Weighted and randomized polyalphabetic substitution ciphers to avoid language detection through frequency analysis.

Abstract

At each step in any encryption or decryption process, substitutions are being made to modify a message. In a very abstract way, if you squint and look from a far enough distance, you could imagine every encryption or decryption process as a composition of many different substitution ciphers. Furthermore, all substitution ciphers share the vulnerability of frequency analysis. As far as I know, frequency analysis is always performed at some point in the process of message cracking, and although a message might not be immediately vulnerable, it becomes so as soon as an attacker can predict the likelihoods of particular substitutions. So, a cipher that is resistant to frequency analysis could be applied to a multiple encryption process, to cryptographically strengthen it, or at least make one part of cracking it harder.

Description

I would like to propose a specific form of polyalphabetic substitution cipher, which can make frequency analysis much more difficult in one of two ways: either by matching the relative frequencies in the cyphertext to English closely but incorrectly, or by making every relative frequency in the cyphertext equal. The former prevents substitutions from being recognized in frequency analysis, but only when applied to English plaintext, while the latter can be applied to any plaintext, but is a weaker cypher on its own. Both can act as a preliminary step in stronger encryption systems, and neither should be used as a standalone encryption system.

In the first cypher, the number of possible substitutions for each letter is its relative frequency times ten, rounded when necessary. So before shuffling the key, “a” will have the possible substitutions “a1 a2 a3 a4 a5 a6 a7 a8”, “b” will have “b1”, “c” will have “c1 c2”, and so on for a total of 100 possible substitutions. After shuffling the key, Alice must exchange it with Bob before sending him a message. Alice and Bob may also use a shorter key to seed a pseudorandom number generator that shuffles the identity key for the first form of cypher, or they may just transmit a key securely with a separate three-pass key exchange method

Implementation

A program written in java has been placed in `cas3-106` in the directory `/home/students/apaterson/Project`. It can be executed with `java PolyCypher.java`. The program will either generate a random key, regenerate a key from a seed passed with the `-s` flag, or use a full key passed with the `-k` flag. It can encrypt plaintext passed with the `-e` flag,

and decrypt cyphertext passed with the `-d` flag. For example, this is the key generated using the pseudorandom seed “Herscovici”:

a	8	a1 a2 a3 a4 a5 a6 a7 a8	n	7	n1 n2 n3 n4 n5 n6 n7	a	o4 s3 o2 y1 n3 e5 e2 b1	n	a7 r2 h1 n5 s2 n2 l2
b	1	b1	o	7	o1 o2 o3 o4 o5 o6 o7	b	n7	o	v1 t7 h6 r3 c2 o6 r5
c	2	c1 c2	p	2	p1 p2	c	a5 e6	p	e8 o1
d	4	d1 d2 d3 d4	q	1	q1	d	s6 x1 r7 j1	q	p2
e	9	e1 e2 e3 e4 e5 e6 e7 e8 e9	r	8	r1 r2 r3 r4 r5 r6 r7 r8	e	q1 z1 a3 o5 f2 a6 i7 h4 h5	r	i8 h2 i5 g2 a1 l3 e1 h3
f	2	f1 f2	s	6	s1 s2 s3 s4 s5 s6	f	t2 d3	s	y2 e3 s1 t6 p1 w2
g	2	g1 g2	t	8	t1 t2 t3 t4 t5 t6 t7 t8	g	n1 l1	t	d2 w1 m1 u3 g1 r4 n6 o3
h	6	h1 h2 h3 h4 h5 h6	u	3	u1 u2 u3	h	r6 w3 t4 d1 t5 u2	u	t8 a8 i6
i	8	i1 i2 i3 i4 i5 i6 i7 i8	v	1	v1	i	c1 e7 r1 k1 i3 m2 a4 t3	v	r8
j	1	j1	w	3	w1 w2 w3	j	a2	w	f1 d4 e4
k	1	k1	x	1	x1	k	s5	x	n4
l	4	l1 l2 l3 l4	y	2	y1 y2	l	o7 i1 i2 e9	y	s4 u1
m	2	m1 m2	z	1	z1	m	i4 l4	z	t1

Figure 1: The unshuffled weighted key (in white) and the shuffled weighted key (in gray).

```
java PolyCypher.java -s Herscovici
```

Now, we simply use the key as we would with a typical substitution cypher, with the caveat that we must randomly chose one of several characters to replace the more frequent letters. For example, this is a list of possible cyphertexts given the plaintext “theaeontheaeontheaeon”:

