2 Statistical Models and Cryptanalysis

Last week we saw that we can break a monoalphabetic substitution cipher with statistical analysis and a little bit of hard work. But there are other ciphers, like the Vigenère cipher, that we can't break easily. But how can we tell which situation we're in? How can we tell if we're looking at a monoalphabetic cipher or a polyalphabetic one, without something already knowing? One answer comes from the index of coincidence.

2.1 The Index of Coincidence

Definition 2.1. Let $\mathbf{s} = c_1 c_2 \dots c_n$ be a string of n letters. The *index of coincidence* of $mathbfs$ is denoted $\text{IndCo}(\mathbf{x})$ and is defined to be the probability that two randomly chosen characters in the string $\mathbf{s}$ are identical.

Proposition 2.2. *Let $\mathbf{s} = c_1 c_2 \dots c_n$ be a string of n, and let F_i be the frequency with which the letter i appears in the string $\mathbf{s}$. Then*

$$\text{IndCo}(\mathbf{s}) = \frac{1}{n(n-1)} \sum_{i=0}^{25} F_i(F_i - 1). \tag{2.1}$$

Proof. There are $\binom{n}{2} = \frac{n(n-1)}{2}$ different ways to select two letters at random from $\sim$.

Each letter i appears F_i times, so there are $\binom{F_i}{2} = \frac{F_i(F_i)-1}{2}$ ways to choose the letter i twice. Adding these ways for each letter, there are

$$\sum_{i=0}^{25} \frac{F_i(F_i-1)}{2} \tag{2.2}$$

ways to choose two of the same letter from the string.

The chance of two randomly chosen letters being identical is equal to the number of ways to choose two identical letters, divided by the total number of ways to choose letters. Thus we divide equation (2.2) by $\frac{n(n-1)}{2}$ and get the formula in equation (2.1). □

For any given string we can calculate the index of coincidence from the frequency table.

Example 2.3. In your homework, we encrypted the string $\mathbf{s}$ ="Rats live on no evil star". This message has twenty letters total; it has ten distinct letters, and each appears twice. Thus we have

$$\text{IndCo}(\mathbf{s}) = \frac{1}{20 \cdot 19} \cdot 10(2 \cdot 1) = \frac{1}{19} \approx 0.53.$$

We also looked at the string $\mathbf{t}$ = "A man a plan a canal panama". This message has 21 total letters, with 10 "a"s, 2 "m"s, 4 "n"s, 2 "p"s, 2 "l"s, and 1 "c". Thus we compute

$$\text{IndCo}(\mathbf{t}) = \frac{1}{21 \cdot 20}(10 \cdot 9 + 2 \cdot 1 + 4 \cdot 3 + 2 \cdot 1 + 2 \cdot 1 + 1 \cdot 0) = \frac{108}{420} \approx .257.$$

As we'll see, this index of coincidence is really unusually high. But this isn't really surprising; you probably noticed already that this text has a ridiculous number of "a"s in it.

(Notice also that the term from the "c" is $1 \cdot 0 = 0$. This makes sense because the odds of the "c" matching another letter in the text are in fact zero, since there's only one of them').

We can also calculate two very important and common values for the index of coincidence:

Proposition 2.4. *1. If* $\mathbf{s}$ *is a string of letters generated uniformly at random, then* $\text{IndCo}(\mathbf{s}) \approx .038$.

2. If $\mathbf{s}$ *is a string of letters with the frequencies common in written English, then* $\text{IndCo}(\mathbf{s}) \approx .068$.

Proof. 1. The letters are generated uniformly at random, so $F_i \approx \frac{n}{26}$ for each i. Thus we have

$$\begin{aligned}\text{IndCo}(\mathbf{s}) &\approx \frac{1}{n(n-1)} \sum_{i=0}^{25} \frac{n}{26}\left(\frac{n}{26} - 1\right) = \frac{1}{n(n-1)} \sum_{i=1}^{25} \frac{n^2}{26^2} - \frac{n}{26} \\ &= \frac{1}{n(n-1)}\left(\frac{n^2}{26} - n\right) = \frac{n/26 - 1}{n-1} \approx \frac{1}{26} \approx .038.\end{aligned}$$

We probably could have guessed this without working through the computation—if the letters are chosen randomly, the chances of two of them matching should indeed be about 1/26.

