

Forum:[Hack Forums](#)

Thread: [Python One Time Pad Generator](#)

MiguelSanchez - April 8, 2022, 11:17 am

I wrote a key generator, which works via recursive SHA256 digests.

Using a starting stream it create a digest from your starting string and a salt, then recursively creates the next characters by using the SHA256digest itself plus the salt as the seed for a new SHA256 digest,

This occurs repeatedly until the desired size is reached, in this case 32MB.

I've attached a class, named distribution, which keeps track of how many times each byte occurs. Over a large testing period I can confirm it's non-repeating with a random distribution that equals out as the volume increases.

To create keys over 100MB I'd turn off the distribution.add_bytes on line 51.

To see the distribution just run Code outdist.graph_distribution() after the file has been made.

```
Code from hashlib import sha256
from time import time
class distribution:
    def __init__(self,size=256):
        self.size=size
        self.occurrences=[0 for i in range(size)]
        self.count=0
    def addbytes(self,bytestring):
        length=len(bytestring)
        while bytestring!=b"":
            self.occurrences[bytestring[0]]+=1
            bytestring=bytestring[1:]
        self.count+=length
    def addints(self,int_list):
        length=len(int_list)
        while int_list!=[]:
            self.occurrences[int_list[0]]+=1
            int_list=int_list[1:]
        self.count+=length
    def graph_distribution(self):
        from matplotlib import pyplot as plt
        plt.plot(self.occurrences)
        plt.show()
```

```
def RecursiveHash(startVal,salt,length=4096):
    ###Only for small keys,
    #the whole thing will be stored in the memory
    #for large keys make a key file
    Sha=sha256(salt+startVal)
    val=Sha.digest()
```

```

o=val
while len(o)<length:
    Sha.update(val+salt)
    val=Sha.digest()
    o+=val
return o[:length]

```

```

def
RecursiveHashtoFile(startVal,salt,filename="test.key",length=4096,distributioninst=distribution()):
    #length isn't exact, and for efficiency it will be rounded up to the nearest 32 bits
    Sha=sha256(salt+startVal)
    val=Sha.digest()
    fileout=open("Keys/"+filename,"wb")
    fileout.write(val)
    distributioninst.addbytes(val)
    n=32
    while n<length:
        Sha.update(val+salt)
        val=Sha.digest()
        distributioninst.addbytes(val)
        fileout.write(val)
        fileout.flush()
        n+=32
    return distributioninst

```

```

for i in [("KeyName",b"Seed goes here",b"My Salt"),
("Key2",b"Another Starting Seed/Password",b"My Salt")]:
    start=time()
    outdist=RecursiveHashtoFile(i[1],i[2],length=32*1024**2,filename="%s.key"%i[0])
    end=time()
    print("Done: %s seconds"%(end-start))

```