7 Lenticrypt: a Provably Plausibly Deniable Cryptosystem; or, This Picture of Cats is Also a Picture of Dogs

by Evan Sultanik

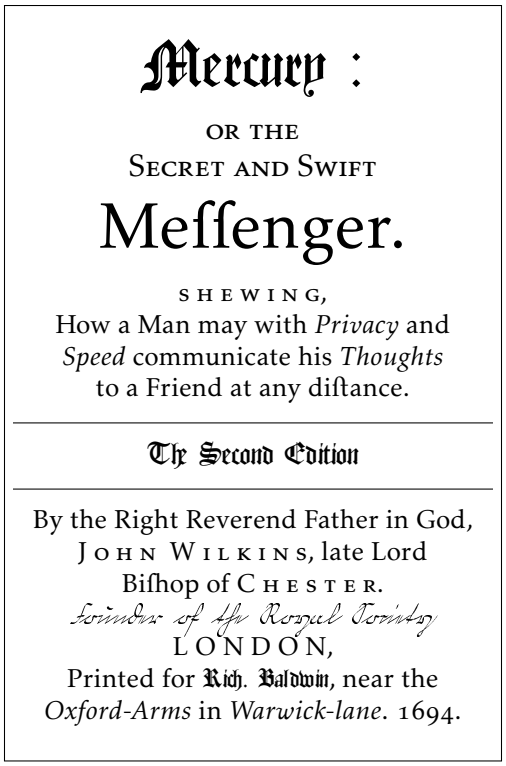
Deniable cryptosystems allow their users to plausibly deny the existence of the plaintext content of their encrypted data. There are many existing technologies for accomplishing this (*e.g.*, TrueCrypt), which usually accomplish it by having multiple separate encrypted volumes in the ciphertext that will decrypt to different plaintexts depending on which decryption key is used. Key k_1 will decrypt to innocuous volume v_1 whereas key k_2 will decrypt to high-value volume v_2. If an adversary forces you to reveal your secret key, you can simply reveal k_1 which will decrypt to v_1: the innocuous volume full of back-issues of PoC||GTFO and pictures of cats. On the other hand, if the adversary somehow detects the existence of the high-value volume v_2 and furthermore gains access to its plaintext, the jig is up and you can no longer plausibly deny its contents' existence. This is a serious limitation, since the high-value plaintext might be incriminating.

An *ideal* deniable cryptosystem would allow the creator of the ciphertext to plausibly deny having created the plaintext *regardless* of whether the true high-value plaintext is revealed. The obvious use-case is for transmitting illegal content: Alice wants to encrypt and send her neighbor Bob a pirated copy of the ColecoVision game *George Plimpton's Video Falconry*. She doesn't much care if the plaintext is revealed, however, she *does* want to have a plausible *legal* argument in the event that she is prosecuted whereby she can deny having sent that particular file, *even if* the high-value file is revealed. In the case of systems like TrueCrypt, she can't really deny having created the alternate hidden volume containing the video game since the odds of it just randomly occurring there *and* a key happening to be able to decrypt it are astronomically small. But what if, using our supposed "ideal" cryptosystem, she *could* plausibly claim that the existence of the video game was due to pure random chance? It turns out that's possible, and we have the PoC to prove it!

Before we get to the details, let's first dispel the apparent nefariousness of this concept by discussing some more legitimate use-cases. For example, we could encrypt a high-value document such that it decrypts to either a redacted or unredacted version depending on the key. If the recipients are not aware that they have unique keys, one could deliver what *appears* to be a single encrypted message to multiple recipients with individualized content. The individualization of the content could also be very subtle, allowing it to be used as a unique watermark to identify the original source of a leaked document: a so-called "canary trap." Finally, "deep-inspection" filters could be evaded by encrypting an innocuous payload with a common, guessable password.

7.1 Running Key Ciphers

A running key cipher is one of the most basic cryptosystems, yet, if used properly, it can be one of the most secure. Being avid PoC||GTFO readers, Alice and Bob both have a penchant for treatises with needlessly verbose titles that are edited by Right Reverend Doctors. Therefore, for their secret key they choose to use a copy of a seminal work on cryptography by the Rt. Revd. Dr. Lord Bishop John Wilkins FRS.


Mercury :

OR THE

SECRET AND SWIFT

Meſſenger.

SHEWING,

How a Man may with *Privacy* and
Speed communicate his *Thoughts*
to a Friend at any diſtance.

The Second Edition

By the Right Reverend Father in God,
JOHN WILKINS, late Lord
Biſhop of CHESTER.
Founder of the Royal Society
LONDON,
Printed for Rich. Baldwin, near the
Oxford-Arms in *Warwick-lane*. 1694.

They have agreed to start their running key on the first line of the book, which reads:

> “Every rational creature, being of an imperfect and dependant Happineſs, if therefore naturally endowed with an Ability to communicate its own Thoughts and Intentionſ ; that ſo by mutual Services, it might better promote it ſelf in the Proſecution of its own Well-being.”