2. This is a straightforward if tedious calculation from the letter frequencies given in figure 1.1. I might write it up for here later.

□

Corollary 2.5. *If* $\mathbf{s}$ *is a string of letters corresponding to an English text encrypted by a simple (monalphabetic) substitution cipher, then we should expect* $\text{IndCo}(\mathbf{s}) \approx .068$.

Proof. A monoalphabetic substitution cipher is just a permutation of the alphabet; so while the frequency of individual letters is altered by applying a substitution cipher, the index of coincidence is not □

This gives us a way to test whether a string of letters was likely enciphered by a simple substitution cipher or not. We compute the index of coincidence of the string. If the index is close to .068 then we likely have a string encrypted with a monoalphabetic cipher. If the index is close to .038 then it is likely that our string is not encrypted monoalphabetically.

This test is especially useful when we proceed to break the Vigenère cipher in the next section.

2.2 Breaking the Vigenère cipher

Monoalphabetic ciphers are fairly simple to implement, but also quite easy to break. The Vigenère cipher is much harder to break; for three centuries it was known as "The Undecipherable Cipher". In 1854 Charles Babbage successfully broke it, and in 1863 Friedrich Kasiski published a method for breaking the Vigenère cipher.

2.2.1 Finding the Key Length

Cryptanalysis of the Vigenère cipher proceeds in two steps. The first (and more difficult) step is to determine the length of the key. There are two basic approaches to this, but both use the same basic idea.

The low-tech way is to look for repeated strings in the ciphertext. If the same string appears in more than one place, it's likely (but not definite!) that it's the same plaintext string encrypted by the same part of the keyword, so the distances between these reoccurrences gives us information about possible keyword lengths.

A simple version of this is to displace the ciphertext by $2, 3, 4, \ldots$ places, and see which displacement generates the most coincidences—points where the displaced ciphertext is identical to the original.

Another variant is to look for places where the same trigram is repeated more than once in the ciphertext, and measure the offset or distance between them. Then find a number that is a factor of most (but not necessarily all) of thse offset numbers; that's probably the length of the keyword.

This method is called the *Kasiski Method* or the *Kasiski Test*, since it was first developed by Charles Babbage.

Example 2.6. Consider the ciphertext :

```
zpgdl rjlaj kpylx zpyyg lrjgd lrzhz qyjzq repvm swrzy rigzh
zvreg kwivs saolt nliuw oldie aqewf iiykh bjowr hdogc qhkwa
```

```
jyagg emisr zqoqh oavlk bjofr ylvps rtgiu avmsw lzgms evwpc
dmjsv jqbrn klpcf iowhv kxjbj pmfkr qthtk ozrgq ihbmq sbivd
ardym qmpbu nivxm tzwqv gefjh ucbor vwpcd xuwft qmoow jipds
fluqm oeavl jgqea lrkti wvext vkrrg xani
```

First, we can compute that the index of coincidences for this ciphertext is about .039. This suggests it wasn't encrypted with a monoalphabetic substitution cipher.

