CSS 337 Secure Systems

Mariana Chagoyan

|  |  |
| --- | --- |
| **HW#5: Encryption Tool -- Ex1** | Submit Assignment |

**1. Derive a Km (master key) of 256-bits/32 bytes size from a PBKDF2 key derivation function**

**a) Required inputs to the KDF:**

* **sha-512 hashing function of 256 bits/32 bytes size**
* **32 bytes of salt (generated from a cryptographically random source)**
* **Appropriate iteration count. (100,000 is suggested)**

**b) Screen shot of my program running**

**c) Tests your system to see How many can be done in under a second, and your password**

**d) Outputs**

**This implementation is provided in test file: pbkdf2-test1.py**

I encountered several problems with C++ and Java libraries for PBKDF2, I ended up writing this implementation in Python, which I am very new to it and don’t have as much experience. I modified the original version to ensure that it was fitting all the required criteria from above.

**Implementation Process Function and Variables**

**DK (generated derived key)**

**Km(Master key)**

**DK = Km**

* Km = PBKDF2 (Password, Salt, iterations, key Len, hashfunc)
* Password = master password from which a derived key is generated

(Function of two parameters with output length hLen (e.g. a keyed HMAC)

* Salt = sequence of bits, known as a cryptographic salt.
* Iterations = number of iterations desired
* Key Len = desired length of the derived key

# -\*- coding: utf-8 -\*-

"""

PBKDF2 IMPLEMENTATION FOR PYTHON.

"""

import hmac

import hashlib

import uuid

import random

from struct import Struct

from operator import xor

from itertools import izip, starmap

\_pack\_int = Struct('>I').pack

password = b'MaStErPaSsWoRd'

salt = uuid.uuid4().hex

iterations = 70000

hashfunc = hashlib.sha512

keylen = 16

**def pbkdf2\_hex(password, salt, iterations, keylen, hashfunc):**

"""Like :func:`pbkdf2\_bin` but returns a hex encoded string."""

return pbkdf2\_bin(password, salt, iterations, keylen, hashfunc).encode('hex')

**def pbkdf2\_bin(password, salt, iterations, keylen, hashfunc):**

"""Returns a binary digest for the PBKDF2 hash algorithm of `data`

with the given `salt`. It iterates `iterations` time and produces a

key of `keylen` bytes. By default SHA-2 is used as hash function,

a different hashlib `hashfunc` can be provided.

"""

**hashfunc = hashfunc or hashlib.sha512**

mac = hmac.new(password, None, hashfunc)

def \_pseudorandom(x, mac=mac):

h = mac.copy()

h.update(x)

return map(ord, h.digest())

buf = []

for block in xrange(1, -(-keylen // mac.digest\_size) + 1):

Km = u = \_pseudorandom(salt + \_pack\_int(block))

for i in xrange(iterations - 1):

u = \_pseudorandom(''.join(map(chr, u)))

Km = starmap(xor, izip(Km, u))

buf.extend(Km)

return ''.join(map(chr, buf))[:keylen]

def main():

**Km = pbkdf2\_hex(password, salt, iterations, keylen, hashfunc);**

print(" Master key Derivation from a PBKDF2 key derivation function:");

print ' Km = %s' % Km

print ' Password = %s' % password

print ' Salt = %s ' % salt

print ' Iterations = %d' %iterations

print ' keylen = %s ' % keylen

print(" (keylen of 16 contains 32 bytes which equals the first 256 bits of key.");

pass

if \_\_name\_\_ == '\_\_main\_\_':

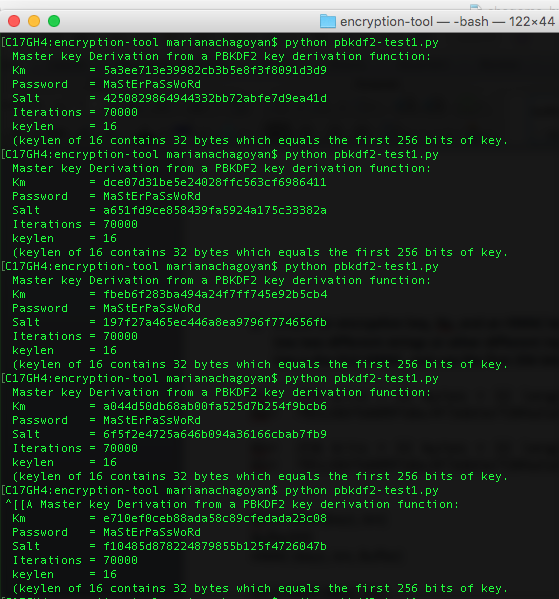
main()

**Reference:**

**https://github.com/fpgaminer/hash-phrase/blob/master/pbkdf2.py**

**b) Screen shot of my program running**

**Run #1a: commands >>python pbkdf2-test1.py**



**c) Tests your system to see How many can be done in under a second, and your password.**

None of the test could run under a second. I run test 5 times in a row and it took an average of 3.75 seconds. However this program for this part, it seems to break when doing 100,000 iteration counts and it gives this error: “segmentation code’. I spent several hours trying to find a similar implementation but apparently there is a bug that hasn’t been fixed. However, I run several tests and it seems to be working under from 1000 to 70000. I did my best to look for better code but ended up wasting too much time and not really solving this specific problem.

**d) Outputs**

Master key Derivation from a PBKDF2 key derivation function:

Km = 9e9ad78a042532ba58934be3be316ef

f703473e1f760aa9d4925bb557ea45263

Password= MaStErPaSsWoRd

Salt = e64702b9a3a44732b48e0aa7768ee9e0

Iterations = 70000

Key-length = 32