

## Forum: [Hack Forums](#)

Thread: [Python XOR Encryption. Fast, Low Memory](#)

MiguelSanchez - April 9, 2022, 2:49 am

I've been posting a lot of cryptokey generation stuff for one time pads lately.

For example:

<https://trogramming.com/blog/2022/03/24/cryptographic-key-generator/>  
& <https://hackforums.net/showthread.php?tid=6200705>

So it only made sense to actually make a one-time pad generator.

While I'm confident enough in my python skills to keep the memory from bloating with larger files, I still thought I'd have to port my algorithm to C++ to get anything meaningfully fast.

Luckily I was wrong, on my slow AF computer I can do about a 2 megabytes per second. I imagine most people will get much faster results.

As it's an XOR encryption you can decrypt the ciphertext back into plaintext with the exact same key.

For security reasons it's important you never reuse the same key thus the name one-time pad.

XOR\_Crypt is a standalone class, with no dependencies aside from the byte\_xor function which could possibly be sped up.

Included at the bottom is a demo of it's use, be sure to replace the filenames with your own files.

Code from time import time

```
def byte_xor(ba1, ba2):  
    return bytes([_a ^ _b for _a, _b in zip(ba1, ba2)])
```

```
class XOR_Crypt:  
    def __init__(self, keyfile):  
        self.keyfile = keyfile  
    def encrypt(self, bytestream):  
        CHUNKSIZE = 1024  
        out = b""  
        file = open(self.keyfile, 'rb')  
        while 1:  
            if len(bytestream) < CHUNKSIZE:  
                #last chunk  
                chunk = file.read(len(bytestream))  
                out += byte_xor(chunk, bytestream)  
                return out  
            else:  
                chunk = file.read(CHUNKSIZE)  
                out += byte_xor(chunk, bytestream)  
    def encryptfile(self, filename, outputname=None):  
        if outputname == None:  
            outputname = filename + "_encrypted"  
        ##Key File must be larger than the Input File
```

```

CHUNKSIZE=1024
file=open(self.keyfile,'rb')
infile=open(filename,'rb')
outfile=open(outputname,'wb')
while 1:
    #load file 1024 bytes at a time.
    bytestream=infile.read(CHUNKSIZE)
    if len(bytestream)<CHUNKSIZE:
        #last chunk
        chunk=file.read(len(bytestream))
        outfile.write(byte_xor(chunk,bytestream))
        outfile.flush()
        break
    else:
        chunk=file.read(CHUNKSIZE)
        outfile.write(byte_xor(chunk,bytestream))
        outfile.flush()

```

```

#Initialise the crypter class with your key. It must be larger than what you're encrypting.
a=XOR_Crypt("C:/Users/User/Documents/Hackforums/Keys/K001.key")
#record the current time
t=time()
#encrypt a file with the initialised crypter class
a.encryptfile("C:/Users/User/Documents/Hackforums/banner.png","outcrypt")
#use the same crypter to decrypt the encrypted file, it will return your original
a.encryptfile("outcrypt","decrypted.png")
#record the time upon completion
t2=time()
#print the total ellapsed time.
print("Took %s seconds" %round(t2-t,4))

```

Make duplicate USBs/SD cards with an offline computer and share the keys with people in person, or if long distance/international key exchange is required consider concealing a QR code.

Personally I use a deck of numbered barcodes that can be shuffled & resequenced then hashed into seeds for generating one time pads.

**unazed** - June 4, 2022, 11:09 pm

> For security reasons it's important you never reuse the same key thus the name one-time pad.

more like for security reasons don't use xor encryption. sure it's theoretically cryptographically impenetrable if you don't know the keyfile, and it's sufficiently large enough, but considering the fact that this is never the case, for various reasons, in the wild, it is absolutely useless.

**MiguelSanchez** - June 4, 2022, 11:43 pm

(Jun 4, 2022 11:09 PM)unazed Wrote: [\[CITE\]](#) > For security reasons it's important you never reuse the same key thus the name one-time pad.

more like for security reasons don't use xor encryption. sure it's theoretically cryptographically impenetrable if you don't know the keyfile, and it's sufficiently large enough, but considering the fact that this is never the case, for various reasons, in the wild, it is absolutely useless.

Why is this "never the case"? Perhaps you would like to elaborate on that extreme claim?

Any decimal approximation of a real number will produce infinite non-repeating non-terminating numbers, even in base 256, it becomes less predictable when you start a certain depth in.

Also fractals exist, and you could always create a deterministic method to interpolate with noise.