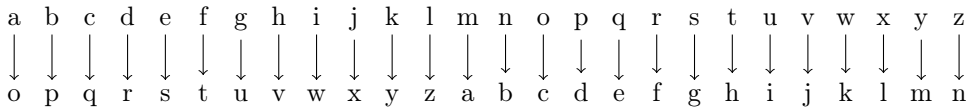
The encryption algorithm is then very simple: Each character from the running key is used as a rotation to permute the associated character of the plaintext. For example, say that the first character of our plaintext is “A”; we would take the first character of our running key, “E”, look up its numerical index in the alphabet, and rotate the plaintext by that much to produce the ciphertext.

```
PLAINTEXT:   AN ADDRESS TO THE SECRET SOCIETY OF POC OR GTFO...
RUNNING KEY: EV ERYRATI ON ALC REATUR EBEINGO FA NIM PE RFEC...
CIPHERTEXT:  EI EUBIELA HB TSG JICKYK WPGQRZM TF CWO DV XYJQ...
```

There are of course many other ways the plaintext could be combined with the running key, another common choice being XORing the bits. If the running key is truly random then the result will almost always be what is called a “one-time pad” and will have perfect secrecy. Of course, my expository example is nowhere near secure since I preserved whitespace and used a running key that is nowhere near random. But, in practice, this type of cryptosystem can be made very secure if implemented properly.

7.2 Book Ciphers

Perhaps the *most* basic type of cryptosystem—one that we’ve all likely independently discovered in our early childhood—is the substitution cipher: Each letter in the alphabet is statically mapped to another. The most common substitution cipher is ROT13, in which the letters of the alphabet are rotated 13 steps.



In fact, we can think of the running key cipher we described above as a sort of substitution cipher in which the alphabet mapping changes for each byte based off of the key.

Book Ciphers marry some of the ideas of substitution ciphers and running key ciphers. First, Alice and Bob decide on a shared secret, much like the book they chose as a running key above. The shared secret needs to have enough entropy in order to have at least one instance of every possible byte in the plaintext. For each byte in the shared secret, they create a lookup table mapping all 256 possible bytes to lists containing all indexes (*i.e.*, file offsets) of the occurrences of that byte in the secret:

```
with open(secret_key_file) as s:
    indexes = dict([(b, []) for b in range(256)])
    for i, b in enumerate(map(ord,s.read())):
        indexes[b].append(i)
```

Then, for each byte encountered in the plaintext, the ciphertext is simply the index of an equivalent byte in the secret key:

```
def encrypt(plaintext, indexes):
    for b in map(ord, plaintext):
        print random.choice(indexes[b]),
```

To decrypt the ciphertext, we simply look up the byte at the specified index in the secret key:

```
def decrypt(ciphertext, secret_key_file):
    with open(secret_key_file) as s:
        for index in map(int, ciphertext.split()):
            s.seek(index)
            sys.stdout.write(s.read(1))
```

In effect, what is happening is that Alice opens her book (the secret key), finds indexes of characters that match the characters she has in her plaintext, writes those indexes down as her ciphertext, and sends it to Bob. When Bob receives the ciphertext, he opens up his identical copy of the book, and for each index he simply looks up the letter in the book and writes that down the letter into the decrypted plaintext. There are various optimizations that can be made, *vi&.*, using variable-length codes within the key similar to LZ77 compression (*e.g.*, using words from the book instead of individual characters).

7.3 Lenticular Book Ciphers

In the previous section, I showed how a book cipher can be used to encrypt plaintext p_1 to ciphertext c using secret key k_1. In order for this to be useful as a plausibly deniable cryptosystem, we will need to ensure that given some *other* secret key k_2, the *same* ciphertext c will decrypt to a totally different plaintext p_2. In this section I'll discuss an extension to the book cipher which achieves just that. I call it a "Lenticular Book Cipher," inspired by the optical device that can present different images to the viewer depending on the lens that is used. I was unable to find any description of this type of cryptosystem in the literature, likely because it is very naïve and practically useless ... except for in the context of our specific motivating scenarios!

Given a set of plaintexts $P = \{p_1, p_2, \ldots, p_n\}$ and a set of keys $K = \{k_1, k_2, \ldots, k_n\}$, we want to find a ciphertext c such that `decrypt(`c`,`k_i`)` $\mapsto p_i$ for all i from 1 to n. To accomplish this, let's consider an individual byte within each of the plaintexts in P. Let $p_i[j]$ represent the j^{th} byte of plaintext i. Similarly, let's define $k_i[j]$ and $c[j]$ to refer to the j^{th} byte of a key or the ciphertext. In order to encrypt the first byte

of all of the plaintexts, we need to find an index m such that $k_i[m] = p_i[0]$ for i from 1 to n. In general, $c[\ell]$ can be any unsigned integer m such that

$$\forall i \in 1, \ldots, n : k_i[m] = p_i[\ell].$$

We can relatively efficiently find such an m by modifying the way we build the `indexes` lookup table:

```
def build_index(secret_keys):
    indexes = {}
    for i, key_bytes in enumerate(zip(*secret_keys)):
        key_bytes = tuple(map(ord, key_bytes))
        if key_bytes not in indexes:
            indexes[key_bytes] = [i]
        else:
            indexes[key_bytes].append(i)
    return indexes
```

Encryption then happens similarly to the regular book ciper:

```
def encrypt(plaintexts, secret_keys):
    indexes = build_index(secret_keys)
    for text_bytes in zip(*plaintexts):
        text_bytes = tuple(map(ord, text_bytes))
        print random.choice(indexes[text_bytes]),
```

Decryption is identical to the regular book cipher.

