hashlib

520次阅读

摘要算法简介

Python的hashlib提供了常见的摘要算法，如MD5，SHA1等等。

什么是摘要算法呢？摘要算法又称哈希算法、散列算法。它通过一个函数，把任意长度的数据转换为一个长度固定的数据串（通常用16进制的字符串表示）。

举个例子，你写了一篇文章，内容是一个字符串'how to use python hashlib - by Michael'，并附上这篇文章的摘要是'2d73d4f15c0db7f5ecb321b6a65e5d6d'。如果有人篡改了你的文章，并发表为'how to use python hashlib - by Bob'，你可以一下子指出Bob篡改了你的文章，因为根据'how to use python hashlib - by Bob'计算出的摘要不同于原始文章的摘要。

可见，摘要算法就是通过摘要函数f()对任意长度的数据data计算出固定长度的摘要digest，目的是为了发现原始数据是否被人篡改过。

摘要算法之所以能指出数据是否被篡改过，就是因为摘要函数是一个单向函数，计算f(data)很容易，但通过digest反推data却非常困难。而且，对原始数据做一个bit的修改，都会导致计算出的摘要完全不同。

我们以常见的摘要算法MD5为例，计算出一个字符串的MD5值：

```
import hashlib

md5 = hashlib.md5()
md5.update('how to use md5 in python hashlib?')
print md5.hexdigest()
```

计算结果如下：

```
d26a53750bc40b38b65a520292f69306
```

如果数据量很大，可以分块多次调用update()，最后计算的结果是一样的：

```
md5 = hashlib.md5()
md5.update('how to use md5 in ')
md5.update('python hashlib?')
print md5.hexdigest()
```

试试改动一个字母，看看计算的结果是否完全不同。

MD5是最常见的摘要算法，速度很快，生成结果是固定的128 bit字节，通常用一个32位的16进制字符串表示。

另一种常见的摘要算法是SHA1，调用SHA1和调用MD5完全类似：

```
import hashlib

sha1 = hashlib.sha1()
sha1.update('how to use sha1 in ')
sha1.update('python hashlib?')
print sha1.hexdigest()
```

SHA1的结果是160 bit字节，通常用一个40位的16进制字符串表示。

比SHA1更安全的算法是SHA256和SHA512，不过越安全的算法越慢，而且摘要长度更长。

有没有可能两个不同的数据通过某个摘要算法得到了相同的摘要？完全有可能，因为任何摘要算法都是把无限多的数据集合映射到一个有限的集合中。这种情况称为碰撞，比如Bob试图根据你的摘要反推出一篇文章'how to learn hashlib in python - by Bob'，并且这篇文章的摘要恰好和你的文章完全一致，这种情况也并非不可能出现，但是非常非常困难。

摘要算法应用

摘要算法能应用到什么地方？举个常用例子：

任何允许用户登录的网站都会存储用户登录的用户名和口令。如何存储用户名和口令呢？方法是存到数据库表中：

```
name    | password
--------+----------
michael | 123456
bob     | abc999
alice   | alice2008
```

如果以明文保存用户口令，如果数据库泄露，所有用户的口令就落入黑客的手里。此外，网站运维人员是可以访问数据库的，也就是能获取到所有用户的口令。

正确的保存口令的方式是不存储用户的明文口令，而是存储用户口令的摘要，比如MD5：

```
username | password
---------+----------------------------------
michael  | e10adc3949ba59abbe56e057f20f883e
bob      | 878ef96e86145580c38c87f0410ad153
alice    | 99b1c2188db85afee403b1536010c2c9
```

当用户登录时，首先计算用户输入的明文口令的MD5，然后和数据库存储的MD5对比，如果一致，说明口令输入正确，如果不一致，口令肯定错误。

练习：根据用户输入的口令，计算出存储在数据库中的MD5口令：

```
def calc_md5(password):
    pass
```

存储MD5的好处是即使运维人员能访问数据库，也无法获知用户的明文口令。

练习：设计一个验证用户登录的函数，根据用户输入的口令是否正确，返回True或False：

```
db = {
    'michael': 'e10adc3949ba59abbe56e057f20f883e',
    'bob': '878ef96e86145580c38c87f0410ad153',
    'alice': '99b1c2188db85afee403b1536010c2c9'
}

def login(user, password):
    pass
```

采用MD5存储口令是否就一定安全呢？也不一定。假设你是一个黑客，已经拿到了存储MD5口令的数据库，如何通过MD5反推用户的明文口令呢？暴力破解费事费力，真正的黑客不会这么干。

考虑这么个情况，很多用户喜欢用123456，888888，password这些简单的口令，于是，黑客可以事先计算出这些常用口令的MD5值，得到一个反推表：

```
'e10adc3949ba59abbe56e057f20f883e': '123456'
'21218cca77804d2ba1922c33e0151105': '888888'
'5f4dcc3b5aa765d61d8327deb882cf99': 'password'
```

这样，无需破解，只需要对比数据库的MD5，黑客就获得了使用常用口令的用户账号。

对于用户来讲，当然不要使用过于简单的口令。但是，我们能否在程序设计上对简单口令加强保护呢？

由于常用口令的MD5值很容易被计算出来，所以，要确保存储的用户口令不是那些已经被计算出来的常用口令的MD5，这一方法通过对原始口令加一个复杂字符串来实现，俗称“加盐”：

```
def calc_md5(password):
    return get_md5(password + 'the-Salt')
```

经过Salt处理的MD5口令，只要Salt不被黑客知道，即使用户输入简单口令，也很难通过MD5反推明文口令。

但是如果有两个用户都使用了相同的简单口令比如123456，在数据库中，将存储两条相同的MD5值，这说明这两个用户的口令是一样的。有没有办法让使用相同口令的用户存储不同的MD5呢？

如果假定用户无法修改登录名，就可以通过把登录名作为Salt的一部分来计算MD5，从而实现相同口令的用户也存储不同的MD5。

练习：根据用户输入的登录名和口令模拟用户注册，计算更安全的MD5：

```
db = {}

def register(username, password):
    db[username] = get_md5(password + username + 'the-Salt')
```

然后，根据修改后的MD5算法实现用户登录的验证：

```
def login(username, password):
    pass
```

小结

摘要算法在很多地方都有广泛的应用。要注意摘要算法不是加密算法，不能用于加密（因为无法通过摘要反推明文），只能用于防篡改，但是它的单向计算特性决定了可以在不存储明文口令的情况下验证用户口令。