We think that it was encrypted with a Vigenère cipher, and we want to find the key length. The first method is to look for coincidences. To do that we can lay out the lines of ciphertext shifted. So the first line would be:

```
0:zpgdl rjlaj kpylx zpyYg lrjgd lrzhz qyjzq repvm swrzy rigzh
1: zpgd lrjla jkpyl xzpYy glrjg dlrzh zqyjz qrepv mswrz yrigz h
```
1 coincidence
```
0:zpgdl rjlaj kpylx zpyyg lrjgd lrzhZ qyjzq repvm swrzy rigzh
2:  zpg dlrjl ajkpy lxzpy yglrj gdlrZ hzqyj zqrep vmswr zyrig zh
```
1 coincidence
```
0:zpgdl rjLaJ kpylx zpyyg lrjgd lrzhz qyjzq repvm swrzy Rigzh
3:   zp gdLrJ lajkp ylxzp yyglr jgdlr zhzqy jzqre pvmsw Rzyri gzh
```
3 coincidences
```
0:zpgdl rjlaj kpylx zpyyg lrjGd lrzhz qyjZQ repvm swrzy rigzh
4:    z pgdlr jlajk pylxz pyyGl rjgdl rzhZQ yjzqr epvms wrzyr igzh
```
3 coincidences
```
0:zpgdl rjlaj kpylx zPYyg lrjgd LRzhz qyjzq repvm swrzy rigZh
5:       zpgdl rjlaj kPYlx zpyyg LRjgd lrzhz qyjzq repvm swrZy rigzh
```
5 coincidences
```
0:zpgdl rjlaj kpyLx zpyYg lrjgd lrzhz qyjZq repvm swrzy rigzh
2:        zpgd lrjLa jkpYl xzpyy glrjg dlrZh zqyjz qrepv mswrz yrigz h
```
3 coincidences
```
0:zpgdl rjlaj kpylx zpyyg lrjgd lrzhz qyjzq repvm swrzy rigzh
2:         zpg dlrjl ajkpy lxzpy yglrj gdlrz hzqyj zqrep vmswr zyrig zh
```
2 coincidences

From this we might guess that the keyword has length five; but this is a small sample size. If we do this count for the entire ciphertext (preferably but not necessarily by computer), we get:

Shift	1	2	3	4	5	6	7	8	9
Coincidences	6	6	9	5	8	13	15	11	11

This suggests but does not prove that the keyword has length 7.

If we look at repeated trigrams instead, we get the following results:

Trigram	Places	Offset	Trigram	Places	Offset
avl	117 and 258	$141 = 3 \cdot 47$	bjo	86 and 121	$35 = 5 \cdot 7$
dlr	4 and 25	$21 = 3 \cdot 7$	gdl	3 and 24	$16 = 2^4$
lrj	5 and 21	$98 = 2 \cdot 7^2$	msw	40 and 138	$84 = 2^2 \cdot 3 \cdot 7$
pcd	149 and 233	$13 = 13$	qmo	241 and 254	$98 = 2 \cdot 7^2$
vms	39 and 137	$84 = 2^2 \cdot 3 \cdot 7$	vwp	147 and 231	$84 = 2^23 \cdot 7$
wpc	148 and 232	$21 = 3 \cdot 7$	zhz	28 and 49	$21 = 3 \cdot 7$

We see that the factor 7 appears often in the offset column; thus the keyword is probably length 7. This matches our tentative conclusion from earlier.

The higher-tech version of this is to use the index of coincidence again. The index of coincidence only pays attention to *letter* distributions, and doesn't care about their order. So if we take all the letters that are encrypted by the same letter of the keyword, and calculate the index of coincidence, we should get a number close to .068; if not, we should get an index closer to .038.

To run this test, we break our ciphertext into k pieces: the first has letters $1, k+1, 2k+1, \ldots$, the second has letters $2, k+2, 2k+2, \ldots$, and so on. We compute the index of coincidence for each of these strings. If most of them are close to .068 then the keyword is probably of length k; if most of them are close to .038 then the keyword is probably not of length k.

Example 2.7. Continuing the look at our previous ciphertext, we can compute the indices of coincidence for each substring, for each possible shift.

Shift	indices							
2	.038	0.40						
3	0.39	0.42	0.38					
4	0.34	0.42	0.39	0.35				
5	0.38	0.39	0.43	0.28	0.36			
6	0.38	0.40	0.39	0.38	0.32	0.33		
7	0.62	0.57	0.65	0.60	0.60	0.64	0.64	
8	0.37	0.29	0.38	0.33	0.34	0.57	0.40	0.39

One of these rows looks very unlike the others; this again gives evidence that the key length is 7.

