3.1 Vigenere Cipher

Although the form of this cipher that is currently in most books isn't exactly what Vigenere proposed, it shows the essential idea that Vigenere had to thwart frequency analysis.

The idea is as follows:

Pick a keyword, such as "COMPUTER". Now, to encrypt a message, do the following:

```
Plaintext:  MEETMEATTHESTORE
Keyword  :  COMPUTERCOMPUTER
Ciphertext: OSQIGXEKVVQHNHVV
```

Essentially, to encrypt, you line up the plaintext with the keyword written down repeatedly (until you get to the end of the message), and then you add the numeric values of both letters (0 to 25) and take the result mod 26 to obtain the corresponding ciphertext letter. In code, we'd have something like this:

```
for (i=0; i<msg.length; i++)
  cipher[i]      =      (char)((msg[i]-'a'+key[i%key.length]-
'a')%26+'a');
```

The code above assumes that all variables are declared, and msg, cipher and key are all character arrays in Java.

Decrypting is performed by simply subtracting out the values of the keyword from the ciphertext in the appropriate manner.

It is clear that this cipher disrupts frequency analysis because the same letter in the ciphertext (such as the two Vs at the end) can be obtained from two different letters in the plaintext. Similarly, two of the same plaintext letters can map to different ciphertext letters.

Of course, if one knew the key length, then one could obtain some frequency information, by picking at every kth character, where k was the key length. (In essence k groups of characters could be formed. Each of the letters within a single group would have been shifted by the same key letter, thus, frequency information of each group would be preserved in these ciphertext letter groups.

So, the question is, how can the key length be obtained?

There are two standard methods:

1) The Kasiski Test
2) Index of Coincidence

The Kasiski Test relies on the following observation:

The probability that two repeating strings in the ciphertext occurring from two different plaintexts is quite rare. The reason for this is that the keyword has relative "spacing" between its characters. For two different sets of plaintext characters to have "complementary" spacing and to STILL make sense is quite unlikely.

In our example above, MEET encrypted to OSQI. Now, imagine using the part of the key UTER instead of COMP. The corresponding plaintext letters, when added to UTER that would yield the ciphertext OSQI are UZMR, which is NOT an English word. In fact, we are likely to find that if we lined up the ciphertext OSQI under any other portion of the keyword than COMP, that the corresponding plaintext would not be valid English.

Thus, if we saw OSQI in the ciphertext twice, it is overwhelmingly likely that it is the SAME plaintext word, encrypted using the SAME keyword letters. Imagine that the first OSQI appeared at index 110 while the second OSQI appeared at index 158. Taking the difference of these two, we get 48. It's likely then, that the keyword length divides evenly into 48, since the same part of the keyword lined up at these two indexes. We could grab further pairs of repeating ciphertext and note the distance between where these pairs occurred. For example, imagine if another string repeated at indexes 83 and 171. Then we'd also know that the keyword length divided 171 – 83 = 88 as well.

The Kasiski Test is simply finding the lengths between pairs of repetitions of the ciphertext, and then taking the greatest common divisor of each of these values, since the keyword length is likely to divide into each of these values.

We can take the greatest common divisors of two integers using Euclid's Algorithm which is essentially repeated division.

Here is an example with 88 and 48:

88 = 1x48 + 40
48 = 1x40 + 8
40 = 5x8 + 0, so the GCD is 8, the value directly above the 0.

We first divided 88 by 48, yielding a quotient of 1 and a remainder of 40.
Then the new remainder becomes the new number to divide by and
we divide 48 by 40, yielding a quotient of 1 and a remainder of 8.
Finally we divide 40 by 8 yielding a quotient of 5 and a remainder of 0.

Here is a longer example to determine the gcd of 588 and 259:

588 = 2*259 + 70
259 = 3*70 + 49
70 = 1*49 + 21
49 = 2*21 + 7

21 = 3*7, so that the gcd(588, 259) = 7.

Another method used to determine the length of the key is the Index of Coincidence test.

Given a group of letters, their index of coincidence is defined as the probability that two randomly chosen letters in the group are the same.

For example, for text from the English language, the probability that the first letter chosen is 'A' is about 9% and the probability that the second letter chosen is also 'A' is also 9%. Thus, the probability that two randomly chosen letters are both 'A' is .09*.09 = .0081. We can simply sum up the probabilities that two randomly chosen letters are both 'B', 'C', …, 'Z' to calculate their index of coincidence.

