Python One Time Pad Software

William M. Marcy, PhD, PE
Las Cruces, NM
Updated 10-20-2023.

These software programs written in Python implement one time pad (OTP) encipher and decipher of computer files. The software performs byte by byte XORs between a source file and a random number file to OTP encipher and decipher ***any arbitrary file.*** As a practical matter any file can be used as a one time pad to encipher any other file. The closer the OTP file is to being a truly random collection of bytes, the more secure the enciphered file will be. As you will learn below the XOR operation is completely reversible. You can use a JPG photo file as your OTP to encipher any other file. You can use a DOCX file to encipher another DOCX file, etc. These Python applications allow you to compute the entropy of any file of 8 bit bytes. A perfectly random file will have an entropy of exactly 8.0000 bits. It is not unusual for a large JPG photo file to have an entropy of 7.998+ bits. Photo files are very usable as OTP files. Large video files are very usable as OTP files. The possibilities are endless. Let's see how this works.

The XOR Process

If ***A*** is a source file and ***OTP*** is a random number file, the enciphering process works as follows:

Note: **^** is the XOR operator in Python

A ^ OTP *yields* ***B*** where ***B*** is the enciphered file. To recover file ***A*** if file ***B*** and the ***OTP*** file are known ***B ^ OTP*** *yields* ***A***. The operation is completely, perfectly reversible.

These programs written in Python take advantage of the fact that all computer files are saved as positive binary 8 bit bytes. With 8 bits per byte computer files consist of decimal numbers in the range of 0 – 255, actually represented as binary 00000000b to 11111111b.

The XOR binary operation proceeds as follows:

A	B	A ^ B
1	0	1
0	1	1
1	1	0
0	0	1

Example: (using 8 bit bytes)

If **A** = 01101110
and **OTP** = 11000011

A ^ OTP = **10101101** (B)
B ^ OTP = **01101110** (A)

A =	0	1	1	0	1	1	1	0
OTP =	1	1	0	0	0	0	1	1
A ^ OTP = B	1	0	1	0	1	1	0	1
B ^ OTP = A	0	1	1	0	1	1	1	0

1. OTP-Encipher-Decipher.py

This cryptographic utility XORs bytes sequentially from any arbitrary file with bytes from a random number one time pad (OTP) file using a cipher key. The OTP is treated as a circular random number list. The cipher key is the starting point in the OTP where the XOR operation begins. The cipher key can be any number between 0 and the length of the OTP -1 since in computer science we start counting bytes at 0.

If the OTP consists of 1 Giga byte of random 8 bit numbers, then the cipher key can be any number between 0 and 999,999,999. The cipher key makes deciphering files extremely difficult even if the OTP file used to perform the cipher is compromised. When the program's resulting XOR bytes are created they are saved in a user chosen file and directory. **This is the enciphered file** which can be sent to another person by any means, e.g., e-mail attachment, DropBox, iCloud, OneDrive, SDC card, etc.

These are the primary functions provided by ***OTP-Encipher-Decipher.py***

- The first function implements **A ^ OTP** : the results are saved to a user chosen directory and file. **This result is logically B, the enciphered file.**
- The second function implements **B ^ OTP** : the results are saved to a user chosen directory and file. **This result is logically A, the deciphered file.**
- The two functions together accomplish one-time-pad encipher and decipher. The Python programming involved is complex, but the idea is very simple. You do not need to know anything about Python programming to use this cryptographic program, but you can learn a lot if you are interested.

In addition to the use of a large random number OTP file, the process incorporates a cipher key which is the point in the OTP file where enciphering begins. The OTP is treated as a circular list. If the enciphering process reaches the end of the OTP before it finishes the process of XORing its bytes with the source file, it starts over at byte 0 of the OTP. It will do this as many times as it takes to complete the enciphering process.

If an OTP consists of 1 Giga bytes of random numbers (1,000,000,000), then the cipher key can be any value between 0 and 999,999,999 since we start numbering bytes at 0. To encipher a file, we choose an OTP file and a starting point (**cipher key)** and perform a byte by byte XOR. To decipher a file, we must

use exactly the same OTP and exactly the same starting point (cipher key). This implements synchronization between the OTP, the cipher key and the files being XORed.

Let's assume that the OTP is compromised, and a brute force attack is attempted to decipher a file. Assume there are 1 x e9 starting points in the OTP to try and let's assume that a "crib" of known bytes from the original plain text file is available to use in discovery of the starting point, i.e., the cipher key. If you are using a fast laptop, you can feasibly try about 1 x e3 (1,000) starting points (cipher keys) per second to match the "crib" bytes enciphered using a given starting point in the OTP with bytes in the enciphered file. If the starting point being used is correct the enciphered "crib" bytes will match the enciphered bytes at that place in the encrypted file. If they don't match add 1 to the place in the OTP file being used as a starting point, i.e., the cipher key, and encrypt the "crib" bytes again. It will take 1e6 seconds to get a match (worst case). Since there are 3600 seconds per hour and 86,400 seconds per day, it will take at most 11.5 days to find a match and thus the cipher key. On average it will take about half this long, 5.78 days.