I agree XOR encryption isn't best practice, or even a good practice, but it was my first introduction to cryptography, 16 years ago now, and it's very nice way to get started in symmetric key encryption, it's what my teacher taught me first, and should serve well as an introduction to others.

I still believe you could absolutely incorporate XOR encryption practically, it just wouldn't be on its own, but as a sub-process in a larger algorithm.

I like to build my custom ciphers in a modular way so the methods I use for transposition and character substitution are developed and tested independently, then layer them in patterns, XOR usually has a place in there.

That said, for anything practical I don't use my own techniques, for now I put faith in libraries developed and bug-tested by large communities.

Welcome to the Forum

unazed - June 5, 2022, 1:14 pm

(Jun 4, 2022 11:43 PM)MiguelSanchez Wrote: [\[CITE\]](#) Why is this "never the case"? Perhaps you would like to elaborate on that extreme claim?

Any decimal approximation of a real number will produce infinite non-repeating non-terminating numbers, even in base 256, it becomes less predictable when you start a certain depth in.

Also fractals exist, and you could always create a deterministic method to interpolate with noise.

I agree XOR encryption isn't best practice, or even a good practice, but it was my first introduction to cryptography, 16 years ago now, and it's very nice way to get started in symmetric key encryption, it's what my teacher taught me first, and should serve well as an introduction to others.

I still believe you could absolutely incorporate XOR encryption practically, it just wouldn't be on its own, but as a sub-process in a larger algorithm.

I like to build my custom ciphers in a modular way so the methods I use for transposition and character substitution are developed and tested independently, then layer them in patterns, XOR usually has a place in there.

That said, for anything practical I don't use my own techniques, for now I put faith in libraries developed and bug-tested by large communities.

Welcome to the Forum

> Why is this "never the case"? Perhaps you would like to elaborate on that extreme claim?

because, like time has proven, when you leave up the security details (i.e. sourcing the randomness) to the developer, it leads to nothing but a trail of vulnerabilities.

ref. any major crypto vuln in the past 15 years

> Any decimal approximation of a real number will produce infinite non-repeating non-terminating numbers, even in base 256, it becomes less predictable when you start a certain depth in.

> Also fractals exist, and you could always create a deterministic method to interpolate with noise.

none of this made any sense, "decimal approximations" cannot be infinite nor non-terminating. real

numbers alone aren't necessarily infinite, nor non-repeating nor non-terminating. i assume you mean the subset of irrationals. nonetheless none of this added any more value to my response.

i'm guessing you're talking about sourcing randomness from irrational numbers, or more generally sequences of "infinite non-repeating non-terminating numbers", which is fine, as long as "when you start a certain depth in" your starting point is actually random, any numbers you then generate are however fully predictable.

> I agree XOR encryption isn't best practice, or even a good practice, but it was my first introduction to cryptography, 16 years ago now, and it's very nice way to get started in symmetric key encryption, it's what my teacher taught me first, and should serve well as an introduction to others.

i don't disagree on the fact it's a good starting point, XOR is a fundamental operation in many modern encryptions. my point was on its regular use as a standalone encryption

**Mr Roboto** - June 5, 2022, 1:16 pm

Great release! We were taught in entry Computer Science that XOR is theoretically impossible to crack through the use of One-Time pads (use once, key change).

**MiguelSanchez** - June 5, 2022, 10:47 pm

(Jun 5, 2022 01:14 PM)unazed Wrote: [CITE](#) none of this made any sense, "decimal approximations" cannot be infinite nor non-terminating.

Any decimal representation of real number is clearly finite and rational, obviously what I mean is that you're able to guaranteed to be able to produce more digits infinitely and they are guaranteed to be non-terminating and non-repeating to any depth.

Take `riemann_zeta_function(sqrt(2))` for example. Someone might be able to recognize it from its decimal form, but they're far less likely to recognize the pattern starting 100 digits in.

If you're worried they'll still know the pattern you can divide it into chunks (they needn't be equidistant), SHA256 them, and create a bytestream from concatenating the resulting hashes.

You're correct that you shouldn't use patterns in general, that's why `dev/random`'s entropy pool works the way it does, (no byte may be related to any of the others) but as long as you're creative with your "coordinates" there's no way someone is going to brute force your converging sum, assuming you chop the front off, or simply map its sections as seeds into a well proven PRNG.

If you'd like to demonstrate I'm happy to XOR encrypt something for you with an easy to guess irrational number OTP, but I'll chop the first X digits off and you try to decrypt the resulting file.

I'll pay you 500 HF bytes if you succeed plus I'll make it easier and ensure the encrypted data is untransposed plaintext, in English, with proper grammar (so you can anagram into a DCG)