Mathematically, for a set of letters, we have: IC = $\sum_{i=1}^{26} \frac{f_i(f_i-1)}{n(n-1)}$, where f_i represents the frequency of the ith letter in the set of letters and n represents the total number of letters in the set of letters. This is equal to the probability that two randonly chosen letters from the set are the same.

It turns out that the Index of Coincidence of English is approximately .065. But, the index of coincidence of random letters is just 1/26 ~ .038. (This is because if each letter is equally likely, then the probability that the second one would match the first is just 1/26, the probability of pulling the appropriate second letter.)

So what you can do with some Vigenere ciphertext is make a guess at the keyword length, k. Then split up the ciphertext letters into k groups. (If k were 8, then the first letter, ninth letter, sixteenth letter, etc. would all be in the same group.) For each of the k groups, calculate the index of coincidence. If most of the groups are around .065, then your guess for k is most likely correct. Otherwise, if these numbers are lower, it is probably not. Try different values of k until you find one that works.

Once the key length, k, is determined, then create the k groups of letters where each letter in a group was shifted by the same value. What we can do for each group is try each of the 26 possible shifts. Our goal is to get the corresponding frequencies to line up with those of the English language. One easy way to do this is to attempt to minimize the frequencies that line up at the low-frequency letters, J, K, Q, X and Z. This is much easier to visualize than to write in text!

Another mathematical way to line up the relative shifts between each set of alphabets is utilizing the mutual index of coincidence test. Given two sets of letters, with frequencies f_i and g_i, with $\sum_{i=1}^{26} f_i = m$ and $\sum_{i=1}^{26} g_i = n$, the mutual index of coincidence of those two sets of letters is defined as $\sum_{i=1}^{26} \frac{f_i g_i}{mn}$, which is the probability that a randomly chosen letter from the first set is the same as a randomly chosen letter from the second set.

If two sets of letters are "matched up" properly, that should maximize the mutual index of coincidence. So, the basic idea is as follows: Once the keyword length is known, split up all the letters into bins representing letters shifted by each keyword letter. Calculate the letter frequencies of each bin. Then, calculate the mutual index of coincidence between bin1 and the letter frequencies of the English language. Then, shift each letter in bin 1 by 1 backwards (B's become A's, C's become B's, etc.) and repeat. Do this 24 more times, trying each possible shift of the letters in bin 1. Whichever shift gives you the greatest mutual index of coinicidence is likely to be keyword letter for that bin.

3.2 Autokey Cipher

Ultimately, the repetition in Vigenere led to its compromise. In order to prevent this repetition, one idea was the Autokey cipher, which uses either part of the plaintext or part of the ciphertext as the key, after the key has been exhausted.

For example, if the key is "HOUSE" and the message is "IAMCOMINGHOME", then we encrypt as follows:

```
Plaintext : IAMCOMINGHOME
Key       : HOUSEIAMCOMIN
Ciphertext: POGUSUIZIVAUR
```

Unlike Vigenere, the same letter is not used to encrypt every fifth plaintext letter.

Alternatively, we can use the ciphertext as the key as well in a similar manner:

```
Plaintext : IAMCOMINGHOME
Key       : HOUSEPOGUSBWT
Ciphertext: POGUSBWTAZPIX
```

In pseudocode, here are code segments that determine the ciphertext for both situations:

Using the plaintext as the key

```
for (i=0; i<k.length; i++)
  c[i] = p[i] + k[i];
for (i=k.length; i<p.length; i++)
  c[i] = p[i] + p[i-k.length];
```

Using the ciphertext as the key

```
for (i=0; i<k.length; i++)
  c[i] = p[i] + k[i];
for (i=k.length; i<p.length; i++)
  c[i] = p[i] + c[i-k.length];
```

For the most part the code above works except for the specific letter calculations. (You'd have to subtract and add ascii values and mod by 26 in the appropriate places to actually

get it to work. I omitted that because it makes it look ugly. The essence of the algorithm is much more easily seen as its written above.)

The latter cipher is quite easy to break because we can solve for the ith plaintext character:

```
p[i] = c[i] - c[i-k.length] (for all i >= k.length)
```

Remember that the whole ciphertext is given. All that is needed to plug into this formula is the length of the key. All you need to do is guess at this length, and see if intelligible plaintext comes out. The letters you can not recover this way are the first k.length letters of the plaintext, since those were encrypted using the key instead of the ciphertext itself. These can typically be deduced from the contents of the rest of the message.

