CSCI3381-Cryptography

Lecture 2: Classical Cryptosystems

January 17, 2017

This describes some cryptographic systems in use long before the advent of computers. All of these methods are quite insecure, from the modern standpoint, but they illustrate some important principles.

1 Caesar Cipher (Shift Cipher)

This method was attributed to Julius Caesar (1st century BC) by Suetonius in *Lives of the Twelve Caesars* (2nd century AD). As crude as it is, the basic idea of a shift cipher reappears in the Vigenère Cipher and the one-time pad that we discuss further on, and (with a 26-letter alphabet replaced by a 2-letter alphabet) in modern stream ciphers.

Each letter of the alphabet is identified with a number in the set $\{0, 1, \ldots, 25\}$. So A is 0, B is 1, *etc.* The key is also an integer $k \in \{0, 1, \ldots, 25\}$. The encryption algorithm proceeds letter by letter, replacing each letter j of the plaintext by $j + k \mod 26$. The decryption algorithm is the same, using $-k \mod 26$ in place of k.
Example. The plaintext "ATTACK AT DAWN" is identified with the sequence of numbers

$$0, 19, 19, 0, 2, 10, 0, 19, 3, 0, 22, 13$$

(In this and the next two examples, we eliminate the spacing between words.) The encryption with key $k = 9$ is then

$$9, 2, 2, 9, 11, 19, 9, 2, 12, 9, 5, 22$$

which is transmitted as

$$JCCJLTJCMJFW.$$

The recipient, to decrypt, turns this back into the corresponding numerical sequence and then adds $-9 \bmod 26 = 17$ to decrypt.

Here is a formal description: The key space is

$$\mathcal{K} = \{A, B, \ldots, Z\} = \{0, \ldots, 25\},$$

and the message space $\mathcal{M}$ is the set of finite sequences of elements of $\mathcal{K}$, which we typically denote by $\mathcal{K}^*$. If

$$m = m_1 m_2 \cdots m_r \in \mathcal{M},$$

then

$$E(k, m) = c_1 c_2 \cdots c_r,$$

where for each $1 \leq i \leq r$,

$$c_i = (m_i + k) \bmod 26.$$

Decryption is the same as encryption, with the encryption key replaced by its additive inverse:

$$D_k = E_{-k \bmod 26}.$$

Security. If an attacker suspects that this method was used, the cipher is broken (without a computer!) by a brute-force ciphertext-only attack, since there are only 26 keys, and in all likelihood only one of the 26 decryptions will make sense.

While this attack is very easy to carry out by hand, the question of how to do it with a computer is an interesting one. It's trivial to compute all 26 decryptions, but how do we pick out the *right* one without human supervision? We'll return to this point in a moment.

1.1 Binary variant

Computers use the two-letter alphabet $\{0, 1\}$ rather than a 26-letter alphabet, and the shift operation is just the *exclusive-or* operation $\oplus$ on bits:

$$0 \oplus 0 = 1 \oplus 1 = 0, 0 \oplus 1 = 1 \oplus 0 = 1.$$

The exclusive-or operation is extended in a natural fashion to sequences of m bits, by performing the operation bit by bit. For example,

$$00101 \oplus 10111 = 10010.$$

The standard ASCII representation of a character is a single byte. Python has the nice property of allowing you to treat *any* byte as a character, even if it does not correspond to a printable character. Thus we can use a single character k as a one-byte shift value, and encrypt a sequence

$$p_1 p_2 \cdots p_r$$

of plaintext bytes as

$$c_1 c_2 \cdots c_r,$$

where $c_i = p_i \oplus k$. The nice thing here is that encryption is exactly the same as decryption, with no modification of the key. Here is an example. Suppose the plaintext is 'Attack at dawn.' (Here we allow spaces and punctuation in the string.) Let the key be '&'. As bytes written in hexadecimal the plaintext is

```
41 74 74 61 63 6b 20 61 74 20 64 61 77 6e 2e
```

and the key is 26. So, for example, the first byte of the ciphertext is

$$41 \oplus 26 = 01000001 \oplus 00100110 = 01100111,$$

or 67 in hex, 'g' in ASCII. If you try to represent the entire ciphertext as a Python string it will appear as

```
gRRGEM\x06GR\x06BGQH\x08
```

since Python cannot display the bytes with hex values 06 and 08 as printable characters.