2.2.2 Finding the Key

Once we have found the key length, there are two basic approaches we can use to finding the key.

First, we can simply do a frequency analysis on subsets of the ciphertext. If we believe that the key has length 5, then the first, sixth, eleventh, etc. letters are all shifted by the same amount. So we can do a frequency analysis on this set to decipher the shift.

Note that the frequency analysis is made much easier by the fact that we know we're working with a Caesar cipher, so there are only 26 possibilities. Thus we're trying to select a shift that makes *all* the high-frequency ciphertext letters into high-frequency plaintext letters.

Example 2.8. In the previous example, we've now seen that the key length is seven. So we can look at the substring $\mathbf{s}_1$ consisting of the 1st, 8th, 15th, ... letters, which is `zlxrh rrhwl oehdw eokli lwvlh phqby nwhwf julrx x` which has frequency counts

Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Frequency	0	1	0	1	2	1	0	6	1	1	1	6	0	1	2	1	1	4	0	0	1	1	5	3	1	1

We see the frequent letters here are H, L, and less so W, R, and X in that order. We might guess that either H or L corresponds to a plaintext e. If H corresponds to e, that sends L to i, R to o, W to t, and X to u, which isn't unreasonable; if L corresponds to e, then that sends H to a, R to k, W to p, and X to q, which seems somewhat less likely, since we don't expect k and q to be among the most common letters in an English string.

This isn't dispositive, but it is highly suggestive that the first letter of the keyword corresponds to a shift three to the right; thus the first letter might be D. We can repeat this process for each substring.

This works, but involves a lot of guesswork and intuition and puzzle-solving. There is a second method that requires a bit more calculation, but is also rather more robust and automatic.

Definition 2.9. Let $\mathbf{s} = c_1c_2\dots c_n, \mathbf{t} = d_1d_2\dots d_m$ be two strings of letters. Then we define the *mutual index of coincidence* to be $\text{MutIndCo}(\mathbf{s}, \mathbf{t})$, the chance that a randomly selected letter of $\mathbf{s}$ is the same as a randomly selected letter of $\mathbf{t}$.

Proposition 2.10. *Let $\mathbf{s} = c_1c_2\dots c_n, \mathbf{t} = d_1d_2\dots d_m$ be two strings of letters, and let $F_i(\mathbf{s})$ be the number of times the ith letter appears in the string $\mathbf{s}$. Then:*

1.

$$\text{MutIndCo}(\mathbf{s}, \mathbf{t}) = \frac{1}{nm}\sum_{i=0}^{25} F_i(\mathbf{s})F_i(\mathbf{t}).$$

2. *If the letters of* $\mathbf{s}$ *and* $\mathbf{t}$ *are drawn from the same distribution, given by taking English frequencies and permuting the letters, then* $\text{MutIndCo}(\mathbf{s}, \mathbf{t}) \approx .068$.

3. *If the letters of* $\mathbf{s}$ *and* $\mathbf{t}$ *are drawn from different such distributions, then* $\text{MutIndCo}(\mathbf{s}, \mathbf{t}) \approx .038$.

We can use the mutual index of coincidence to test if two strings are drawn from the same substitution. To decrypt a Vigenere cipher, we need to figure out the shift of each substring. We can use the mutual index of coincidence to compute the *relative* shifts.

Let $\mathbf{s_i} = c_i, c_{i+k}, c_{i+2k}, \dots$, and define $\mathbf{s}_i + \sigma$ to be the string $\mathbf{s}_i$ with each letter shifted by σ. Then we can compute $\text{MutIndCo}(\mathbf{s}_i, \mathbf{s}_j + \sigma)$ for $0 \leq \sigma \leq 25$. We expect 25 of these computations to be low like .038, and the last to be high like .068; this last one gives us the relative shift between the ith and jth letter of the keyword.

Thus if β_i is the ith letter of the keyword, this does not and cannot tell us what β_i is; but it can give us relationships among the β_i.

