One Time Pad Example and Brute Force Attacks

By Steven Gordon on Sun, 18/11/2012 - 11:43am

The One Time Pad is covered in my course on Security and Cryptography [3] at SIIT. Below are some examples using crypto [4]. See also my other examples of classical ciphers and letter frequency analysis [5].

The One Time Pad [6] is the only known cipher that is unconditionally secure. That means if a malicious user obtains a ciphertext created with the One Time Pad, then there is no way in which they can determine the corresponding plaintext or key. Since the key is random and the same length as the plaintext, there is no information in the ciphertext (such as the frequency of letters) that the malicious user can use to discover the plaintext/key. Also, even if a brute force attack could be applied, where the malicious user decrypts the ciphertext will all possible keys, the malicious user would no way in knowing which plaintext is the original plaintext. This is because a brute force attack will produce many potential plaintexts that make sense to the malicious user. In the following I try to demonstrate these concepts with some examples.

Here is an example plaintext:

```
$ cat plaintext4.txt
theinternetisaglobalsystemofinterconnectedcomputernetworksthatusethestandardinternetp
```

There are 69 words and 371 letters in the plaintext, an average of about 5.4 letters per word. The most frequent letters in this plaintext are `t`, `e` and `o`, each making up about 10% of all letters, i.e. occurrring much more frequently than most of the other letters:

```
$ crypto count letters plaintext4.txt percentsort
370
 10.54 t
 10.54 e
 10.00 o
  7.57 n
  7.30 s
  7.03 i
  7.03 a
  6.49 l
  5.95 r
  4.86 c
  2.97 p
  2.97 d
  2.16 w
  2.16 u
  2.16 h
```

The most frequent digrams are `te`, `et` and `ne`:

```
$ crypto count digrams plaintext4.txt percentsort
369
  2.71 te
  2.71 et
  2.44 ne
  2.17 of
  1.90 er
  1.90 al
  1.63 wo
  1.63 th
  1.63 or
  1.63 on
  1.63 in
  1.63 ca
  1.36 tw
  1.36 st
  1.36 rk
```

I've generated a random key of 371 letters:

```
$ crypto random 371
xbqjjxiwyiowtijdkjiqmvtpofjxbwdwlmivwzltdzalrzwgreiripbzolvnnmubozfjuufstaunqhmtshihw
```

and encrypt the plaintext with that key using the One Time Pad (using the `crypto` program I used the Vigenere cipher, which is identical to the One Time Pad if a random key as long as the plaintext is used):

```
$ crypto vigenere enc theinternetisaglobalsystemofinterconnectedcomputernetworksthatu
qiurwqmnlmhelipoykibetlisrxcjjwacowijdnmhcczdoqzvvvvblpqydounfotssmnmnffwalqyufxjumal
```

Now lets look at the statistics of the ciphertext.

```
$ crypto count letters ciphertext4.txt percentsort
370
  5.68 i
  5.41 a
  5.14 t
  4.86 y
  4.86 j
  4.59 w
  4.59 q
  4.59 m
  4.59 l
  4.32 n
  4.32 d
  3.78 o
  3.51 k
  3.51 h
  3.51 c

$ crypto count digrams ciphertext4.txt percentsort
369
  1.08 oq
  1.08 mh
  0.81 xa
  0.81 wi
```

```
0.81 wa
0.81 vv
0.81 td
0.81 ta
0.81 nm
0.81 mn
0.81 ma
0.81 gi
0.54 zy
0.54 yw
0.54 yk
```

The most frequent letters are `i`, `a` and `t`. But note the percentages. Most of the letters occur with *about* the same frequency. Ideally, with 26 possible letters, each letter would be 3.85% of the total. In the example some letters occur more frequently than this because the plaintext is short: with a longer plaintext, you will see the frequency of all letters approaching 3.85%. Comparing the letter (and digram) frequencies in the plaintext and ciphertext, the distribution in the ciphertext is much more even, whereas in the plaintext some letters/diagrams occur much more frequently than others. This is one illustration of how the One Time Pad takes a plaintext with some structure and produces a ciphertext which appears random (no structure). Without any structure in the ciphertext, a malicious user has no way to determine the key or plaintext from the ciphertext.

But what if a malicious user could perform a brute force attack, trying all possible keys? Can they find the original plaintext?

