How difficult Is It to "Hack" the Cipher Key of a One-Time-Pad?

William M. Marcy, PhD, PE
Las Cruces, NM
Updated 10-23-2021

Claude Shannon proved mathematically that if the *One-Time-Pad* (aka OTP) used to perform an XOR encryption is not available, then the enciphered file cannot be deciphered. Clearly, it is critical that OTP's be kept secret. Keeping secrets from bad actors is extremely difficult. *Dictionary.com defines a bad actor as a mean, ill-tempered, troublemaking, or evil person*. This definition can easily be extended to criminal organizations, rogue government agencies and hostile nation states. It is wise to assume that some bad actors have powerful computer resources and very talented people who work 24 hours per day to *"hack"* enciphered documents of all kinds.

Bad actors will resort to any means necessary to obtain OpenPGP public/private keys, OTP's, enciphered documents, cipher keys, passwords, and pass phrases. They send "phishing" e-mails to trick people into revealing their usernames and passwords. They hack into personal computers, smart phones, tablets, networks, and file servers. They exploit programming errors in software and operating systems to install "malware" to capture usernames and passwords to enable download of millions of enciphered computer files. They bribe and threaten IT personnel to gain access to networks and organizational file servers. They physically steal unsecured laptops, smart phones, and tablets. They market public/private keys, OTP's, cipher keys and deciphered documents daily on the "dark web" for millions of dollars.

Once a public/private key pair is obtained by bad actors everything they obtain that was enciphered previously with the public key can be quickly deciphered with the corresponding secret key. Tens of thousands of documents may have been enciphered with a single public/private key pair. Once enciphered files and public/private keys are in the hands of bad actors every file can be deciphered. There is no recovery. Bad actors can send false messages in the key owner's name. Changing a public/private key pair only protects future documents. Revoking a public/private key pair can be extremely difficult.

OTP encrypted documents are more difficult for bad actors to exploit than OpenPGP documents. Once bad actors have obtained one or more OTP's they still face a significant hurdle in deciphering OTP enciphered documents without the cipher key used for encryption.

Even if the same OTP is used to encipher more than one document, if the cipher key used is unique for each document, knowing the cipher key for one document does not compromise other documents. An analogy is a room full of safes. Knowing the combination to one safe does not open other safes if they all have unique combinations**.**

An *OTP encrypted file is analogous to a safe's physical container. The cipher key is analogous to the combination for that container. Finding the cipher key used with each OTP to encipher each document is paramount for hackers to succeed. The minute a document comes out of a safe it is unprotected. An enciphered document can be moved place to place and still be protected. It is as though the document takes its safe with it.*

The Hacker's Dilemma

For purposes of this analysis, we will assume that the OTP used to XOR encipher a valuable document is obtained by a bad actor as well as the resulting enciphered file **but not the cipher key that was used.** How hard is it to use just the OTP and the enciphered file to recover the original document if enough computer power is available?

A Brute Force Attack

An obvious method is to use a "brute force" attack. Start with a cipher key of 0 in the OTP and XOR the enciphered file with the OTP starting at this position. Analyze the resulting file to see if it is recognized as the original file. If it is not, increment the cipher key by one and repeat the XOR process starting at the next position. If this is done for every possible value of the cipher key, XORing the OTP with the enciphered file will eventually produce the original file. It is just a matter of time.

Let's assume that the OTP is 2,000,000,000 (2e9) bytes in size. The worst case scenario is having to perform the XOR operation 2e9 times and be able to recognize the resulting file as the original file. The problem of automatically recognizing the original file when it results is nontrivial and may be difficult if not impossible to program.

If the original file was comprised of Unicode characters for a known language this might be feasible to program. If the file was a standard format photo file, word file, spread sheet file, compressed file, or video file it might be more challenging. If it was a file of compiled executable computer code this might be extremely difficult to program. No doubt sophisticated bad actors could use AI methods to recognize the resulting file as the original file. XORing the OTP and the enciphered file to produce a trial "deciphered" file can be done very quickly.

Recognizing the correct file deciphered file following each XOR trial decryption can be very time consuming.

A Direct Method for Recognizing the Original File

A direct method is to obtain (or guess) a fragment of the original document that is known to be enciphered using the OTP that has been compromised. Many enciphered files have content that is known or can be guessed such as headings, signature blocks, file identifiers, dates, address blocks, etc. Knowing that the original file type was a text, jpg, docx, PDF, file, etc. can also provide a fragment known to be enciphered using the compromised OTP.