After computing $k-1$ of these, with luck we should know the relative shifts of all the letters of the keyword; this means there are only 26 possible keys, and we can try all of them. Of course, sometimes the mutual index of coincidence happens, randomly to not give enough information; this is a problem we can solve by simply computing more possible mutual indices, possibly up to all $\frac{k(k-1)}{2}$ possibilities.

Example 2.11. If we compute all the possible mutual indices of coincidence in our ciphertext, for a key length of seven, then we get the following "large" indices that indicate a true collision:

i	j	σ	$\text{MutIndCo}(i, j+\sigma)$	Relative shift equation
1	3	1	.067	$\beta_1 - \beta_3 = 1$
3	7	10	.069	$\beta_3 - \beta_7 = 10$
1	4	19	.071	$\beta_1 - \beta_4 = 19$
1	6	16	.071	$\beta_1 - \beta_6 = 16$
3	4	18	.073	$\beta_3 - \beta_4 = 18$
3	5	24	.067	$\beta_3 - \beta_5 = 24$
3	6	15	.074	$\beta_3 - \beta_6 = 15$
4	6	23	.066	$\beta_4 - \beta_6 = 23$
4	7	18	.071	$\beta_4 - \beta_7 = 18$
6	7	21	.069	$\beta_6 - \beta_7 = 21$

Our goal is to solve the equations in the right-hand column—or at least as many as possible! It's entirely possible that one of the high indices will be an accident; when this

happens, you can try dropping one constraint or another and see what you get.

But in this case, the system is fairly straightforward to solve. We get:

$$\beta_3 = \beta_1 + 25 \qquad \beta_4 = \beta_1 + 7$$
$$\beta_6 = \beta_1 + 10 \qquad \beta_7 = \beta_3 + 16 = \beta_1 + 15$$
$$\beta_5 = \beta_3 + 2 = \beta_1 + 1$$

and we can check that this doesn't generate any inconsistencies. Notice that we don't have a solution for β_2 in here, because we didn't get any high indices of coincidence involving $\mathbf{s}_2$. One option is to take the best ones we have; the highest is $\text{MutIndCo}(2, 4 + 24) = .061$, which suggests $\beta_2 = \beta_4 + 24 = \beta_1 + 5$. The other is to look at the results not involving β_2 and basically guess.

So what do our results give us? Well, they tell us that if we know β_1, we know the entire keyword. For instance, if $\beta_1 = 0 = A$, then the keyword is AFZHBKP. If we try decrypting our ciphertext with this word (by *subtracting* it from the ciphertext), we get `zkhwkhulvkdoowxuq...` which isn't very promising.

But recall that there are now only 26 possible keys, so we can simply try each of them. If we do that, we get a table something like this:

β_1	Keyword	Potential plaintext
0	AFZHBKP	`zkhwkhulvkdoowxuq`
1	BGAICLQ	`yjgvjgtkujcnnvwtp`
2	CHBJDMR	`xifuifsjtibmmuvso`
3	DICKENS	`whetherishallturn`
4	EJDLFOT	`vgdsgdqhrgzkkstqm`
5	FKEMGPU	`ufcrfcpgqfyjjrspl`
6	GLFNHQV	`tebqebofpexiiqrok`

We see that when $\beta_1 = 3$, the keyword is "DICKENS", and we get recognizable English out of the ciphertext: we get

`wheth erish alltu rnout tobet heher oofmy ownli feorw hethe rthat stati onwil lbehe ldbya nybod yelse these pages musts howto begin mylif ewith thebe ginni ngofm ylife ireco rdtha tiwas borna sihav ebeen infor medan dbeli eveon afrid ayatt welve ocloc katni ghtit wasre marke dthat thecl ockbe ganto strik eandi began tocry simul taneo usly`

and after replacing spacing and punctuation, we get:

"Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show. To begin my life with the beginning of my life,

I record that I was born (as I have been informed and believe) on a Friday, at twelve o'clock at night. It was remarked that the clock began to strike, and I began to cry, simultaneously."

