Thomas Fahle

HowTo - Crypt::SaltedHash

Crypt::SaltedHash - eine einfach zu bedienende Bibliothek zum Erzeugen und Validieren gesalzener Hashes

Crypt::SaltedHash von Sascha Kiefer bzw. Gerda Shank ist eine einfach zu bedienende Bibliothek zum Erzeugen und Validieren *gesalzener Hashes*.

Salted Hashes werden oft zur sicheren Speicherung von Passwörtern verwendet. Über die Sicherheit gesalzener Hashes mag man trefflich streiten. Diese kleine Anleitung wendet sich an Anwender dieses in der Praxis oft anzutreffenden Verfahrens und behandelt die beiden praxisrelevanten Fälle Erzeugen und Prüfen von *Salted Hashes*.

Beispiel: Salted Hash erzeugen

Die Methode `generate()` erzeugt einen gesalzenen Hash aus einem Klartextpasswort und gibt diesen als RFC2307-(userPassword)-codierte Zeichenkette zurück.

```
#!/usr/bin/perl
use strict;
use warnings;

use Crypt::SaltedHash;

my $csh = Crypt::SaltedHash->new(
          algorithm => 'SHA-1' );

my $cleartext = 'secret';

$csh->add( $cleartext );

my $salted = $csh->generate;

print „Salted: $salted\n";
```

Das Programm erzeugt z.B. folgende Ausgabe:

```
Salted:
  {SSHA}9GnzDgL3ChgeupyOkQtSrN/0/v8sGBf3
```

In den geschweiften Klammern steht eine Kennung für den verwendeten Algorithmus, gefolgt vom einem MIME Base 64 codiertem String, der wiederum die Verkettung des Hashes und des zufälligen Salts enthält.

Das zufällige `Salt`, welches zusammen mit dem Hash ausgegeben wird, kann mit der Methode `salt_hex()` einfach ausgelesen werden.

Stärkere Algorithmen, bitte!

Der gewünschte Hashingalgorithmus kann über den Parameter `algorithm` eingestellt werden - hier sind alle Algorithmen zulässig, die von *Digest* unterstützt werden, z.B. `SHA-256`, `SHA-384` oder `SHA-512`. Per Vorgabe wird `SHA-1 (FIPS 160-1)` verwendet.

Beispiel:

```
my $csh = Crypt::SaltedHash->new(
          algorithm => 'SHA-256' );
```

Das Beispielprogramm von oben erzeugt nun z.B. folgende Ausgabe:

```
Salted: {SSHA256}fqlu4iT+
  JI2MsDWAH7OtlFARRrKL8OibOawogcsezQs/v9Qg
```

Beispiel: Salted Hash validieren

Die Methode `validate()` kann einen vorgegebenen gesalzenen Hash validieren:

```
#!/usr/bin/perl
use strict;
use warnings;

use Crypt::SaltedHash;

my $csh = Crypt::SaltedHash->new(
           algorithm => 'SHA-1' );

my $cleartext = 'secret';

my $salted =
  '{SSHA}9GnzDgL3ChgeupyOkQtSrN/0/v8sGBf3';

my $valid = Crypt::SaltedHash->validate(
             $salted, $cleartext );

if ($valid) {
   print „OK\n";
} else {
   print „Not OK\n";
}
```

Crypt::SaltedHash kann natürlich auch die gesalzenen Hashes, die mit anderen Programmen, z.B. `slappasswd`, dem OpenLDAP-Passwort-Werkzeug, erzeugt wurden, validieren.

In Listing 1 habe ich das Klartextpasswort `123456` ein paar Mal durch `slappasswd` laufen lassen, um die unterschiedlichen gesalzenen Hashes für dasselbe Passwort zu erzeugen.

Das Programm erzeugt folgende Ausgabe:

```
OK ({SSHA}jppWV0DCxDJeOtaSS434nv5WewNYZSCS)
OK ({SSHA}ilJ95JAFKS4TgDbjRkcYrBMBRp+4mmCE)
OK ({SSHA}vrPil747oBHVgriDBbE+04XAs6BhXis0)
OK ({SSHA}ew3Xf1C9vK+H0kNeJ12Gc1M4fpbT41+x)
OK ({SSHA}9bRacd1WzrrUTiav7QBkMrxHtKTbjhpi)
OK ({SSHA}4m91OtOLhIfL+xDlfO/L5YdYaABKF60U)
OK ({SSHA}b+8sUhnuy6gnmlpf5BD58pww8NiQDZ3S)
OK ({SSHA}SeN6Yubt+pFOfTYDPPQ8JVp7MSfBkr1l)
OK ({SSHA}oSX0fmh8wNPGaJgEGQHY28RMiawsmraB)
OK ({SSHA}K4+dl7hft3VY7gfjPOdz8OOyXflbxA1f)
OK ({SSHA}7X/z6qeaDP6NpKQM3PYUQrERTTTj+VPD)
```

```
#!/usr/bin/perl
use strict;
use warnings;

use Crypt::SaltedHash;

my $csh = Crypt::SaltedHash->new(
           algorithm => 'SHA-1' );

#$ slappasswd -s 123456

my @salted = qw!
  {SSHA}jppWV0DCxDJeOtaSS434nv5WewNYZSCS
  {SSHA}ilJ95JAFKS4TgDbjRkcYrBMBRp+4mmCE
  {SSHA}vrPil747oBHVgriDBbE+04XAs6BhXis0
  {SSHA}ew3Xf1C9vK+H0kNeJ12Gc1M4fpbT41+x
  {SSHA}9bRacd1WzrrUTiav7QBkMrxHtKTbjhpi
  {SSHA}4m91OtOLhIfL+xDlfO/L5YdYaABKF60U
  {SSHA}b+8sUhnuy6gnmlpf5BD58pww8NiQDZ3S
  {SSHA}SeN6Yubt+pFOfTYDPPQ8JVp7MSfBkr1l
  {SSHA}oSX0fmh8wNPGaJgEGQHY28RMiawsmraB
  {SSHA}K4+dl7hft3VY7gfjPOdz8OOyXflbxA1f
  {SSHA}7X/z6qeaDP6NpKQM3PYUQrERTTTj+VPD
  !;

my $cleartext = '123456';

foreach my $salted (@salted) {

    my $valid = Crypt::SaltedHash->validate(
                 $salted, $cleartext );
    if ($valid) {
        print „OK ($salted)\n";
    } else {
        print „Not OK ($salted)\n";
    }
}
```

Listing 1

Sicherheitshinweis

Gesalzene Hashes sind, unabhängig vom gewählten Hashingalgorithmus, unter Sicherheitsapekten grundsätzlich genau so zu behandeln, wie Klartextpasswörter. Auch *Salted Hashes* können durch Brute-Force- oder Wörterbuchattacken angegriffen werden.