In our example above, with the ciphertext `POGUSBWTAZPIX` let's try different possible keyword lengths:

```
POGUSBWTAZPIX
-POGUSBWTAZPI
-------------
 ZSO, could continue, but doesn't look promising

POGUSBWTAZPIX
- POGUSBWTAZP
-------------
  RGM, also doesn't look promising

POGUSBWTAZPIX
-  POGUSBWTAZ
-------------
   FEVC, not promising

POGUSBWTAZPIX
-   POGUSBWTA
-------------
    DNQ, not promising

POGUSBWTAZPIX
-    POGUSBWT
-------------
     MINGHOME, the key length is 5.
```

The other version of the autokey cipher is more difficult to attack. However, it does retain a periodicity that can be exploited. In particular, what one can do is guess the key length. From there, guess what the initial key is. In our example, the key length was 5 and the first keyword letter was H.

```
Plaintext : IAMCOMINGHOME
Key       : HOUSEIAMCOMIN
Ciphertext: POGUSUIZIVAUR
```

Take the first ciphertext letter, P, and subtract H from it, yielding I as the first plaintext letter. Now, take the sixth ciphertext letter U and subtract I from it, yielding M, which should be the sixth plaintext letter. Next, take the 11th ciphertext letter A, and subtract from it M to yield O, the 11th plaintext letter.

In this manner, if the guess to the keylength and keyword letter are correct, then the letters obtained will be plaintext letters. We can check to see if our guesses are correct by taking the letters we obtain for the supposed plaintext and running them through a low frequency test. The same can then be done for each other letter. Once we get one success, the rest will come quickly because after one success, the key length is known.

3.4 Nihilist Cipher

Here are the mechanics of the cipher:

1) Pick a keyword without repeating letters.
2) Create a 5x5 grid using the keyword as follows. (The grid below is for the keyword "PROBLEMS"

	1	2	3	4	5
1	P	R	O	B	L
2	E	M	S	A	C
3	D	F	G	H	I/J
4	K	N	Q	T	U
5	V	W	X	Y	Z

3) Select a second key word, say "HOUSE"
4) Using the grid above, replace the second key word with the corresponding two numbers per each letter, so "HOUSE" becomes "34 13 45 23 21"
5) Convert the plaintext into pairs of numbers as the second key word is.
6) Now, proceed like a Vigenere cipher, adding the plaintext to the key. Repeat the key as often as necessary.

As an example, consider the plaintext `IAMCOMINGHOME`.

```
Plaintext : I  A  M  C  O  M  I  N  G  H  O  M  E
Converted : 35 24 22 25 13 22 35 42 33 34 13 22 21
Key       : 34 13 45 23 21 34 13 45 23 21 34 13 45
Ciphertext: 69 37 67 48 34 56 48 87 56 55 47 35 66
```

To decrypt, just subtract the key from the ciphertext. The exception to these rules is if the sum of the plaintext and key exceeds 100. In this case, the matching ciphertext should just be the result mod 100. To decrypt for a number less than 12, add 100 and then subtract the ciphertext and proceed from there.

Ultimately, this is nothing but a Vigenere cipher where the plaintext is in numbers instead of letters. This does complicate things, but the same tests that worked for Vigenere can be applied here. Similarly, once you make a guess at a couple letters in the grid, many more can be filled in, due to the pattern that is used to do so.

3.5 Cylinder Cipher

There are potentially many different versions of this cipher, but the one discussed in the text is due to Etienne Bazeries. It includes twenty interchangeable cylinders, each with a different permutation of the alphabet. These cylinders are all listed in the text on page 52. They are hard-coded values. The key to the cipher isn't the cylinders, but the order in which they are chosen. (There are 20! such orders, so this is the keyspace.)

To encrypt, line up the first 20 letters of the plaintext on the cylinders. The letters directly below will be the ciphertext.

For the next 20 letters of the plaintext, get these lined up on the cylinders. The letters two spots below will be the ciphertext.

Repeat this pattern until you've looped through all rows below the plaintext. At this point, wrap around so that you use the letters directly below the plaintext again. (This will occur at character number 501, since there are (20 cylinders)x(25 letters rotations of each cylinder) = 500 characters.)