Remark 2.12. These sorts of attacks can totally work on a Vigenère cipher applied to binary messages. But you have to rely entirely on things more like trigram coincidences and less like single-letter coincidences, since there are only two "letters".

2.3 The Autokey cipher and cribs

2.3.1 Using a crib

One common tool in cryptanalysis is a *crib*, which is a known or guessed bit of plaintext corresponding to a ciphertext. (The term comes from the phrase "to crib notes" or "to crib an answer", meaning to copy or cheat on an assignment).

Often a crib can be used to dramatically simplify cryptanalysis. (In fact, frequency analysis is essentially an attempt to imitate a crib).

Cribs were famously used in Bletchley Park during World War II (where the term was coined). Many German Enigma operators used standardized terminology, including the regular use of the word *Wetter* ("weather") in weather reports, and one operator who repeatedly transmitted the message "Nothing to report".

Enigma operators were required to spell out all numbers, so Turing determined that the single most common word in messages was *eins*, meaning "one". Turing precomputed a catalog of what *eins* would look like encrypted in every possible position with various keys, which dramatically sped up decryption processes by seeing which of those were possible and judging them most likely.

You will notice that this is basically the same idea as frequency analysis: instead of taking common letters, we instead look for common words. *eins* was not enough to break messages on its own, but it could give substantial speedups and hints for other encryption messages.

2.3.2 Breaking the Autokey cipher

Cribs are an especially powerful tool in breaking the Autokey cipher, since the plaintext is *also* most of the keystream.

The basic idea is that we guess a word or phrase we expect to see in the plaintext. Since that word would also have to appear in the keystream, we try using that word as the key at every possible point, and see when the results are plausible English strings. We can extend this out to guess most or all of the message.

Example 2.13. Suppose we intercept the ciphertext:

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
```

We guess that it contains the word "the" somewhere. So we simply see what happens if we use "the" as the key at every possible position.

First we decrypt everything with an offset of zero:

Ciphertext:	O	O	F	I	K	A	A	Q	W	M	P	Q	U	M	X
Key:	T	H	E	T	H	E	T	H	E	T	H	E	T	H	E
Plaintext:	V	H	B	P	D	W	H	J	S	T	I	M	B	F	T
Ciphertext:	Z	X	Y	I	R	K	T	Z	S	P	G	M	G	P	K
Key:	T	H	E	T	H	E	T	H	E	T	H	E	T	H	E
Plaintext:	G	Q	U	P	K	G	A	S	O	W	Z	I	N	I	G
Ciphertext:	Q	M	I	P	L	C	N	W	X	K	E	N	Q	L	D
Key:	T	H	E	T	H	E	T	H	E	T	H	E	T	H	E
Plaintext:	X	F	E	W	E	Y	U	P	T	R	X	J	X	E	Z
Ciphertext:	I	R	F	S	N	I	J	A	M	G	P	W			
Key:	T	H	E	T	H	E	T	H	E	T	H	E			
Plaintext:	P	K	B	Z	G	E	Q	T	I	N	I	S			

A couple of these strings look like they might be English; we might notice "tim" or "aso". Let's see what happens if we assume "tim" is actually part of the plaintext. We now need to see what happens if we assume the original keyword is any of various lengths.

If the keyword has length four, we get

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
--- --- --- --- --- --f bn- the -as o-- --- --- --- --- --- --- --- --- ---

--- --- --- --- --- --t he- aso -gu s-- --- --- --- --- --- --- --- --- ---
```

and while "gus" is possible, "fbn" probably is not.