Here the size of the keyspace is 256 rather than 26, but of course that is still very small, and the system is broken manually by a brute-force attack.

2 Monoalphabetic Substitution Ciphers

The Caesar cipher is a special instance of a cryptographic system that you see printed in newspapers and on puzzle sites as 'Cryptogram puzzles'. Here is a typical example, taken from one such site:

```
SAM QOBRAGVSLGOS SJPU AGO QVSGMES
OAM DVO OCTTMLGEN TLJK V OQPGS
QMLOJEVPGSB.  OAM LMQPGMU, "QVLS JT
KM VNLMMO DGSA BJC. HCS SAM JSAML
QVLS JT KM DJCPU PGXM V OMRJEU
JQGEGJE."
```

The idea is that each ciphertext letter represents a fixed plaintext letter. For instance, in this puzzle, 'S' is the encryption of 'T' and 'R' is the encryption of 'C'. (I'll leave it to you to solve the rest of the puzzle and recover the plaintext, which is a dumb joke.) Providing the spacing between words and the punctuation makes the puzzle quite simple; it's a lot more interesting if you give it in the form

```
SAMQO BRAGV SLGOS SJPUA GOQVS GMESO
AMDVO OCTTM LGENT LJKVO QPGSQ MLOJE
VPGSB OAMLM QPGMU QVLSJ TKMVN LMMOD
GSABJ CHCSS AMJSA MLQVL SJTKM DJCPU
PGXMV OMRJE UJQGE GJE
```

Here the message space $\mathcal{M}$ is again the set of all finite sequences of letters, but the key space $\mathcal{K}$ is the set of all *permutations*, that is, all one-to-one onto functions

$$\pi : \{A, B, \ldots, Z\} \to \{A, B, \ldots, Z\}.$$

The encryption of a sequence

$$p_1 \cdots p_m \in \mathcal{M}$$

of plaintext letters is

$$c_1 \cdots c_m,$$

where for each i, $c_i = \pi(p_i)$. Decryption is given by

$$D(\pi, m) = E(\pi^{-1}, m),$$

where π^{-1} is the inverse permutation of π. Observe that the Caesar cipher is just monoalphabetic substitution with a very restricted set of keys.

Security and cryptanalysis The number of keys is the number of permutations of a 26-element set, namely

$$26! \approx 4.03 \times 10^{26}.$$

A brute-force attack is not feasible on a key space this large. But this kind of puzzle can be solved by hand. How is this possible?

There are two serious weaknesses: First, the frequency distribution of letters in English is very uneven, and this same uneven distribution is present in the ciphertext. (Figure 1.)

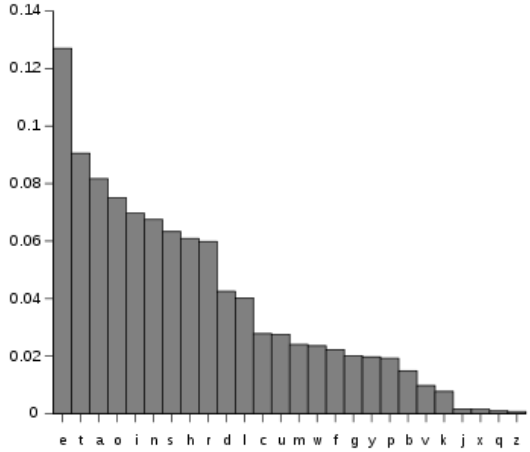

Figure 1: *Relative letter frequencies in English. The six most frequently occurring letters ETAOIN account for about 44% of all letters in a large corpus.*

Thus in a long ciphertext, one might well guess that the most frequently occurring character is the encryption of one of the more commonly-occurring letters, like E,T, A, most likely of E. Similarly, the pairs and triples of successive letters that occur give away a lot of information: The triple 'PKD' should probably never occur in the plaintext, even across word boundaries; the pair 'TH' is very common.