First lets consider how many possible keys exist. The key in the example is 371 characters, where each character can take any of 26 possible values. Hence the number of possible different keys is 26^{371} or about 10^{525}. This is far too many for a practical brute force attack. Assuming the malicious user has access to 1 billion computers that can each decrypt at a rate of 10^{18} ciphertexts per second, then it would take 10^{490} years to try all keys.

But what if the malicious user could perform a brute force attack: would they find the original plaintext? A successful brute force attack requires the malicious user to be able to recoginise the plaintext. This is possible if all but one of the plaintext values make sense, e.g. one is an English phrase, while all others are random characters. With the brute force attack there are 10^{525} possible plaintexts.

Lets calculate the number of possible English plaintexts that may make sense. The Oxford English Dictionary has about 200,000 words. To create an English plaintext that makes sense, the words are combined. Lets assume they can be combined in any order. So to create a plaintext with about 70 words, then the 1st word can be any one of 200,000, the 2nd word can be any one of 200,000, and so on until the 70th word can be chosen as any one of 200,000. That is, there are $200{,}000^{70}$ combinations of words. (Of course the resulting English plaintext will not make sense if allowing any order, so in fact the possible number of English plaintexts that make sense is much less).

So the possible English plaintexts that make sense is $200{,}000^{70}$ or about 10^{371} (note the exponent of 371 has nothing to do with the number of characters in the plaintext - 371. Its just a coincidence that in this example the exponent and the number of characters are the same value). So when the malicious user looks at all 10^{525} plaintext values generated by trying all keys, they will find 10^{371} that make sense. How does the malicious user know which of these 10^{371} plaintext values was the original plaintext? They cannot. (If they guessed, then they have 10^{-371} chance of being correct, i.e. no chance). Hence even if a malicious user could try all possible keys, they still cannot find the original plaintext. The One Time Pad is secure in all conditions.

Brute Force Attack on a Monoalphabetic Cipher

Would a brute force attack work on a monoalphabetic cipher? There are 26! or about 10^{26} possible keys in such a cipher. Assuming a malicious user had enough computing power to try them all, could they then identify the original plaintext? As calculated above, there are about 10^{371} English plaintexts that could make sense. And there are 10^{525} possible plaintexts that have 371 characters.

Of course one of the plaintext values makes sense (it was the origianl plaintext). What is the chance that of the 10^{26} plaintext values generated, that at least one more makes sense (in English)? The percentage of plaintexts that make sense is $(10^{371})/(10^{525}) = 10^{-154}$. Multiple that by 10^{26} to find the percentage of generated plaintexts from the brute force attack that make sense in English: 10^{-128}. So the chance that another generated plaintext makes sense in English is 10^{-128}, or effectively 0. Hence a brute force attack could be successful against a monoalphabetic cipher.

Find two or more keys for my example ciphertext that produce meaningful plaintext is not easy. To see an example, see the textbook by Stallings [7] or my lecture notes on Classical Encryption Techniques [8] (specifically slide 27, but note there is an error in the textbook and my old lecture notes: the first three letters of key 2 should be `pft`, not `mfu`).

In summary, the One Time Pad is unconditionally secure because: (a) there is no information in the ciphertext that a malicious user can use to determine the plaintext or key, i.e. the ciphertext appears random; and (b) a brute force attack would not be successful because the malicious user would not be able to determine which of the generated plaintext values was the original (as many would make sense).

Content: Articles [9]
Topic: Security [10]

Source URL: http://sandilands.info/sgordon/one-time-pad-example-brute-force-attacks

Links:
[1] http://sandilands.info/sgordon/one-time-pad-example-brute-force-attacks
[2] http://sandilands.info/sgordon/user/2
[3] http://ict.siit.tu.ac.th/~sgordon/css322/
[4] http://sandilands.info/sgordon/doc/security/crypto.txt
[5] http://sandilands.info/sgordon/classical-ciphers-frequency-analysis-examples
[6] http://en.wikipedia.org/wiki/One-time_pad
[7] http://ict.siit.tu.ac.th/~sgordon/resources/study.html#stallings_crypt
[8] http://ict.siit.tu.ac.th/~sgordon/css322/lectures.html
[9] http://sandilands.info/sgordon/taxonomy/term/144
[10] http://sandilands.info/sgordon/taxonomy/term/116