Having the compromised OTP file and the corresponding enciphered file, we start with a specified position in the OTP (the default is 0) and XOR the OTP bytes starting at this position with the known fragment bytes (also called a “crib”) producing a short, enciphered segment that is contained in the enciphered file if the correct starting position (cypher key) in the OTP is used. Now we must search for a match.

We start our search for a match at byte zero of the enciphered file. If a match is not found, we advance one byte in the enciphered file continue the search. We continue to do this until we have scanned the entire enciphered file. If no match is found, we advance one byte in the OTP and encipher the fragment again. We repeat the scan of the enciphered file looking for a match starting at byte zero. Eventually we will find a position in the OTP where the resulting enciphered fragment bytes match bytes at a position in the enciphered file.

We know that the enciphered file and the original file are exactly the same size. The position in the enciphered file where the match occurs is the same as the position in the original file where the known fragment occurred. For example, if the match occurs at position 1297 in the enciphered file and we are using position 3176 in the OTP file then the cipher key must be 1297 bytes before this position in the OTP. The cipher key is directly computed as 3176-1297 = 1879. We now use the cipher key 1879, the compromised OTP and the enciphered file to decipher the

entire file. **How Fast Is This Process?**

As an experiment we implemented this algorithm in the Python program ***OTP-Hacker-Tool.py.*** To keep things manageable, we used a 2e6 OTP and a 56 kB original file to produce a 56kB enciphered file using a chosen (arbitrary) cipher key. Using the 2e6 OTP and the 56kB enciphered file, a 20 character known fragment of the original file and an Intel NUC personal computer.

Device name	Intel-NUC
Processor	13th Gen Intel(R) Core(TM) i7-1360P 2.20 GHz
Installed RAM	32.0 GB (31.6 GB usable)
System type	64-bit operating system, x64-based processor

We found that we can test about 6 trial OTP cipher key positions per second. The worst case of 2e6 trials (2,000,000) at a rate of 6 per second would require 3.33e5 seconds. At 3600 seconds per hour this will require 92.5 hours or 3.85 days (worst case) to find the cipher key. On average we might expect to find the cipher key quicker, in about half that time.

A 2e6 OTP is rather small. Multi-gigabyte size OTP files are easy to create or obtain. Entropy measurements of high resolution 12 megabyte jpg photo files indicate entropies on the order of 7.9977 or higher. 8.0000 is perfect with every byte value from 00000000 to 11111111 being equally likely. 2 gigabyte video files have similar entropy. The Python program **OTP-C*ompute-Entropy.py*** can be used to measure the entropy of any 8 bit file.

These are very acceptable files for use as OTP files. Changing to a 2e9 size OTP result in taking 3,800 days (worst case) to find the cipher key. Even half that time, 1900 days, still requires more than 5 years effort to recover the cipher key and requires knowing an included fragment. This is clearly not feasible with a desktop computer. We need a faster computer.

A Google search of the internet to get comparisons of laptop speeds with supercomputer speeds indicates that the best supercomputers in 2020 (e.g., the top two were two IBM-built supercomputers, Summit and Sierra, installed at the Department of Energy's Oak Ridge National Laboratory (ORNL) in Tennessee and Lawrence Livermore National Laboratory in California) are estimated to be 1e6 times faster than the Intel NUC desktop we used to conduct these experiments.

Bad actors, especially nation states, will have access to a supercomputer that is 1e6 times faster than an Intel NUC computer. A supercomputer will find the cipher key a million times faster, in about 3.8 milliseconds. Bad actors with supercomputers will have access to AI algorithms that can recognize an original file without knowing an included fragment. When you are dealing with bad actors with supercomputers given that they have both the compromised OTP and the enciphered file, they will obtain the original file in fractions of a second.

How to Make Bad Actors with Super Computers Struggle?

We can even make bad actors with supercomputers struggle. We encrypt the valuable file twice using two different OTPs and two different randomly chosen cipher keys.   Bad actors will not know the order in which the OTP's were used. They will only have the final enciphered file plus the two compromised OTP files. The result of the first trial decryption will be what appears to be a file of random bytes. Even advanced AI algorithms will not know if it is the original file or not. There are no shortcuts. If they guessed the wrong order the brute force attack will fail. They will need to use the compromised OTP's in both orders.