Matching the bytes allows us to compute the true starting point (cipher key) used in the OTP. We know the enciphered file is exactly the same length as the original plain text file. We search for the point where the match occurred in the enciphered file. We subtract the point of the match in the enciphered file from the point in the OTP that enciphered the known bytes to obtain the true starting point. If the point in the OTP where the crib bytes were enciphered is 6780 and the point in the enciphered file where the match occurs is 2341 then the actual cipher key is 6780-2341 = 4439. Knowing the cipher key allows deciphering the entire enciphered file using the OTP..

Of course, the trial crib bytes used to find a point of match are not guaranteed to even exist in the original file. This could be a bad guess. If no match is found, then the hacker must find or guess some new crib bytes and start over. With an OTP of 10 Giga bytes, it will take an average of 57.8 days just to try one set of crib bytes. The key is the set of "crib" bytes known to be in the original plain text message. If you are attempting a brute force decipher of an enciphered photo or a video file, a set of crib bytes is very unlikely to be known. It is worth pointing out that supercomputers are at least 1000 times faster than a laptop computer. Even if the OTP is 10 Giga bytes, given a known set of "crib" bytes, it could be deciphered in about 0.057 days using a supercomputer.

As a point of historical interest this is the same method used at Bletchley Park in England during World War II to decipher the German Enigma codes. Amazingly this was all done by hand, no computers even existed.

If you have a top secret file you need to share, encrypt the file twice using two different OTP cipher codes and two different OTP files. Using this method having a set of known crib bytes is impossible. Even supercomputers will not be able to break the cipher in thousands of years of trying.

In terms of security the OTP process is theoretically unbreakable if the OTP is not known and truly random. The question is how random is random?

2. OTP-Random-File-Generator.py

The random number generator written in Python can produce very large random number files consisting of bytes with binary values between decimal 0 - 255 (00000000b – 11111111b). These files use a cryptographic quality random number generator called "os.urandom." The seed for the random number generator is generated from a real time computer-based time source. Every random number file is unique since the seed number cannot be reproduced.

3. OTP-Compute-Entropy.py

This program, written in Python, calculates the probability of each number 0-255 occurring in a file. This utility computes the entropy of a file of 8 bit bytes using ***sum (p*log2(p))*** for all values in the file. P is the probability that a given value occurs. Entropy is measured in bits. An 8 bit byte can store positive integers in the range 0-255. If all values in this range are equally likely to occur, then the entropy will be exactly 8.0000. The average value will be exactly 127.5. The closer the entropy value is to 8.0 the closer the values are to being equally likely. Equally likely is as random as it gets.

This is a very interesting utility since it can be applied to any computer file of 8-bit bytes. Large OTP files of cryptographic quality random numbers frequently have entropies in the range 7.99999+. These are very random numbers. It is informative to explore other files which often exhibit a surprisingly high degree of entropy (randomness). A 10 mb JPG photo file will often have an entropy in the range of 7.997+. A MP4 video file will frequent have entropy in the range 7.99997+. This means is that innocuous files can be pressed into service as an OTP.

Using the OTP-Random-File generator a random file of 999,999,999 bytes was created. The entropy utility computed the median value as 127.49911655 and the entropy in bits: 7.999999807984.

Correspondents can innocuously share high-resolution JPG photos for use as an OTP for encryption. They only need to share a secret cipher key to encipher and decipher files using the known photo as an OTP. In fact, they could share hundreds of photos and have an agreed upon method for choosing one on a given day for use as an OTP. They could also have an agreed upon method for choosing a starting point as well. This is a convenient way to share OTPs without drawing unwanted attention. The NASA Image of the day high resolution photos make great one time pads.

https://www.nasa.gov/multimedia/imagegallery/iotd.html

Just specify the row and column of the one you are using e.g., R2C3 on today's data.

4. The OTP-Binary-Segment- Utility

This utility allows you to segment a large existing random number file to produce a new OTP. Many large files can be segmented for use as an OTP. A downloaded video file is a good source. Videos are often gigabytes in size. Photos are typically many megabytes in size and can exhibit very high entropy. The **OTP-*Compute-Entropy*** program will allow you to assess their potential for use as a source of random bytes. Entropy values in the range of 7.97+ are perfectly adequate. A value of 8.0000 is perfect, but not

achievable in practice. The entropy of large OTP files included in this ZIP file are in the 7.998+ range. The 999,999,999 generated by the random number program and included in the ZIP file has the following entropy data: Median: 127.49911655, Entropy in bits: 7.999999807984.