“W1U2O5S3H5O6A7W1W3H5S3F2O6N2R4T5A3Y1Z1R5S2”
“M1R6F2E5Q1O6N2M1T4H5S3Q1H6H1G1T5A6O4O5H6H1”
“R4R6F2O4Q1H6L2R4R6O5B1A6V1S2R4W3Q1O4H4R5R2”
“O3T5A6O2H5C2N5D2T4F2S3Q1C2N2G1D1F2Y1H5R3A7”
“D2T5A6O4A3R5A7W1T5O5S3H4H6N5O3W3H5N3H5T7S2”
“theaeontheaeontheaeon”
“theaeontheaeontheaeon”
“theaeontheaeontheaeon”
“theaeontheaeontheaeon”
“theaeontheaeontheaeon”

Figure 2: Differing cyphertext (in white) and matching plaintext (in gray).

```
java PolyCypher.java -s Herscovici -e theaeontheaeontheaeon
java PolyCypher.java -s Herscovici -d <in white>
```

In this case, I chose a plaintext with the most frequently used English letters, to show that their the relative frequencies no longer correspond to relative frequencies in the cyphertext. The opposite effect will be seen in a longer English plaintext.

Attack

Like the process of Alice and Bob using the substitution cypher, the process of Eve attacking the cyphertext with a frequency analysis seems familiar. In fact, after encryption the

frequencies of letters will have barely changed from their original values. We can see from the key and the cyphertext that the highest frequency letters have been randomly substituted for several lower frequency letters, making the cypher appear to the analysis as very weakly encrypted, or even unencrypted, sample. Obviously this analysis does not include the numbers, but they could be substituted with letters as well, numbers are just easier to read and test.

Frequencies in file	Frequencies in English	Frequencies in file	Frequencies in English	Frequencies in file	Frequencies in English	Frequencies in file	Frequencies in English
I: 0.111	e: 0.127	R: 0.030	m: 0.024	E: 0.120	e: 0.127	M: 0.030	m: 0.024
E: 0.103	t: 0.091	M: 0.030	w: 0.023	A: 0.102	t: 0.091	U: 0.019	w: 0.023
T: 0.085	a: 0.082	Y: 0.022	f: 0.022	I: 0.094	a: 0.082	Q: 0.019	f: 0.022
A: 0.081	o: 0.075	G: 0.015	y: 0.020	O: 0.086	o: 0.075	K: 0.015	y: 0.020
S: 0.070	i: 0.070	F: 0.011	g: 0.020	N: 0.071	i: 0.070	G: 0.015	g: 0.020
P: 0.066	n: 0.067	D: 0.011	p: 0.019	T: 0.064	n: 0.067	D: 0.015	p: 0.019
L: 0.066	s: 0.063	X: 0.007	b: 0.015	S: 0.056	s: 0.063	P: 0.011	b: 0.015
H: 0.055	h: 0.061	W: 0.004	v: 0.010	H: 0.053	h: 0.061	B: 0.011	v: 0.010
O: 0.048	r: 0.060	V: 0.004	k: 0.008	W: 0.045	r: 0.060	V: 0.008	k: 0.008
N: 0.048	d: 0.043	K: 0.004	j: 0.002	R: 0.045	d: 0.043	F: 0.008	j: 0.002
C: 0.044	l: 0.040	Z: 0.000	z: 0.001	Y: 0.038	l: 0.040	J: 0.004	z: 0.001
B: 0.044	u: 0.028	Q: 0.000	x: 0.001	C: 0.038	u: 0.028	Z: 0.000	x: 0.001
U: 0.041	c: 0.028	J: 0.000	q: 0.001	L: 0.034	c: 0.028	X: 0.000	q: 0.001
I: 0.120	e: 0.127	U: 0.026	m: 0.024	O: 0.117	e: 0.127	G: 0.026	m: 0.024
E: 0.105	t: 0.091	M: 0.026	w: 0.023	E: 0.117	t: 0.091	M: 0.023	w: 0.023
N: 0.102	a: 0.082	Y: 0.015	f: 0.022	N: 0.102	a: 0.082	D: 0.019	f: 0.022
O: 0.090	o: 0.075	Z: 0.011	y: 0.020	A: 0.094	o: 0.075	C: 0.019	y: 0.020
A: 0.083	i: 0.070	G: 0.011	g: 0.020	I: 0.090	i: 0.070	F: 0.015	g: 0.020
T: 0.071	n: 0.067	F: 0.011	p: 0.019	T: 0.071	n: 0.067	Q: 0.011	p: 0.019
R: 0.056	s: 0.063	V: 0.008	b: 0.015	R: 0.056	s: 0.063	P: 0.011	b: 0.015
S: 0.049	h: 0.061	Q: 0.008	v: 0.010	U: 0.049	h: 0.061	Z: 0.008	v: 0.010
D: 0.049	r: 0.060	K: 0.008	k: 0.008	H: 0.038	r: 0.060	V: 0.008	k: 0.008
H: 0.045	d: 0.043	X: 0.004	j: 0.002	L: 0.034	d: 0.043	K: 0.004	j: 0.002
W: 0.038	l: 0.040	P: 0.004	z: 0.001	Y: 0.030	l: 0.040	B: 0.004	z: 0.001
L: 0.030	u: 0.028	J: 0.000	x: 0.001	W: 0.030	u: 0.028	X: 0.000	x: 0.001
C: 0.030	c: 0.028	B: 0.000	q: 0.001	S: 0.026	c: 0.028	J: 0.000	q: 0.001