If the keyword has length five, we get:

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
--- --- --- --- --- -er e-- the --a so- --- --- --- --- --- --- --- --- ---

--- --- --- --- --- -th e-- aso --m ob- --- --- --- --- --- --- --- --- ---
```

This looks more promising, but if we continue building out we get:

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
```

```
--- --- ono --m di- -er e-- the --a so- -mo b-- auo --- --- --- --- --- ---

--- --- mdi --e re- -th e-- aso --m ob- -au o-- ncj --- --- --- --- --- ---
```

which looks unlikely. With a key of length six we get

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
--- --- --- --- --- gqu --- the --- aso --- --- --- --- --- --- --- --- ---

--- --- --- --- --- the --- aso --- gxw --- --- --- --- --- --- --- --- ---
```

Which again doesn't look like English. We can try longer keywords but nothing useful will show up. Similarly, we can try working with the "tim" but we won't really get anywhere.

So is this all the possibilities? No: the table we made started with "the" with an offset of zero from the ciphertext. We need to try again with offsets of one and two. If we build the table with an offset of two, we get

Ciphertext:	O	O	F	I	K	A	A	Q	W	M	P	Q	U	M	X
Key:	-	-	T	H	E	T	H	E	T	H	E	T	H	E	T
Plaintext:	-	-	M	B	G	H	T	M	D	F	L	X	N	I	E
Ciphertext:	Z	X	Y	I	R	K	T	Z	S	P	G	M	G	P	K
Key:	H	E	T	H	E	T	H	E	T	H	E	T	H	E	T
Plaintext:	S	T	F	B	N	R	M	V	Z	I	C	T	Z	L	R
Ciphertext:	Q	M	I	P	L	C	N	W	X	K	E	N	Q	L	D
Key:	H	E	T	H	E	T	H	E	T	H	E	T	H	E	T
Plaintext:	J	I	P	I	H	J	G	S	E	D	A	U	J	H	K
Ciphertext:	I	R	F	S	N	I	J	A	M	G	P	W			
Key:	H	E	T	H	E	T	H	E	T	H	E	T			
Plaintext:	B	N	M	L	J	P	C	W	T	Z	L	D			

We don't see anything terribly promising until we see past the wraparound; then we notice that the putative plaintext has "est" in it. So let's try assuming that's correct. With a keyword length of four we get

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
--- --- --- -wj q-t he- est --- --- --- --- --- --- --- --- --- --- --- ---

--- --- --- -th e-e st- ezr --- --- --- --- --- --- --- --- --- --- --- ---
```

which doesn't look like English. So we try a key length of five:

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
--- --- --- tim --t he- -es t-- --- --- --- --- --- --- --- --- --- --- ---

--- --- --- the --e st- -ns a-- --- --- --- --- --- --- --- --- --- --- ---
```

which looks like it could work. So we expand out another step, keeping in mind that we're guessing the keyword is length five so our keystream doesn't actually go earlier than the sixth character:

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
--- --s o-- tim --t he- -es t-- nsa --- --- --- --- --- --- --- --- --- ---

so- --i m-- the --e st- -ns a-- com --- --- --- --- --- --- --- --- --- ---
```

This still looks good, so we fill out the rest:

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
wa- --s o-- tim --t he- -es t-- nsa --c om- -ic a-- dan --h ea- -we r-- res

so- --i m-- the --e st- -ns a-- com --i ca- -da n-- hea --w er- -re s-- ple
```

At this point we just need to find any part of the message where we can make a guess to fill in the blanks, and we're done. We can try a few things, but perhaps we notice that the last part is "s– ple", which might be "sample" or "simple". Guessing "sample" gives us

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
wa- --s o-- tim --t he- -es t-- nsa --c om- -ic a-- dan --h ea- -we raa res
so- --i m-- the --e st- -ns a-- com --i ca- -da n-- hea fsw era are sam ple
```

which doesn't work very well; but guessing "simple" and working backwards gives us

```
OOF IKA AQW MPQ UMX ZXY IRK TZS PGM GPK QMI PLC NWX KEN QLD IRF SNI JAM GPW
wat ers ome tim est heq ues tio nsa rec omp lic ate dan dth ean swe rsa res

som eti mes the que sti ons are com pli cat eda ndt hea nsw ers are sim ple
```

"Sometimes the questions are complicated, and the answers are simple."

References

[Singh(2015)] Vikram Singh. A practical key exchange for the internet using lattice cryptography. Cryptology ePrint Archive, Report 2015/138, 2015. http://eprint.iacr.org/2015/138.