OK, let's assume they guessed the correct order to use the OTP's. Their first step is to use the OTP file that was used ***second*** to do a trial decipher of the enciphered file starting with cipher key-2 (default position of zero). This is an attempt to obtain the enciphered file resulting from the first encryption that used the first OTP starting with cipher key-1 (default position zero). Note that the result of this first step will be a file of random bytes and AI will not be able to determine if it is the one needed for step-2 or not. The second step is deciphering this intermediate result using cipher key-1 to produce what is potentially the original file. Now they can use AI to analyze the result of step-2 to see if the original file is recognized. If the original file does not emerge, advance cipher key-2 by 1 and repeat step-1 then repeat step-2. Continue this process until all values for key-1 and key-2 have been exhausted or the original file is produced.

If each OTP is 2e9 bytes then the bad actors will potentially need to repeat this process 2e9 times with cipher key-2 and for each value of cipher key-2, 2e9 times for cipher-key-1. The worst case scenario is that 4e18 trials will be required to decipher the file.   Using a supercomputer, they would be able to conduct about 1e6 trials per second. This would require 4e12 seconds or about 1.33e9 hours. **At 8,760 hours per year this is approximately 1.5e5 years, 150,000 years. This is safe enough!**

What About Bad actors with limited budgets?

Bit Coin Miners regularly use banks of thousands of computers to solve block chain equations to earn *Bit Coins.* Our experiments with an Intel NUC computer indicate that about 6 trials per second are possible. With a 2e9 OTP it potentially would take 3.33e8 seconds worst case to hack a single encrypted file. Dividing by 3600 this is 9.24e4 hours. Dividing by 24 this is 259 days. That seems pretty safe.

These same bitcoin mining computers could just as easily be used to take a 2e9 OTP and divide it into 2000 segments, one assigned to each computer. Each computer operates in parallel looking for the cipher key. Instead of 259 days using one laptop the cipher key can be found in 1.29 days or about 0.64 days on average. This doesn't look so safe anymore.

Each bitcoin mining computer draws about 130 watts of power. 2000 computers draw about 260 kilowatts per hour. Electricity in Las Cruces costs about $0.13 per kilowatt hour. Each hour to run 2000 computers costs about $33.8. This is $811.2 per day. It would cost about $1040 to decipher one document (worst case). This would be about $520 on average. The enciphered document would need to be valuable enough to cover the electricity bill to operate a bank of 2000 computers. If the document contains 500,000 valid credit card numbers, names, addresses and CVC numbers it would be worth much more than the $520-$1040 to hack. One sale on the dark web could be worth many times that amount. The same deciphered document would be sold hundreds of times. The bad actors would likely earn millions of dollars for their efforts

The bottom line is this:

1. Without the OTP, bad actors using a single desktop class computer can never decipher your enciphered files.
2. With the OTP and your enciphered file bad actors cannot feasibly decipher your files in any reasonable amount of time using a desktop class computer.
3. Using a bank of several hundred computers (e.g., Bit Coin miners) bad actors can decipher your files in about half a day if they have the OTP and the enciphered file.
4. Bad actors with super computers can decipher your files in milliseconds if they have the OTP and your enciphered file.
5. If your files are enciphered twice with 2 OTP's, even bad actors with super computers cannot decipher your files in any feasible amount of time even if they have both OTP's and your enciphered file.

To keep your valuable documents safe follow these rules.

1. Make your OTP's as hard to obtain as possible. Keep all of your OTP software and associated files on a "bit locker" encrypted external USB drive or Secure Digital Card (SDC). Set the options in the OTP-Encipher-Decipher.py so the settings files will not have the names of the OTP, cipher or pass phrase used. Remove the USB drive or SDC card when not needed. Lock it up or hide it where it is very unlikely to be found. If it is encrypted, even if found, you are still protected.

2. Use galleries of image files or video files as OTP’s. Use a PDF of a book as an OTP.

Establish an agreed upon method for choosing an OTP and a pass phrase or cipher key.

If practical, use a different OTP for every file encrypted. This rule is not absolute if you follow Rule 4.

3. Use a different, pass phrase generated cipher key for every file.

4. If you encipher a file twice using two different OTP’s and a pass phrase generated cipher key for each, even bad actors with supercomputers will not be able to decipher your files in any feasible period of time. There is a second PDF document that explains in detail how to use the OTP-Encipher-Decipher tool to perform double encryption.

5. Don’t send pass phrases or cipher keys by clear text e-mail. Use ProtonMail or PGP encrypted e-mail. You can even use a public source of pass phrases such as headlines on news articles.

An Example of a Brute Force Hacker Tool

The *OTP-hacker_tool.py* program implements a brute force “hacking” algorithm using a known fragment of the source file. It measures performance for a laptop class computer in terms of iterations per second of the algorithm. Use it for your own experiments.