Figure 3: Relative frequencies of short English plaintext (gray) and possible cyphertext (white) remain mostly unchanged.

```
frequencies cyphertext
java PolyCypher.java -s Herscovici -e "$(cat plaintext)" > cyphertext
java PolyCypher.java -s Herscovici -d "$(cat cyphertext)"
```

Instead of a traditional frequency analysis, we might instead assume that the frequency of letters in the cyphertext is inversely proportional to their frequency in plain English. However, looking at the key, we can see that some common letters still represent other common letters, and some uncommon letters still represent other uncommon letters. So, we have not reversed the correlation that frequency analysis attempts to find, nor have we amplified it, but instead just weakened it as much as possible.

The only other known-cyphertext attack I can imagine is finding substrings that are not found in the cyphertext, recording which ones cannot be polygraphs, and working backwards by the process of elimination, which seems like a monumental task. But this was to be expected, many simple algorithms are considered uncrackable with known cyphertext attacks.

Improvements

The biggest remaining weakness in the algorithm, from the substitution standpoint, is that our relative frequencies in the plaintext must closely match English to get a good variance of substitutions in the key. We can start to fix this by using our own plaintext relative frequencies in place of the English ones. This will not significantly change our algorithm, but it will unfortunately make it weaker when used on its own, because the relative frequencies in our cyphertext will match the relative frequencies in our plaintext:

Frequencies in file	Frequencies in English	Frequencies in file	Frequencies in English
T: 0.090	e: 0.127	B: 0.030	m: 0.024
I: 0.086	t: 0.091	W: 0.026	w: 0.023
P: 0.083	a: 0.082	J: 0.026	f: 0.022
A: 0.083	o: 0.075	Z: 0.023	y: 0.020
L: 0.071	i: 0.070	X: 0.023	g: 0.020
E: 0.071	n: 0.067	K: 0.023	p: 0.019
S: 0.053	s: 0.063	G: 0.023	b: 0.015
H: 0.045	h: 0.061	F: 0.023	v: 0.010
M: 0.034	r: 0.060	N: 0.019	k: 0.008
C: 0.034	d: 0.043	V: 0.015	j: 0.002
Y: 0.030	l: 0.040	O: 0.011	z: 0.001
U: 0.030	u: 0.028	D: 0.011	x: 0.001
R: 0.030	c: 0.028	Q: 0.008	q: 0.001

Figure 4: The relative frequencies in the cyphertext now closely match those from the plaintext in Figure 3. On its own this is a weaker cypher, but it provides more variance for other cyphers to work with.

```
frequencies relativecyphertext
java PolyCypher.java -r -e "$(cat plaintext)" >
relativecyphertext
<copy and separate the key from the cyphertext>
java PolyCypher.java -r -k "$(cat relativecyphertextkey)" -d
"$(cat relativecyphertext)
```

The advantage of this is that additional cyphers will be more effective, as the highest frequencies will possess many different substitutions, as they would in any asymmetric encryption algorithm. And, once we can match the relative frequencies consistently, we can manipulate them further, for example by making every letter have approximately equal frequencies, or by mimicking the relative frequencies of other languages. Only a few more lines of code give the normalized relative frequency cypher, which just flips all the relative frequencies across their average before creating the key:

Frequencies in file	Frequencies in English	Frequencies in file	Frequencies in English
Z: 0.086	e: 0.127	B: 0.034	m: 0.024
Q: 0.086	t: 0.091	H: 0.023	w: 0.023
Y: 0.079	a: 0.082	D: 0.023	f: 0.022
M: 0.079	o: 0.075	U: 0.019	y: 0.020
K: 0.079	i: 0.070	R: 0.019	g: 0.020
J: 0.079	n: 0.067	V: 0.011	p: 0.019

```
  I: 0.060    s: 0.063  |   N: 0.011    b: 0.015
  W: 0.056    h: 0.061  |   G: 0.011    v: 0.010
  A: 0.053    r: 0.060  |   S: 0.008    k: 0.008
  L: 0.049    d: 0.043  |   P: 0.008    j: 0.002
  F: 0.049    l: 0.040  |   T: 0.000    z: 0.001
  E: 0.045    u: 0.028  |   O: 0.000    x: 0.001
  X: 0.034    c: 0.028  |   C: 0.000    q: 0.001
```

```
frequencies normalizedcyphertext
java PolyCypher.java -n -e "$(cat plaintext)" >
normalizedcyphertext
<copy and separate the key from the cyphertext>
java PolyCypher.java -n -k "$(cat normalizedcyphertextkey)" -d
"$(cat normalizedcyphertext)"
```

This version of the process has the nicest properties to show off; the relative frequencies of the cyphertext no longer match those of plaintext, both in order and in proportion, and the digraphs and trigraphs are all split up. Unfortunately, several letters were not used, but they can still be found in the key. They are not clustered around the lower frequencies though; the key is just too large for this plaintext to use every substitution. This is just as much due to lack of variance in the plaintext as it is to the key, as the length of the key is still exactly 200 characters, which brings us to our last set of improvements.

This entire algorithm was implemented to look good as a demonstration, which limits its potential effectiveness. The algorithm uses only thirty-five characters externally and internally, and both the cyphertext and the key itself has been formatted to be easy to look at on paper. There is no reason, for example, that we cannot use the same character sets for the first and second mark on each key point; I was only using single letters and digits to avoid continuity. Doing so could greatly increase the number of possible substitutions each character possesses, without necessarily increasing the size of the key, because it could also hold more information. The next step would be to just use byte pairs instead of letter-number pairs, which would not change the algorithm significantly.

Alice and Bob would benefit if Eve could not see this detail, although she might guess it eventually if she notices all their messages are of even length. In terms of analyzing the cypher, a larger alphabet will make almost any cypher much stronger, so it is really a moot point. The algorithm is most vulnerable to known plaintext attacks, although it slightly more well equipped than a traditional substitution cypher, because Eve recovers less of the plaintext if she knows a single letter of it than she would with the traditional cypher. This could easily be avoided by complicating the algorithm, but it would better serve its purpose as a lightweight component in a multi cypher with a separate component like a Vigenère cypher chosen to provide resistance to known plaintext attacks. So, the version provided is the most vulnerable possible.

There is also potential in seeing how many layers can be conveniently added to the cypher with a given key size. If separate sets of characters are still used for the letter and number marks, they could be interleaved by keeping the letters and numbers in their own separate orders, but weaving them between each other in the key. This would the key harder to interpret in a known-key attack, and an extra 100 bits of information could be recovered from a 200-length

key in this way at no additional cost. However, it would probably be more practical to just use a smaller key without separate character sets, and append information to it as necessary.

Conclusion

I think this cypher is part of a larger pattern that I would like to explore more, which is taking effective decryption methods that are already known, and seeing if there is a way to work backwards with them to develop evil traps to snare Eve in. I would also like to see which of these individual cyphers can work well together and analyze the strength of smaller key sizes. There is already an abundance of encryption algorithms available for use with all computers, and they will probably stay about the same until a chunk of NP gets knocked loose. So, a niche I feel is still worth exploring is finding the most powerful cryptographic procedure that can easily performed with paper and pencil. These serve both as a challenge, and a failsafe if urgent cryptographic power were ever necessary in the absence of technology.

Many once trusted cyphers such as the Vigenère have suddenly revealed an unseen weakness, sometimes after decades of accumulated confidence that comes from frequent use has long since set in. I would guess that this would be a much worse problem now more than ever, but luckily, the technology that threatens cryptography currently the same that strengthens it. This cypher idea was the result of me imagining the worst-case scenario in a substitution cypher assignment; it is designed to be hard for humans. However, there are still a few things left that humans may do more effectively than computers, and if there is a such thing as an unbreakable encryption algorithm, I think that it *must* be incomprehensible to computers. And so, I imagine that algorithm would be written and performed on paper, or some other medium computers cannot easily manipulate. Perhaps an artistic form of pictogram could work well for this cypher, with the key hidden in the fine visual details of an image or drawing. But for now, and at least until my art class next semester, I am out of ideas.