Clearly not all 20 cylinders must be used. Furthermore, the users could agree upon a different system of which line of the cylinders to use at each juncture.

To cryptanalyze the Bazeries cipher, the book discusses a known-plaintext attack.

Imagine that you knew that the first five letters of the plaintext were "ALICE" and that the first five letters of the ciphertext are "EKJDA"

Then it could be deduced that whichever cylinder was used for the first letter must have an E directly after an A on it. Two cylinders fit this description: #2 and #3.

Similarly, we check to see which cylinders have the letter K directly after the letter L. These are cylinders #4, #5 and #6.

In this manner, we can greatly reduce our search for the order of the cylinders. We can simply apply a brute force attack from this point, searching through all of the remaining possibilities, until a recognizable plaintext appears.

If it is unknown how many cylinders are being used, then there are most definitely major complications in this method.

3.6 Rotor Ciphers – The Enigma

The Code Book does a great job of explaining the Enigma and allowing the reader to visualize its setup.

Basically, here are the major components:

1) The Rotors
2) The Reflector
3) The Plugboard
4) The Lampboard
5) The Keyboard

The keyboard is where the user types in the plaintext and the lampboard is where the ciphertext shows up.

The keyboard is directly connected to the plugboard. The plugboard allows the exchange of six pairs of letters. So, for example, if the plugboard connections were (A-W, F-D, J-Q, K-L, I-R and Z-M) then when the user types in an A, it will get changed to a W, W to A, etc.

Then the plugboard is connected to the first of three rotors. Each rotor performs an arbitrary (fixed) permutation of the letters. Thus, the three rotors, all connected together, ultimately perform a single permutation. Then this output is reflected by the reflector back through the three rotors, then through the plugboard to the lampboard where the ciphertext lights up.

The story about the breaking of the Enigma is quite entertaining and also does a great job of illustrating many principles of the implementation of cryptographic schemes. Included below is a abbreviated version of this story (which is also discussed in The Code Book.)

Hans-Thilo Schmidt, a disgruntled worker in the German cipher office sold blueprints of the Enigma to a secret French agent. They met at a hotel room and the French agent photographed pages of the blueprint. The French then gave the information to the Polish due to the peace-time treaty that the two countries had to share intelligence information. Marian Rejewski exploited properties of the Enigma that Scherbius most likely never knew existed. In particular, since they encrypted the message key twice at the beginning of every message, he started noting that the same letters were encrypted differently, but followed a pattern. For example, if the message key was "HWN", then the first six letters of the message were always "HWNHWN". It may be the case that the first H encrypts to P and the second one encrypts to X. Thus, P -> X. Continue writing these all down. Ultimately, they will link together, you might get P->X, X->Q, and Q->P. This would be

a chain of link three. Rejewski noticed that for each possible setting of the scrablers, the lengths of these links would be unique. So, he started categorizing the length of links for each possible scrambler setting. He did so with the aid of "bombes" he built that were machines that automatically took care of most of the work. After a full year of work, Rejewski had a full catalog of all of the different possible scrambler settings. Thus, after intercepting enough messages in the morning, the Polish could calculate the length of those links for the day, and look up in their book which scrambler setting was being used for that day code.

Ultimately, the British continued the work of the Polish. Some other techniques the British used include making use of "cillies", which were common lines in the plain text or biased choices of the message key. (The term may have come from one of the common message keys, CIL, which were the initials of a girlfriend of an Enigma operator.) Also, each day at around 6am, there would be a message with the weather report. It always contained the German word for weather, "WETTER" This allowed for a known-plaintext attack. The British were also able to reduce their searches because no plugboard cables could connect adjacent letters and no scrambler could be left in the same location as the previous day. The latter "mistake" allowed the British to only search HALF of the possible scrambler settings.

The key thing to remember about how Enigma is broken is that the machine exhibited hidden patterns which didn't depend on the plugboard. Hence, a brute force search was possible since only $6x26^3$ settings of the scramblers had to be examined. Most encryption systems ultimately have such patterns that are not easily detectable. Furthermore, espionage and misuse of an encryption system can lead to its downfall as well, and are just as important as the theoretical strength of the system, if not more so. Hans Thilo-Schmidt and the way the Germans used the system both helped the Allies break the system.