Second, each character of ciphertext depends on only one character of plaintext and one character of the key. Thus when one part of the key is guessed correctly, it is possible to build on this progress to get other parts of the key. In better systems, changing any piece of the key or the plaintext should completely change the ciphertext.

3 Statistical Analysis

We return to the question raised earlier about automatically selecting the correct decryption from the 26 candidate plaintexts produced by the brute force attack on the Caesar cipher. We show a statistical method for doing this, which we will also use in analyzing the Vigenère cipher below.

Consider the following experiment—we have two bins of letters, one containing all the letters from a long English text, and the other containing the letters

from a Caesar-encryption of the original text. We choose one letter from each bin. What is the probability that the two letters are the same?

If α is a letter, then we will denote by P_α the probability that a randomly-selected English letter is α. For example, P_e gives the largest value, about 0.121, while P_z gives the smallest value 0.0002. Let's suppose that the second bin was made by encrypting with a shift of 21, so that 'a' is the encryption of 'f', 'b' the encryption of 'g', *etc.* The probability that both letters are 'a' is then $P_a P_f$, that both are 'b' is $P_b P_g$, and, since these are disjoint events, the probability that the two letters are the same is the sum

$$P_a P_f + P_b P_g + P_c P_h + \cdots P_z P_e.$$

If the two bins are identical (*i.e.*, if the shift is zero), then the probability that the two letters are the same is

$$P_a^2 + P_b^2 + \cdots + P_z^2.$$

It is not too hard to prove that the second value is larger than the value that we would get with any nonzero shift. In fact, it is a *lot* larger: We can use tabulated values for the frequency of letters in large texts (like those used to produce the graph in the figure) and compute the first sum for different shift values:

```
 0 0.06441384
1 0.03943454
2 0.03078133
3 0.03429628
4 0.04417932
5 0.03421635
6 0.03645208
7 0.03886936
8 0.03449249
9 0.03358449
10 0.03854469
11 0.04354422
12 0.03899343
13 0.04120904
14 0.03899343
15 0.04354422
16 0.03854469
```

```
17 0.03358449
18 0.03449249
19 0.03886936
20 0.03645208
21 0.03421635
22 0.04417932
23 0.03429628
24 0.03078133
25 0.03943454
```

With no shift, the sum is 0.064, but for nonzero shifts, the sum is never larger than 0.044. This gives us a method for distinguishing original plaintext from Caesar-encrypted plaintext: We count the number f_α of occurrences of the letter α in the text, and compute

$$P_a f_a + \cdots P_z f_z.$$

If this is real English text, then the sum will be approximately $0.064n$, where n is the length of the text, but for encrypted text it should not be much larger than $0.044n$. Thus if we want to find out which of the 26 possible decryptions of our ciphertext is the right one, we just have to compute this sum for each of the candidate plaintexts and take the one that gives the largest value.

4 Vigenère Cipher (polyalphabetic substitution cipher)

This dates to the 16th century. (The actual inventor was named Bellaso; Vigenère invented a related system, but the misattribution has stuck.) It was widely thought to be essentially unbreakable by commentators writing as late as the early 20th century, but successful attacks were discovered by researchers in the 19th century.

Instead of using a single shift value, as in the Caesar cipher, the Vigenère cipher uses different shift values for different letters of the plaintext. The key is a string typically between 5 and 10 letters long. The shift values are given by the usual integer encodings (0 for A, 1 for B, etc.) of the key characters.

Example. Here the key is

PIGSTY

and the plaintext is

LAUNCH THE ATTACK AT DAWN ON MONDAY UNDER CLEAR SKIES. HOLD YOUR POSITION IN LIGHT RAIN. RETREAT IN HEAVY RAIN. SEND ADVANCE SCOUTING PARTIES TO VERIFY ESCAPE ROUTES.