So, in fewer than twenty lines of Python, we have coded a PoC of a cryptosystem that allows us to do the following:

```
encrypt([open("plaintext1").read(), open("plaintext2").read()],
        [open("key1").read(), open("key2").read()])
```

If we pipe STDOUT to the file "`cipher.enc`", we can decrypt it as follows:

```
with open("cipher.enc") as enc:
    decrypt(enc.read(), "key1") # This will print out plaintext1
    decrypt(enc.read(), "key2") # This will print out plaintext2
```

There do seem to be a number of limitations to this cryptosystem, though. For example, what keys should Alice use? The keys need to be long enough such that every possible combination of bytes that appears across the plaintexts will occur in `indexes`; the length of the keys will need to increase exponentially with respect to the number of plaintexts being encrypted. Fortunately, in practice, you're not likely to ever need to encrypt more than a few plaintexts into a single ciphertext. One possible source of publicly available keys to use would be YouTube videos: Alice could simply download a video and use its raw byte stream as the key. Then all she needs to do is communicate the name of or link to the video to Bill off-the-record.

I have created a complete and functional implementation of this cryptosystem, including some optimizations (*e.g.*, variable block length, compression, length checksums, error checking, *&c.*). It is available here:

`https://github.com/ESultanik/lenticrypt`

7.4 Proving a Cat is Always Also a Dog

So far, I've gone through a lot of trouble to describe a cryptosystem of dubious information security[9] whose apparent functionality is already available from tools like TrueCrypt. In this section I will make a

[9]While I do have a few letters after my name that suggest I know a thing or two about Computer Science, cryptography is not my specific area of specialization.

mathematical argument that provides what I believe to be a legal basis for the plausible deniability provided by lenticular book ciphers, enabling its use in our motivating scenarios.

Laws and contracts aren't interpreted like computer programs; legal decisions are often dictated less by the defendant's actions than by his or her *intent*. In other words, if it appears that Alice *intended* to send Bob a copy of Video Falconry, she will be found guilty of piracy, regardless of how she conveyed the software.

But what if Alice legitimately only knew that key k_1 decrypted c to a picture of cats, and didn't know of its nefarious use to produce a copy of Video Falconry from k_2? How likely would it be for k_2 to produce Video Falconry simply by coincidence?

For sake of this analysis, let's assume that the keys are documents written in English. For example, books from Project Gutenberg could be used as keys. I am also going to assume that each character in a document is an independent random variable. This is a rather unrealistic assumption, but we shall see that the asymptotic properties of the problem make the issue moot. (This assumption could be relaxed by instead applying Lovász's local lemma[10].)

First, let's tackle the problem of figuring out the probability that `decrypt(`c`,`k_2`)` $\mapsto p_2$ completely by chance. Let n be the length of the documents in characters and let $m < n$ be the minimum required length of a string for that text to be considered a copyright violation (*i.e.*, outside of fair use). The probability that `decrypt(`c`,`k_2`)` contains no substrings of length at least m from p_2 is

$$(1 - q^m)^{(n-m+1)},$$

where q is the probability that a pair of characters is equal. Here we have to take into account letter frequency in English. Using a table from Wikipedia[11], I calculate q to be roughly 6.5 percent (it's the sum of squares of the values in the table). According to Google, there are about 130 million books that have ever been written[12]. Let's be conservative and say that two million of them are in English. Therefore, the probability that *at least one pair* of those books will produce a copyrighted passage from c is

$$1 - \left((1 - q^m)^{(n-m+1)}\right)^{\binom{2000000}{2}},$$

which is extremely close to 100% for all $m < n \ll 2000000$.

Therefore, for any ciphertext c produced by a lenticular book cipher, it is almost certain that there exists a pair of books one can choose that will cause a copyright violation! Even though we don't know what those books might be, they must exist!

Proving that this is a valid *legal* argument—one that would hold up in a court of law—is left as an exercise to the reader.

[10] Paul Erdős and László Lovász. *Problems and results on 3-chromatic hypergraphs and some related questions.* Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to Paul Erdős on his 60th birthday), Volume II, North-Holland, Amsterdam, 1975, pp. 609–627. Colloq. Math. Soc. János Bolyai, Volume 10.

[11] `http://en.wikipedia.org/wiki/Letter_frequency#Relative_frequencies_of_letters_in_the_English_language`

[12] Leonid Taycher. *Books of the world, stand up and be counted! All 129,864,880 of you.* August 5, 2010. `http://booksearch.blogspot.com/2010/08/books-of-world-stand-up-and-be-counted.html` Retrieved March 21, 2014.