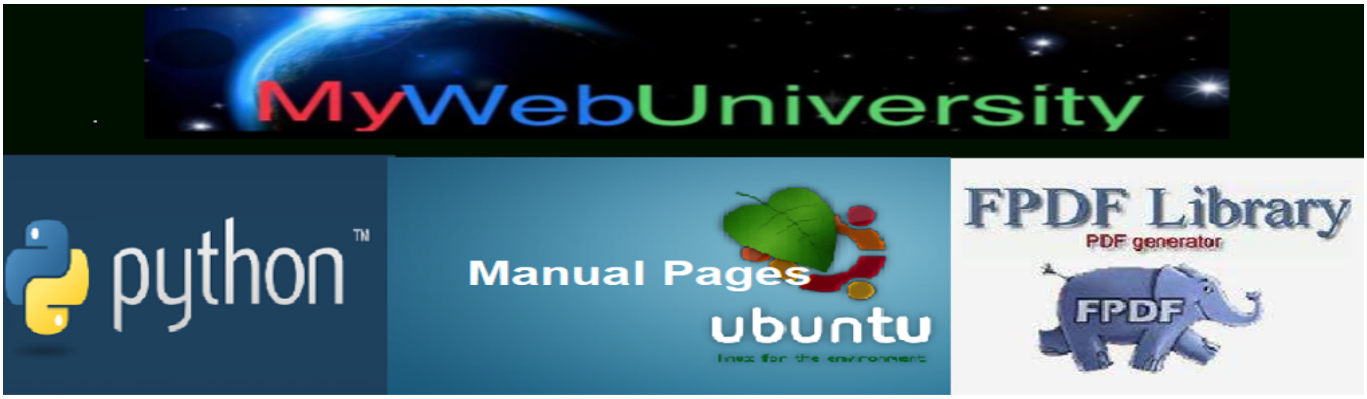

Rocky Enterprise Linux 9.2 Manual Pages on command 'Crypt::PBKDF2.3pm'

$ man Crypt::PBKDF2.3pm

Crypt::PBKDF2(3pm) User Contributed Perl Documentation Crypt::PBKDF2(3pm)

NAME

Crypt::PBKDF2 - The PBKDF2 password hashing algorithm.

VERSION

version 0.161520

SYNOPSIS

```
use Crypt::PBKDF2;

my $pbkdf2 = Crypt::PBKDF2->new(
    hash_class => 'HMACSHA1', # this is the default
    iterations => 1000,       # so is this
    output_len => 20,         # and this
    salt_len => 4,            # and this.
);

my $hash = $pbkdf2->generate("s3kr1t_password");
if ($pbkdf2->validate($hash, "s3kr1t_password")) {
    access_granted();
}
```

DESCRIPTION

PBKDF2 is a secure password hashing algorithm that uses the techniques of "key strengthening" to make the complexity of a brute-force attack arbitrarily high. PBKDF2 uses any other cryptographic hash or cipher (by convention, usually HMAC-SHA1, but "Crypt::PBKDF2" is fully pluggable), and allows for an arbitrary number of iterations of the hashing function, and a nearly unlimited output hash size (up to 2**32 - 1 times the size of the output of the backend hash). The hash is salted, as any password hash should be, and the salt may also be of arbitrary size.

ATTRIBUTES

hash_class

Type: String, Default: HMACSHA1

The name of the default class that will provide PBKDF2's Pseudo-Random Function (the backend hash). If the value starts with a "+", the "+" will be removed and the remainder will be taken as a fully-qualified package name. Otherwise, the value will be appended to "Crypt::PBKDF2::Hash::".

hash_args

Type: HashRef, Default: {}

Arguments to be passed to the "hash_class" constructor.

hasher

Type: Object (must fulfill role Crypt::PBKDF2::Hash), Default: None.

It is also possible to provide a hash object directly; in this case the "hash_class" and "hash_args" are ignored.

iterations

Type: Integer, Default: 1000.

The default number of iterations of the hashing function to use for the "generate" and

"PBKDF2" methods.

output_len

Type: Integer.

The default size (in bytes, not bits) of the output hash. If a value isn't provided, the output size depends on the "hash_class"?/?"hasher" selected, and will equal the output size of the backend hash (e.g. 20 bytes for HMACSHA1).

salt_len

Type: Integer, Default: 4

The default salt length (in bytes) for the "generate" method.

encoding

Type: String (either "crypt" or "ldap"), Default: "ldap"

The hash format to generate. The "ldap" format is intended to be compatible with RFC2307, and looks like:

```
{X-PBKDF2}HMACSHA1:AAAD6A:8ODUPA==:1HSdSVVwlWSZhbPGO7GIZ4iUbrk=
```

While the "crypt" format is similar to the format used by the "crypt()" function, except with more structured information in the second (salt) field. It looks like:

```
$PBKDF2$HMACSHA1:1000:4q9OTg==$9Pb6bCRgnct/dga+4v4Lyv8x31s=
```

Versions of this module up to 0.110461 generated the "crypt" format, so set that if you want it. Current versions of this module will read either format, but the "ldap" format is preferred.

length_limit

Type: Integer

The maximum password length to allow, for generate and verify functions. Allowing passwords of unlimited length can allow a denial-of-service attack in which an attacker asks the server to validate very large passwords.

For compatibility this attribute is unset by default, but it is recommended to set it to a reasonably small value like 100 -- large enough that users aren't discouraged from having secure passwords, but small enough to limit the computation needed to validate any one password.

METHODS

generate ($password, [$salt])

Generates a hash for the given $password. If $salt is not provided, a random salt with length "salt_len" will be generated.

There are two output formats available, depending on the setting of the "encoding" attribute: "ldap" and "crypt"; see the documentation for "encoding" for more information.

validate ($hashed, $password)

Validates whether the password $password matches the hash string $hashed. May throw an exception if the format of $hashed is invalid; otherwise, returns true or false. Accepts both formats that the "generate" method can produce.

PBKDF2 ($salt, $password)

The raw PBKDF2 algorithm. Given the $salt and $password, returns the raw binary hash.

PBKDF2_base64 ($salt, $password)

As the "PBKDF2" method, only the output is encoded with MIME::Base64.

PBKDF2_hex ($salt, $password)

As the "PBKDF2" method, only the output is encoded in hexadecimal.

encode_string ($salt, $hash)

Given a generated salt and hash, hash, generates output in the form generated by "generate" and accepted by "validate". Unlikely to be of much use to anyone else.

decode_string ($hashed)

Given a textual hash in the form generated by "generate", decodes it and returns a HashRef containing:

- "algorithm": A string representing the hash algorithm used. See "hasher_from_algorithm ($algo_str)".
- "iterations": The number of iterations used.
- "salt": The salt, in raw binary form.
- "hash": The hash, in raw binary form.

This method is mostly for internal use, but it has been left public as it may come in handy. If the input data is invalid, this method may throw an exception.

hasher_from_algorithm ($algo_str)

Attempts to load and instantiate a "Crypt::PBKDF2::Hash::*" class based on an algorithm string as produced by "encode_string" / "generate".

clone (%params)

Create a new object like this one, but with %params changed.

SEE ALSO

- Wikipedia: PBKDF2: <http://en.wikipedia.org/wiki/PBKDF2>
- RFC2898, PKCS#5 version 2.0: <http://tools.ietf.org/html/rfc2898>
- RFC2307, Using LDAP as a Network Information Service: <http://tools.ietf.org/html/rfc2307>

AUTHOR

Andrew Rodland <arodland@cpan.org>

COPYRIGHT AND LICENSE