The table below shows the encryption of the first 15 characters. This is just addition of each column mod 26, using the integer encodings of the characters.

L	A	U	N	C	H	T	H	E	A	T	T	A	C	K
P	I	G	S	T	Y	P	I	G	S	T	Y	P	I	G
A	I	A	F	V	F	I	P	K	S	M	R	P	K	Q

Decryption is done in exactly the same way, subtracting the shift value instead of adding it.

Formal Description Here $\mathcal{M}$ is again the set of all finite sequences of letters, while

$$\mathcal{K} = \{0, 1, \ldots, 25\}^k,$$

i.e., sequences of k letters, where k is the length of the key. Encryption is given by

$$E(s_0 s_1 \cdots s_{k-1}, p_0 p_2 \cdots p_{m-1}) = c_0 \cdots c_{m-1}$$

with

$$c_i = (p_i + s_{i \bmod k}) \bmod 26.$$

Decryption is the same, with $s_0, \ldots, s_{k-1}$ replaced by $-s_0 \bmod 26, \ldots, -s_{k-1} \bmod 26$.

Security and cryptanalysis. Using several different shift values smooths out the uneven distribution that we get from using a single shift or a single substitution alphabet, so the frequency distribution of letters in the ciphertext is much more uniform. Another advantage over the monoalphabetic substitution was that the short key could be easily memorized.

With a key length of 9, the number of keys is about 5.4×10^{12}. Even with computers, checking every key is a stretch, but can be carried out. In the 19th century environment, of course, a brute-force attack was absolutely out of the question.

Cryptanalysis proceeds in two phases: The first phase determines the length of the key, and the second recovers the key itself. For the first step, align the ciphertext with itself advanced i characters for $i = 2, 3, \ldots, 9$ characters (as high as the key length is likely to be) and count the number of matches for each alignment. The table below shows the alignment for the first few characters of ciphertext in our example, with $i = 4, 5$.

A	I	A	F	V	F	I	P	K	S	M	R	P	K	Q
				A	I	A	F	V	F	I	P	K	S	M
					A	I	A	F	V	F	I	P	K	S

There are three matches for $i = 5$ in this example). If the amount we advanced the text is a multiple of the key length, then the probability that a pair of aligned letters matches is

$$P_a^2 + P_b^2 + \cdots + P_z^2.$$

If the amount is not a multiple of the key length, then we are essentially repeating the experiment of drawing one letter from each of two distributions with different Caesar shifts, and as we saw above, the probability of a match is significantly smaller. Thus the advance that gives the largest number of matches is probably the correct key length.

In the second step we use the information about the key length (let us suppose it is 6) to divide the text into 6 subtexts: the first subtext consists of the 1st, 7th, 13th, etc. characters, the next consists of the 2nd, 8th, 14th, etc., characters, and so on. Each of these subtexts represents a selection of English characters encrypted with a *single shift*. It remains to find the shift associated with each subtext. Once again, we compute the values

$$(f_a, f_b, \ldots, f_z),$$

giving the number of occurrences of $a, b, c, etc.$. These should be roughly proportional to the probabilities $P_a, P_b, \ldots,$ but shifted: for instance, we would have $(f_a, f_b, \ldots, f_z)$ is approximately a multiple of $(P_y, P_z, P_a, \ldots, P_w, P_x)$ if the shift value was 2. We use an estimate of the probabilities $P_a, P_b, \ldots, P_z$ and compute each of the sums

$$f_a P_a + f_b P_b + \cdots + f_z P_z,$$

$$f_a P_b + f_b P_c + \cdots + f_z P_a,$$

$$\cdots$$

$$f_a P_z + f_b P_a + \cdots + f_z P_y.$$

Again our inequality suggests that the largest of these gives the correct shift for each subtext, and thus reveals the key. An example is worked out in detail in the Python notebook posted on the course website.